



Modelación de sistemas multiagentes con gráficas computacionales (Gpo 301)

2 dic 2022



Evidencia 2. Avances y presentación del reto

Juan Daniel Rodríguez Oropeza A01411625
Sergio Hiroshi Carrera A01197964
Jesús Sebastián Jaime Oviedo A01412442
Jackeline Conant A01280544
William Frank Monroy Mamani A00829796
Premsyl Pilar A01760915





Movilidad urbana

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.





Intersección: escenario a modelar

En un cruce vehicular, se simulará la intersección vial de varios agentes asignados como vehículos automotores.

Otros conceptos más



Agente Inteligente

Agente que percibe su entorno y actúa conforme a ello.



Reflejos simples

Agente que no evoluciona, reglas fijas.
“El carro en luz roja, se detiene”



C#

Lenguaje de programación creado por Microsoft, pilar en el desarrollo de videojuegos en Unity.



01

Programación

Multiagentes en **Mesa** en Python

Definición

Un **sistema multiagente** se componen múltiples agentes que interactúan entre ellos, con el fin de resolver un problema difícil que uno solo no podría (Quiroz, 2022)

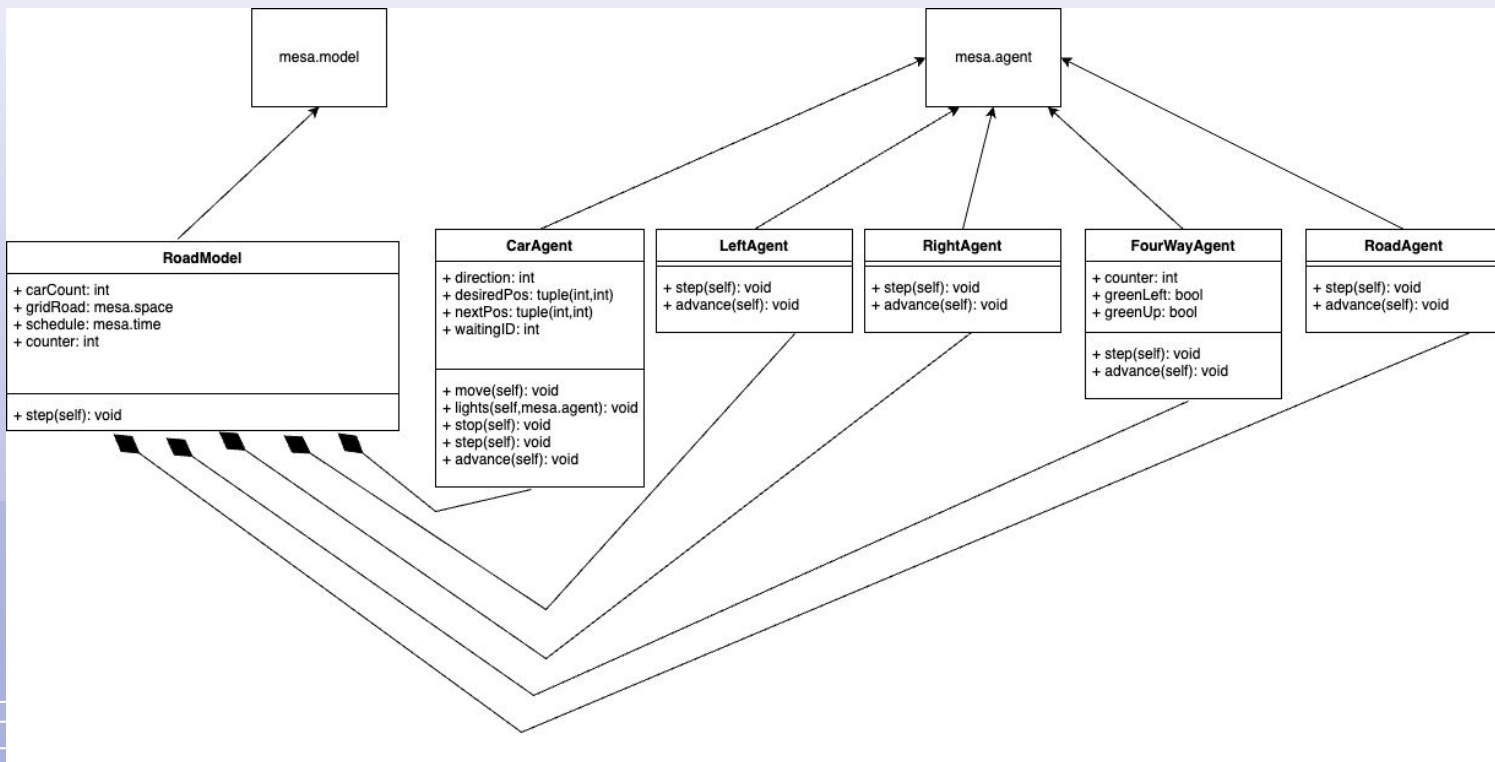


MESA

Los modelos basados en agentes son simulaciones informáticas en las que intervienen múltiples entidades (los agentes) que actúan e interactúan entre sí en función de su comportamiento programado. Los agentes pueden utilizarse para representar células vivas, animales, seres humanos individuales, incluso organizaciones enteras o entidades abstractas.



Diagrama de clases – Mesa Py





Modelos





Modelo de rutas

```
class RoadModel(mesa.Model):
    def __init__(self, roadMap, numberOfCars):
        self.carCount = 0
        self.gridRoad = mesa.space.MultiGrid(len(roadMap), len(roadMap[0]), False)
        self.schedule = mesa.time.SimultaneousActivation(self)
        self.counter = 0
        self.wantedCars = numberOfCars
        for i in range(len(roadMap)):
            for j in range(len(roadMap[i])):
                # Make the traffic light first so it is called in scheduler first and does not change between phases
                # ID is 0
                if roadMap[i][j] == 4:
                    agent = FourWayAgent(self.counter, self)
                    self.schedule.add(agent)
                    self.counter += 1
                    self.gridRoad.place_agent(agent, (i,j))

        # Load map
        for i in range(len(roadMap)):
            for j in range(len(roadMap[i])):
                if roadMap[i][j] == 1:
                    agent = RoadAgent(self.counter, self)
                    self.schedule.add(agent)
                    self.counter += 1
                    self.gridRoad.place_agent(agent, (i,j))
                if roadMap[i][j] == 3:
                    agent = LeftAgent(self.counter, self)
                    self.schedule.add(agent)
                    self.counter += 1
                    self.gridRoad.place_agent(agent, (i,j))
                if roadMap[i][j] == 2:
                    agent = RightAgent(self.counter, self)
                    self.schedule.add(agent)
                    self.counter += 1
                    self.gridRoad.place_agent(agent, (i,j))

    def step(self):
        self.schedule.step()
        if self.carCount < self.wantedCars:
            agent = CarAgent(self.counter, self)
            agent.direction = Direction.RIGHT
            self.gridRoad.place_agent(agent, (0,0))
            self.schedule.add(agent)
            self.counter += 1
            self.carCount += 1
```




Agentes


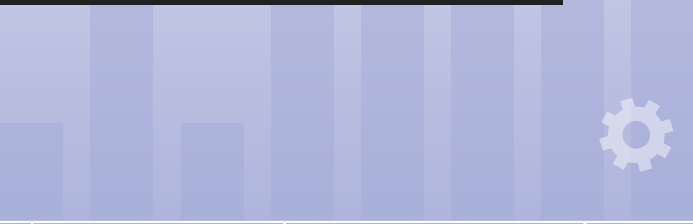




Agente para la ruta



```
class RoadAgent(mesa.Agent):  
    def __init__(self, unique_id, model):  
        super().__init__(unique_id, model)  
  
    def step(self):  
        pass  
  
    def advance(self):  
        pass
```





Agente para la ruta izquierda

```
class LeftAgent(mesa.Agent):  
    def __init__(self, unique_id, model):  
        super().__init__(unique_id, model)  
    def step(self):  
        pass  
    def advance(self):  
        pass
```










Agente para la ruta derecha

```
class RightAgent(mesa.Agent):  
    def __init__(self, unique_id, model):  
        super().__init__(unique_id, model)  
    def step(self):  
        pass  
    def advance(self):  
        pass
```






Agente para la ruta intersección



```
# Agent that represents intersection in our model, it supports only one ways and straight movement
class FourWayAgent(mesa.Agent):
    def __init__(self, unique_id, model):
        super().__init__(unique_id, model)
        self.greenLeft = False
        self.greenUp = True
        self.counter = 0

    # During step it counts ticks and changes every few ticks and there is no space in between.
    # Also traffic lights go first in simulation so it does not change between step and advance for other agents
    def step(self):
        self.counter += 1
        if self.counter == trafficLightTime:
            if self.greenLeft:
                self.greenLeft = False
            else:
                self.greenLeft = True
            if self.greenUp:
                self.greenUp = False
            else:
                self.greenUp = True
            self.counter = 0

    def advance(self):
        pass
```



Agente para el carro

```
# Represents car in our model. It contains baseline members and direction which it is currently going, desired position
# which is position where it wants to go, next position is the actual position it will take next tick and it can remain
# in the place and waitID which is ID of agent that is blocking the path.
class CarAgent(mesa.Agent):
    def __init__(self, unique_id, model):
        super().__init__(unique_id, model)
        self.direction = Direction.NONE
        self.desiredPos = (0,0)
        self.nextPos = (0,0)
        self.waitingID = -1

# Based on current direction it evaluates desired position
    def move(self):
        self.waitingID = -1
        if self.direction == Direction.UP:
            self.desiredPos = (self.pos[0],self.pos[1]+1)
        if self.direction == Direction.DOWN:
            self.desiredPos = (self.pos[0],self.pos[1]-1)

        if self.direction == Direction.LEFT:
            self.desiredPos = (self.pos[0]-1,self.pos[1])

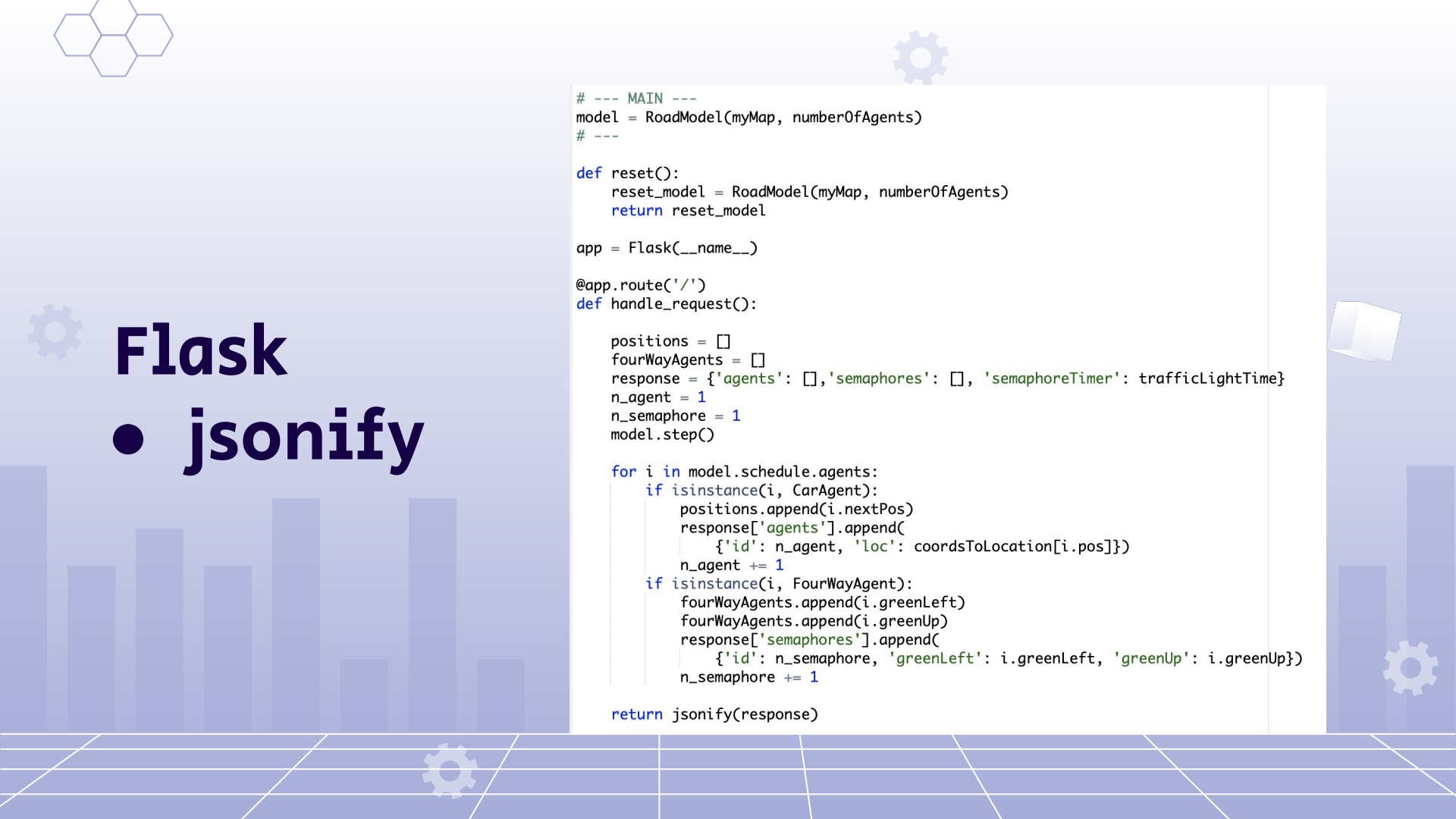
        if self.direction == Direction.RIGHT:
            self.desiredPos = (self.pos[0]+1,self.pos[1])

# When agent encounters traffic lights it needs to behave correctly, either stop or go
    def lights(self,agent):
        if agent.greenUp and (self.direction == Direction.UP or self.direction == Direction.DOWN):
            self.nextPos = self.desiredPos
        elif agent.greenLeft and (self.direction == Direction.LEFT or self.direction == Direction.RIGHT):
            self.nextPos = self.desiredPos
        else:
            self.waitingID = agent.unique_id
            self.nextPos = self.pos
```




API





Flask

- jsonify

```
# --- MAIN ---
model = RoadModel(myMap, numberOfAgents)
# ---

def reset():
    reset_model = RoadModel(myMap, numberOfAgents)
    return reset_model

app = Flask(__name__)

@app.route('/')
def handle_request():

    positions = []
    fourWayAgents = []
    response = {'agents': [], 'semaphores': [], 'semaphoreTimer': trafficLightTime}
    n_agent = 1
    n_semaphore = 1
    model.step()

    for i in model.schedule.agents:
        if isinstance(i, CarAgent):
            positions.append(i.nextPos)
            response['agents'].append(
                {'id': n_agent, 'loc': coordsToLocation[i.pos]})
            n_agent += 1
        if isinstance(i, FourWayAgent):
            fourWayAgents.append(i.greenLeft)
            fourWayAgents.append(i.greenUp)
            response['semaphores'].append(
                {'id': n_semaphore, 'greenLeft': i.greenLeft, 'greenUp': i.greenUp})
            n_semaphore += 1

    return jsonify(response)
```

Matriz - Mesa Py

0 - Bloqueado
1 - Camino Normal
2 - Derecha
3 - Izquierda
4 - Intersección

Python Array

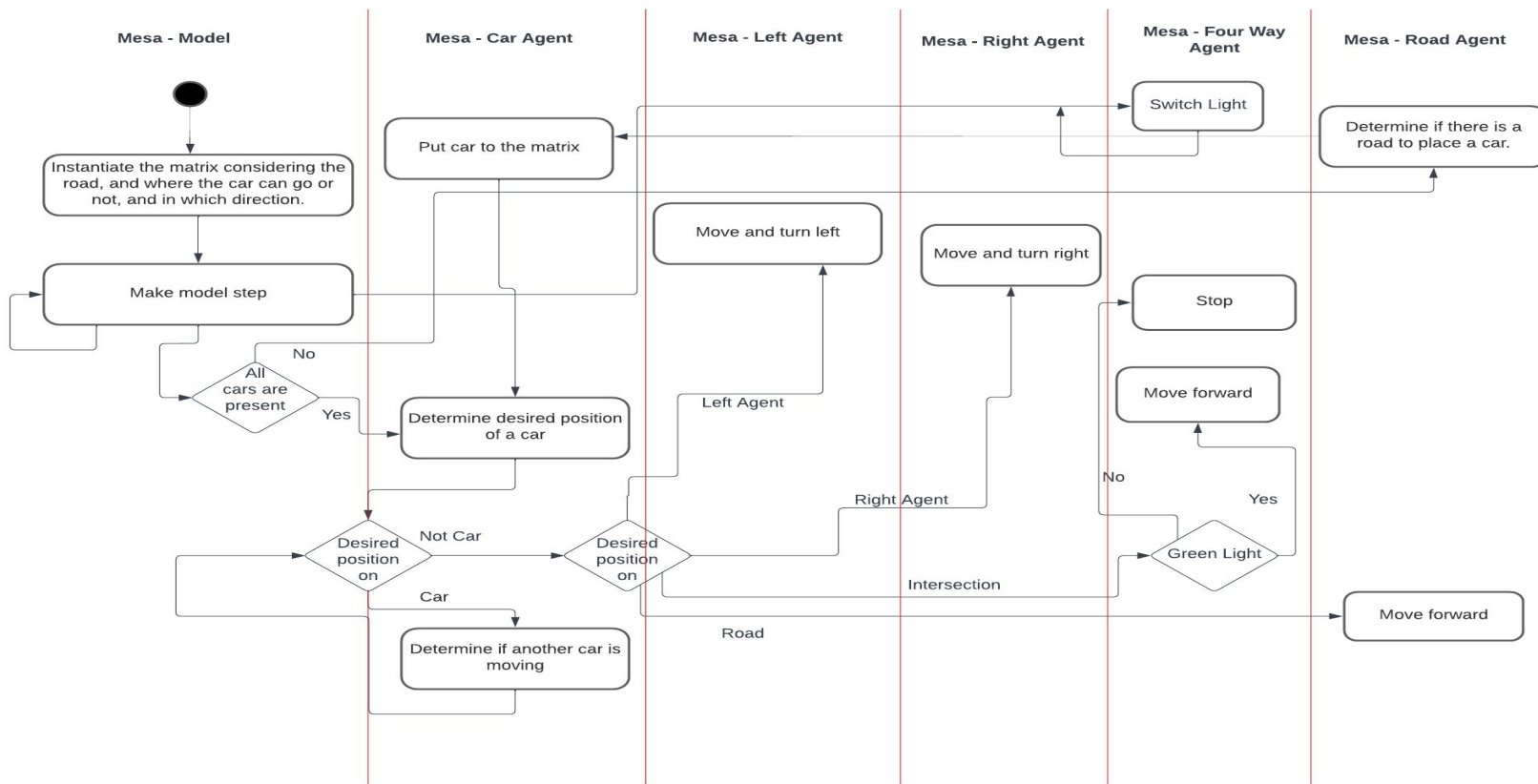
```
00 01 02  
10 11 12  
20 21 22
```

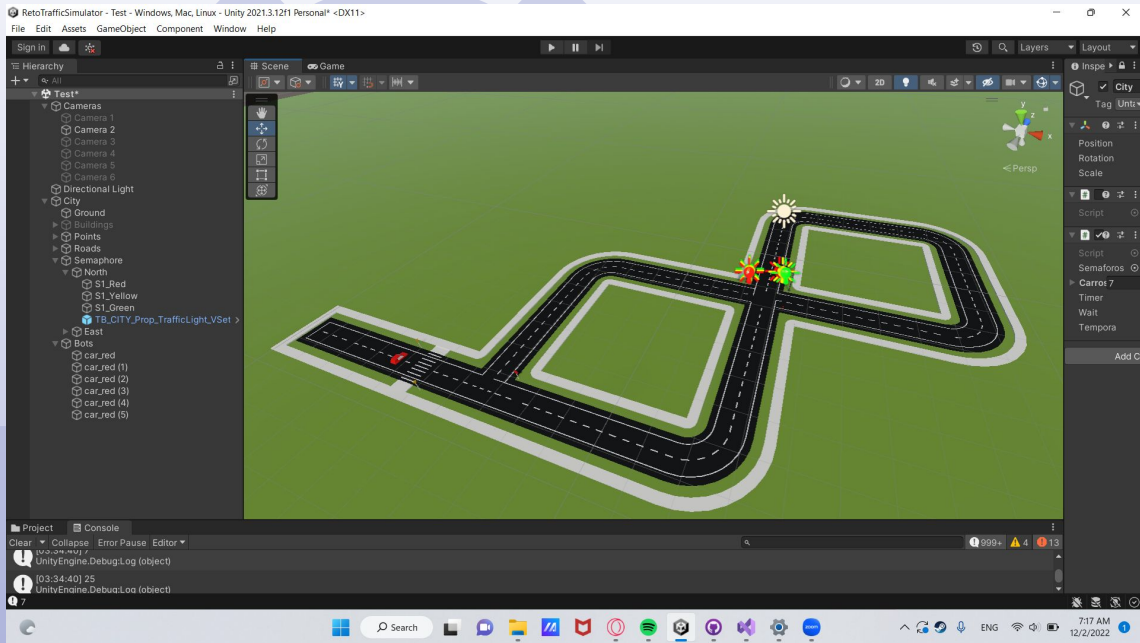
Mesa Grid

```
02 12 22  
01 11 21  
00 10 20
```

```
myMap = [[3, 1, 1, 1, 3, 1, 1, 1, 2],  
          [1, 0, 0, 0, 1, 0, 0, 0, 1],  
          [1, 0, 0, 0, 1, 0, 0, 0, 1],  
          [1, 0, 0, 0, 1, 0, 0, 0, 1],  
          [3, 1, 1, 1, 4, 1, 1, 1, 2],  
          [1, 0, 0, 0, 1, 0, 0, 0, 1],  
          [1, 0, 0, 0, 1, 0, 0, 0, 1],  
          [1, 0, 0, 0, 1, 0, 0, 0, 1],  
          [3, 1, 1, 1, 2, 1, 1, 1, 2]]
```

```
myMap = [[3, 1, 1, 1, 3, 1, 1, 1, 2],  
          [1, 0, 0, 0, 1, 0, 0, 0, 1],  
          [1, 0, 0, 0, 1, 0, 0, 0, 1],  
          [1, 0, 0, 0, 1, 0, 0, 0, 1],  
          [3, 1, 1, 1, 4, 1, 1, 1, 2],  
          [1, 0, 0, 0, 1, 0, 0, 0, 1],  
          [1, 0, 0, 0, 1, 0, 0, 0, 1],  
          [1, 0, 0, 0, 1, 0, 0, 0, 1],  
          [3, 1, 1, 1, 2, 1, 1, 1, 2]]
```





Interfaz Grafica

MADE
WITH



Unity®

Diagrama de clase - Unity

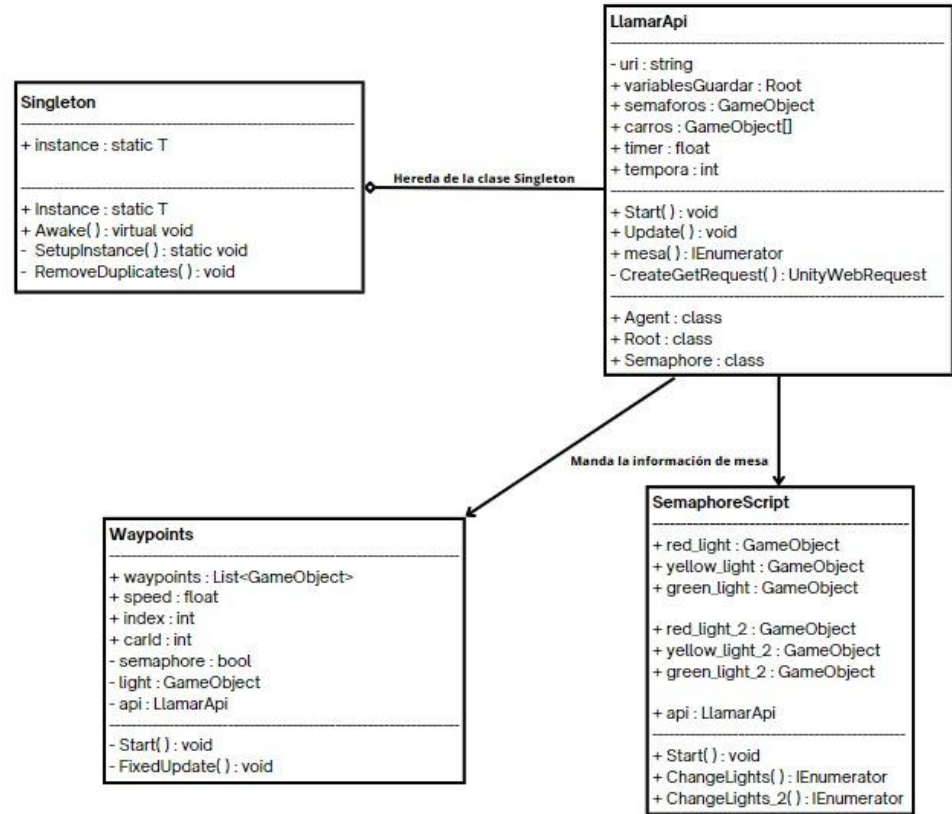
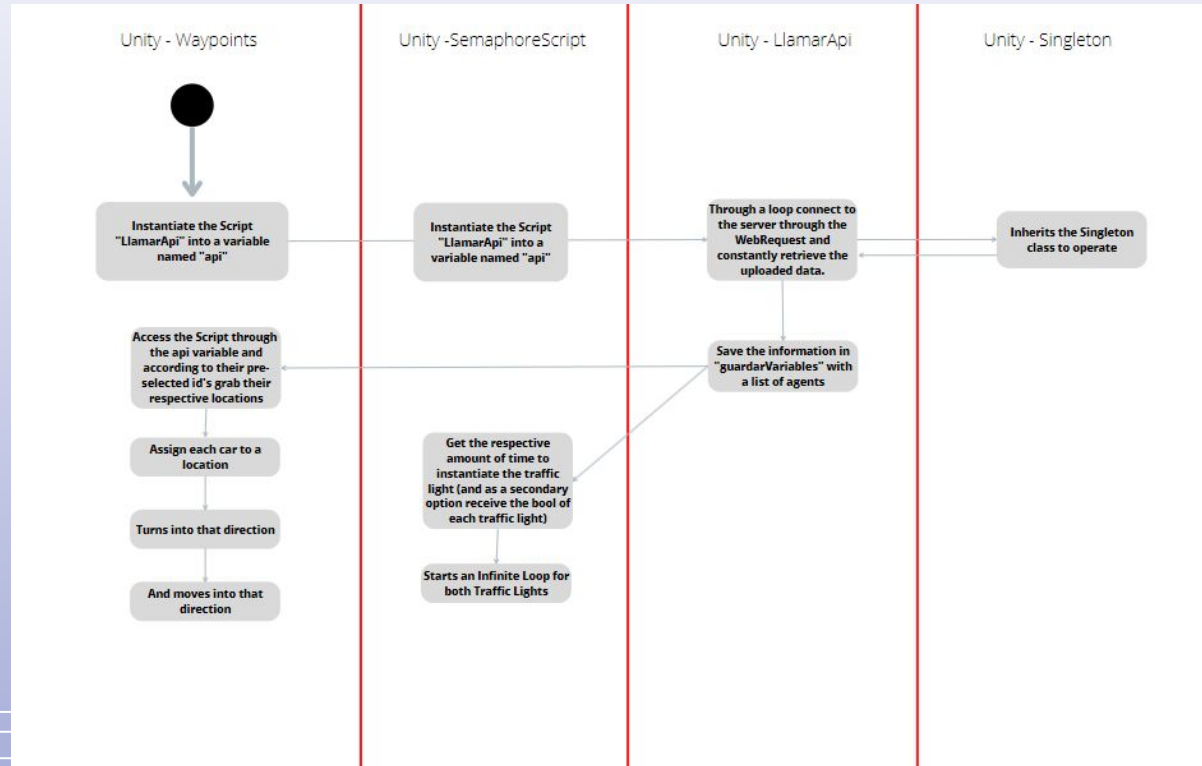




Diagrama de interacciones – Unity



Diseño del escenario

Tenemos un cruce vehicular donde automóviles multicolores van a cruzar, estos autos serán dirigidos por un sistema multiagente para evitar choques y cruzar apropiadamente dentro de la intersección.





API por parte de Unity

Por medio de la librería Newtonsoft y al de-jsonificar.

Esto lo aprendimos en la tarea 3DBall.

```
public class Agent
{
    public int id { get; set; }
    public Position position { get; set; }
}

public class Position
{
    public int x { get; set; }
    public int y { get; set; }
}

public class Root
{
    public List<Agent> agents { get; set; }
    public int semaphoreTimer { get; set; }
    public List<Semaphore> semaphores { get; set; }
}

public class Semaphore
{
    public bool greenLeft { get; set; }
    public bool greenUp { get; set; }
    public int id { get; set; }
}

public class LlamarApi : Singleton<LlamarApi>
{
    private string uri = "http://chiron.pythonanywhere.com/";
```



Veamoslo en acción

Demostración del reto



**¡Mil
gracias!**





Bibliografía

- Quiroz, A. (2022, May 26). Sistemas multiagente: qué son y cómo funcionan - B2Chat. B2Chat. <https://www.b2chat.io/blog/b2chat/sistemas-multiagente-que-son-como-funcionan/#:~:text=Un%20sistema%20multiagente%20es%20b%C3%A1sicamente,individual%20o%20un%20sistema%20monol%C3%ADtico.>
- Project Mesa Team (2022, Dec 1). Mesa Overview. <https://mesa.readthedocs.io/en/latest/overview.html>

