INSTITUTO TECNOLÓGICO DE ESTUDIOS SUPERIORES DE MONTERREY

CAMPUS SANTA FE

RETO : MOVILIDAD URBANA

MODELACIÓN DE SISTEMAS MULTIAGENTES CON GRÁFICAS COMPUTACIONALES

Pablo Yamamoto A01022382
Gianluca Beltrán A01029098
Javier Corona A01023063

RETO MOVILIDAD URBANA

DIAGRAMA DE LOS AGENTES

AGENTE COCHE

Nuestra primera clase para la solución del reto es la del coche. Al crear una instancia de esté, se le tiene que dar un id único, un destino y una posición inicial, pero tiene varias propiedades como por ejemplo :

- Pasos tomados
- Condición
- Dirección
- Espacios Libres
- Siguientes movimientos

```
class Car(Agent):
    """
    Agent that moves randomly.
    Attributes:
        unique_id: Agent's ID
        direction: Randomly chosen direction chosen from one of eight directions
    """

    def __init__(self, unique_id, model, destination, startPos): …

    def getCarDirections(self, pos): …

    def getFreeSpaces(self, pos): …

    def checkForTrafficLight(self, pos): …

    def checkForOtherCar(self, pos): …

    def move(self): …

    def step(self): …
```

El coche además cuenta con varias funciones para que pueda funcionar de manera correcta y simular lo más natural posible un conductor de la vida real. La función getCarDirecions llena 4 listas con los índices de las direcciones que podemos tomar : arriba, abajo, izquierda y derecha.

```python
def getCarDirections(self, pos):
    self.upIndexes = []
    self.downIndexes = []
    self.leftIndexes = []
    self.rightIndexes = []
    currentPos = pos
    counter = 0
    for step in self.possible_steps:
        if step[0] - currentPos[0] < 0:
            self.leftIndexes.append(counter)
        if step[0] - currentPos[0] > 0:
            self.rightIndexes.append(counter)
        if step[1] - currentPos[1] > 0:
            self.upIndexes.append(counter)
        if step[1] - currentPos[1] < 0:
            self.downIndexes.append(counter)
        counter += 1
```

Nuestra función getFreeSpaces es la que nos permite mirar el ambiente desde la perspectiva del coche, con cada paso que dá ve que objetos hay en cada celda y con base a lo que va encontrando, decide si puede moverse o no (La función toma en cuenta la dirección de las calles). Realizamos cambios en el mapa de la ciudad que explicaremos más adelante, pero esta función toma en cuenta estos cambios (especialmente en las intersecciones y esquinas) para ver si puede moverse a la siguiente celda.

```python
def getFreeSpaces(self, pos):
    self.freeSpaces = []
    currentIndex = 0
    for step in self.possible_steps:
        content = self.model.grid.get_cell_list_contents(step)[0]
        if isinstance(content, Road):
            if not content.twoWay:
                if content.direction == "Up" and currentIndex in self.upIndexes:
                    self.freeSpaces.append(True)
                elif content.direction == "Down" and currentIndex in self.downIndexes:
                    self.freeSpaces.append(True)
                elif content.direction == "Left" and currentIndex in self.leftIndexes:
                    self.freeSpaces.append(True)
                elif content.direction == "Right" and currentIndex in self.rightIndexes:
                    self.freeSpaces.append(True)
                else:
                    self.freeSpaces.append(False)
            else:
                if content.direction[0] == "Up" and currentIndex in self.upIndexes:
                    self.freeSpaces.append(True)
                elif content.direction[0] == "Down" and currentIndex in self.downIndexes:
                    self.freeSpaces.append(True)
                elif content.direction[1] == "Left" and currentIndex in self.leftIndexes:
                    self.freeSpaces.append(True)
                elif content.direction[1] == "Right" and currentIndex in self.rightIndexes:
                    self.freeSpaces.append(True)
                else:
                    self.freeSpaces.append(False)
        elif isinstance(content, Traffic_Light):
            if content.direction == "Up" and currentIndex in self.upIndexes:
                self.freeSpaces.append(True)
            elif content.direction == "Down" and currentIndex in self.downIndexes:
                self.freeSpaces.append(True)
            elif content.direction == "Left" and currentIndex in self.leftIndexes:
                self.freeSpaces.append(True)
            elif content.direction == "Right" and currentIndex in self.rightIndexes:
                self.freeSpaces.append(True)
            else:
                self.freeSpaces.append(False)

        elif isinstance(content, Destination) and content.pos == self.destination:
            self.freeSpaces.append(True)
        else:
            self.freeSpaces.append(False)
        currentIndex += 1
```

Después implementamos dos funciones que regresan valores booleanos que revisan si en alguna de las celdas adyacentes hay instancias de coches o de semáforos.

```python
def checkForTrafficLight(self, pos):
    for content in self.model.grid.iter_cell_list_contents(pos):
        if isinstance(content, Traffic_Light):
            return True
    return False

def checkForOtherCar(self, pos):
    for content in self.model.grid.iter_cell_list_contents(pos):
        if isinstance(content, Car) and content.condition != "Arrived":
            return True
    return False
```

Cuando se instancia un coche, se utiliza el Algoritmo de A* para determinar la ruta más rápida hacia su destino, con cada paso que da el coche, revisa los pasos de este algoritmo y dependiendo de ciertas situaciones lo actualiza o continua con el camino recomendado, por ejemplo, si ya espero más de 10 pasos (esta situación se puede dar por un embotellamiento) entonces recalculo el camino más corto desde su posición actual hasta su destino. Cuando llegue a su destino, su condición pasa de estar manejando a completado, entonces el vehículo ya no se moverá más.

```python
def step(self):
    """
    Determines the new direction it will take, and then moves
    """
    print(f"Agente: {self.unique_id} movimiento {self.steps_taken}")
    if self.condition == "Driving":
        if self.pos != self.destination:
            # self.move()
            next_position = self.path[self.steps_taken]
            if self.checkForOtherCar(next_position):
                if self.stepsWaited > 10:
                    print(
                        f"RECALCULATING (Waited for more than 10 steps for car to move), Agent: {self.unique_id}")
                    self.path = algorithm(
                        self, self.pos, self.destination, h)
                else:
                    print(
                        f"Theres a car where I want to go!!, I'll wait here for a sec, Agent: {self.unique_id}")
                    self.stepsWaited += 1
            elif self.checkForTrafficLight(next_position):
                if not self.model.grid.get_cell_list_contents(next_position)[0].state:
                    print(
                        f"Traffic light is red, I need to wait for green, Agent: {self.unique_id}")
                else:
                    self.model.grid.move_agent(self, next_position)
                    self.steps_taken += 1
                    self.stepsWaited = 0
            else:
                self.model.grid.move_agent(self, next_position)
                self.steps_taken += 1
                self.stepsWaited = 0

        else:
            print(
                f"Arrived to destination at: {self.pos}!!!, Agent: {self.unique_id}")
            self.condition = "Arrived"
            self.model.complete_trips += 1
```

AGENTE SEMÁFORO

Nuestro semáforo cambia cada 10 segundos (o pasos) y uno de los cambios que agregamos es que tiene dirección, esto es para no interrumpir el flujo de las calles, especialmente en los cruces y esquinas.

```python
class Traffic_Light(Agent):
    """
    Obstacle agent. Just to add obstacles to the grid.
    """

    def __init__(self, unique_id, model, direction, state=False, timeToChange=10):
        super().__init__(unique_id, model)
        self.state = state
        self.timeToChange = timeToChange
        self.condition = ""
        self.direction = direction

    def step(self):
        # if self.model.schedule.steps % self.timeToChange == 0:
        #     self.state = not self.state
        pass
```

AGENTE CALLE

La calle como mencionamos tiene una dirección pero también puede ser que en un cruce, un vehículo quiera seguir el rumbo que ya llevaba o cambiar de calle, por eso mismo puede tener dos direcciones.

```python
class Road(Agent):
    """
    Obstacle agent. Just to add obstacles to the grid.
    """

    def __init__(self, unique_id, model, twoWay, direction="Left"):
        super().__init__(unique_id, model)
        self.condition = ""
        self.direction = direction
        self.twoWay = twoWay

    def step(self):
        pass
```

AGENTE OBSTÁCULO

Este agente impide que los vehículos puedan salir del grid. Lo usamos para representar los edificios.

```
class Obstacle(Agent):
    """
    Obstacle agent. Just to add obstacles to the grid.
    """

    def __init__(self, unique_id, model):
        super().__init__(unique_id, model)
        self.condition = ""

    def step(self):
        pass
```

AGENTE DESTINO

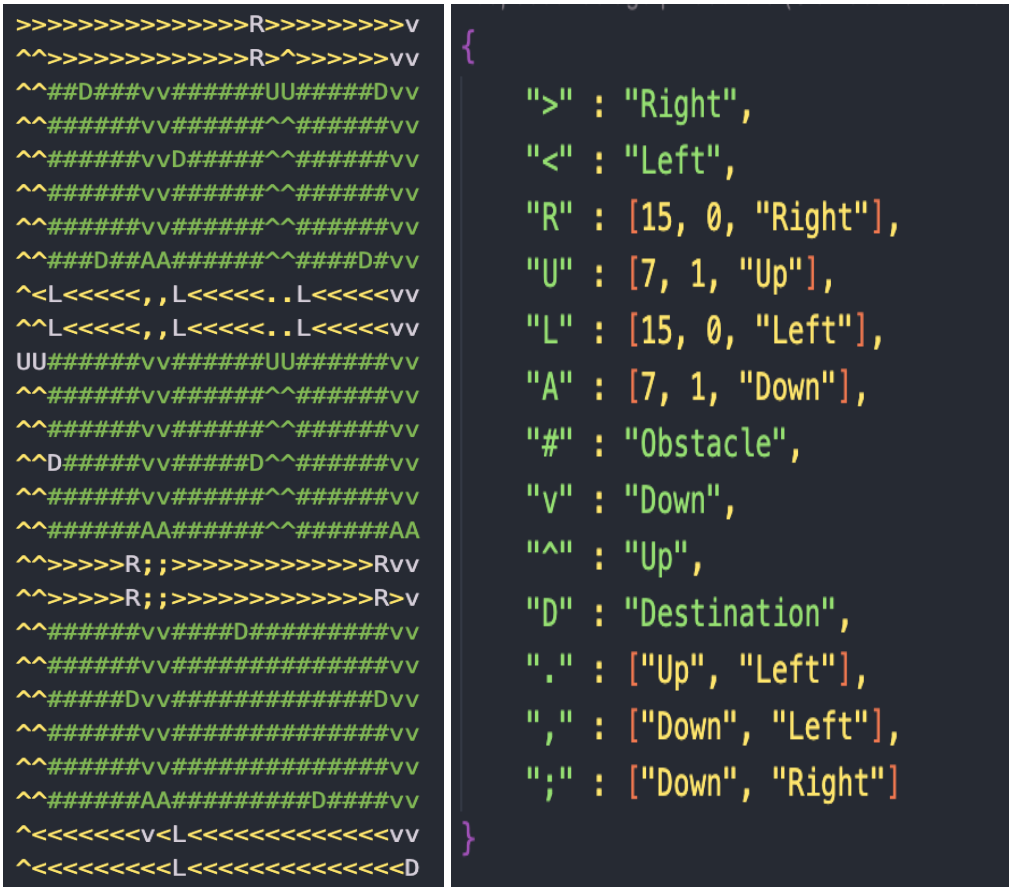
Una vez que llega al destino, cada vehículo se detiene en donde está su respectivo objetivo..

```
class Destination(Agent):
    """
    Obstacle agent. Just to add obstacles to the grid.
    """

    def __init__(self, unique_id, model):
        super().__init__(unique_id, model)
        self.condition = ""

    def step(self):
        pass
```

CIUDAD



```
>>>>>>>>>>>>>>>R>>>>>>>>>>v
^^>>>>>>>>>>>>>>R>^>>>>>>vv
^^##D###vv######UU#####Dvv
^^######vv######^^######vv
^^######vvD#####^^######vv
^^######vv######^^######vv
^^######vv######^^######vv
^^###D##AA######^^####D#vv
^<L<<<<<,,L<<<<<..L<<<<<vv
^^L<<<<<,,L<<<<<..L<<<<<vv
UU######vv######UU######vv
^^######vv######^^######vv
^^######vv######^^######vv
^^D#####vv#####D^^######vv
^^######vv######^^######vv
^^######AA######^^######AA
^^>>>>>R;;>>>>>>>>>>>>>Rvv
^^>>>>>R;;>>>>>>>>>>>>>R>v
^^######vv####D########vv
^^######vv##############vv
^^#####Dvv#############Dvv
^^######vv##############vv
^^######vv##############vv
^^######AA########D####vv
^<<<<<<<v<L<<<<<<<<<<<<vv
^<<<<<<<<<L<<<<<<<<<<<<<D
```

```
{
    ">" : "Right",
    "<" : "Left",
    "R" : [15, 0, "Right"],
    "U" : [7, 1, "Up"],
    "L" : [15, 0, "Left"],
    "A" : [7, 1, "Down"],
    "#" : "Obstacle",
    "v" : "Down",
    "^" : "Up",
    "D" : "Destination",
    "." : ["Up", "Left"],
    "," : ["Down", "Left"],
    ";" : ["Down", "Right"]
}
```

Como mencionamos anteriormente hicimos varias modificaciones a la organización de la ciudad para que los coches puedan transitar en esta de mejor manera. Para empezar, en la esquina superior izquierda, las direcciones de la calle en la esquina ya no apuntan a la izquierda, sino hacia arriba, esto para que puedan dar la vuelta debidamente y se eviten colisiones con los edificios. También añadimos varios términos al diccionario que lee python, “R”, “U”, “L”, “A” indica la dirección de la calle del semáforo, R = right, U = up, L = left y A = abajo, porque D se utilizó para los destinations, los quinces y sietes representan el tiempo que tarda cada semáforo en cambiar de estado y los unos y ceros representan el estado inicial del semáforo. Decidimos hacer este cambio para poder mantener la lógica de direccionamiento que utilizamos para las calles.

Como se pueden tener varias direcciones en un mismo camino, prioritariamente en intersecciones, añadimos esto de igual manera. “.” Indica un camino en el cual puedes seguir subiendo o girar a la izquierda. “,” Indica un camino en el cual puedes seguir bajando o girar a la izquierda. “;” Indica un camino en el cual puedes seguir bajando o girar a la derecha.

INSTALACIÓN

Prerrequisitos :

- Git
- Python (versión 3 en adelante)
- Mesa
- Flask
- Unity
- Cloudant

Paso 1.- Clonar el repositorio del equipo en donde se deseé guardar (Puede ser en el escritorio)

Link:
https://github.com/JavierCoronaA01023063/Modelacion-de-sistemas-multiagentes-con-graficas-computacionales---Equipo-4.git

Paso 2.- Abrir Unity Hub

Paso 3.- Añadir la carpeta del modelo de Unity como un nuevo proyecto, en este caso la carpeta se llama Test

Paso 4.- Picar play en el editor de Unity, el modelo debería correr solo, ya que el servidor está en la nube de IBM

Paso 5 (Opcional).- Si se quiere correr el servidor localmente, navegar a Actividad Integradora/CarAgents dentro del repositorio y correr el script flask_server.py, también se debe cambiar el url en el editor de Unity en el objeto AgentController, al url que flask imprima en la terminal.