TareaM3

November 28, 2021

1 Tarea M3

Se uso la librería **AgentPy, Matplotlib, e IPython** para modelar y mostrar un cruce de 2 calles usando un sistema multiagentes.
De forma resumida se cuentan con **2 clases de tipo agente**, los agentes semáforos y los agentes coches; **1 clase de tipo modelo** la cual contiene los agentes y el entorno donde van a interactuar.

[ ]:
```
#Libraries
import agentpy as ap
import matplotlib.pyplot as plt
import IPython
```

1.1 Clase Agente Semáforo

En la función `setup()` se declaran e inicializan las variables a usar en el agente.
En la función `run()` se pone la lógica para el control de las luces y es la función que será llamada en cada paso del modelo.
Observaciones:
- `setup()` recibe "timeColor" que es la duracion que tendrá cada color.

[ ]:
```
class AgenteLight(ap.Agent):
    def setup(self, timeColor):
        #0 rojo, 1 verde, 2 amarillo
        self.color = 0
        self.direction = [1,0]
        self.time = timeColor
        self.timer = 0
    def run(self):
        if self.timer >= self.time:
            self.color += 1
            self.timer =0
        if self.color > 2:
            self.color = 0
        self.timer += 1
```

1.2 Clase Agente Carro

En la función `setup()` se declaran e inicializan las variables a usar en el agente.
En la función `run()` se revisa la luz del semáforo para actualizar la velocidad a la que debe de ir el

coche, después el coche se mueve según la velocidad y dirección a la que va.
En la función `checkLight()` se revisa de todos los semáforos cual es el que le corresponde a ese coche, después revisa si el coche ha pasado la posición del semáforo o no, si no lo ha pasado, regula su velocidad según el color del semáforo, pero si ya pasó al semáforo va a la velocidad máxima.
Observaciones:
- `setup()` recibe "maxVel" que es la velocidad máxima que tendrá el coche.

[ ]:
```
class AgenteCarro(ap.Agent):
    def setup(self, maxVel):
        self.MaxVelocity = maxVel
        self.velocity = 0
        self.direction = [1,0]
    def run(self):
        self.checkLight()
        self.model.street.move_by(self, [self.velocity * self.direction[0],␣
↪self.velocity * self.direction[1]])
    def checkLight(self):
        for light in self.model.lights:
            if light.direction == self.direction:
                if self.model.street.positions[light] > self.model.street.
↪positions[self]:
                    if light.color == 0:
                        self.velocity = 0
                    elif light.color == 1:
                        self.velocity = self.MaxVelocity
                    else:
                        self.velocity = int(self.velocity/2)
                else:
                    self.velocity = self.MaxVelocity
```

1.3 Clase Modelo Interseccion

En la función `setup()` se declaran e inicializan las variables a usar en el modelo, se crea el entorno y se le añaden los agentes creados en las posiciones especificadas.
En la función `step()` es la función que será llamada en cada paso del modelo, y en esta se llama a las funciones de los agentes.
Observaciones:
- Al crear la lista de agentes semáforo se pasa la duración de cada color que se puso en los parámetros del modelo.
- Al segundo agente de la lista de agentes semáforo se le cambia su dirección y el color en el que comienza.
- Al crear la lista de agentes carro se pasa la velocidad máxima que se puso en los parámetros del modelo.
- A la mitad de los coches creados se les cambia de dirección.
- Se cren variables locales que se usan para calcular las posiciones iniciales de los agentes.

[ ]:
```
class Interseccion(ap.Model):
    def setup(self):
        self.lights = ap.AgentList(self, 2, AgenteLight, timeColor=self.p.
↪colorTime)
        self.lights[1].direction = [0,1]
        self.lights[1].color = 2

        self.carros = ap.AgentList(self, self.p.carros, AgenteCarro,␣
↪maxVel=self.p.maxVelocity)

        for i in range(int(self.p.carros/2), self.p.carros):
            self.carros[i].direction = [0,1]

        half = int(self.p.size/2)
        separation = int((half-10)/int(self.p.carros/2))
        validPos = range(separation, half-5, separation)

        carsPos = [(x, half-5) for x in validPos] + [(half-5, y) for y in␣
↪validPos]
        lightsPos = [[half, half-5],[half-5, half]]

        self.street = ap.Grid(self, [self.p.size, self.p.size], torus=True)

        self.street.add_agents(self.lights, lightsPos)
        self.street.add_agents(self.carros, carsPos)

    def step(self):
        self.lights.run()
        self.carros.run()
```

1.4 Parámetros

Se definen los parámetros que usará nuestro modelo.
- **step_time**: el tamaño de cada paso.
- **size**: que tamaño tendrá X, Y del entorno.
- **steps**: cantidad máxima de pasos que puede durar el modelo.
- **carros**: cantidad de coches en total (horizontales + verticales) que tendrá el modelo.
- **colorTime**: cantidad de pasos que va a durar cada color del semáforo.
- **maxVelocity**: velocidad máxima a la que pueden ir los coches.

[ ]:
```
parameters = {
    'step_time': 1,
    'size': 500,
    'steps': 100,
    'carros':6,
    'colorTime':10,
    'maxVelocity': 10
```

```
}
```

1.5 Funciones para graficar la simulación

1.5.1 `hacerPlot()`

- En cada paso se hace una gráfica con la posición de los coches, de los semáforos y sus colores. ### `hacerAnimacion()`

- Crea la animación juntando cada plot como un frame.

[ ]:
```
def hacerPlot(m, ax):
    ax.set_title(f"Avenida t={m.t*m.p.step_time:.2f}")

    colors = ["red", "green", "yellow"]

    pos_s1 = m.street.positions[m.lights[0]]
    ax.scatter(*pos_s1, s=20, c=colors[m.lights[0].color])
    pos_s2 = m.street.positions[m.lights[1]]
    ax.scatter(*pos_s2, s=20, c=colors[m.lights[1].color])

    ax.set_xlim(0, m.street.shape[0])
    ax.set_ylim(0, m.street.shape[1])

    for car in m.carros:
        pos_c = m.street.positions[car]
        ax.scatter(*pos_c, s=20, c="black")

    ax.set_axis_off()
    ax.set_aspect('equal', 'box')

def hacerAnimacion(m, p):
    fig = plt.figure(figsize=(10, 10))
    ax = fig.add_subplot(111)
    animation = ap.animate(m(p), fig, ax, hacerPlot)
    return IPython.display.HTML(animation.to_jshtml(fps=20))
```

1.5.2 Se llama a la función `hacerAnimacion()` para correr todo y crear la animación basada en el modelo con los párametros dados.

[ ]:
```
hacerAnimacion(Interseccion, parameters)
```

[ ]:
```
<IPython.core.display.HTML object>
```