Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Querétaro

TC2008B. Modelación de sistemas multiagentes con gráficas computacionales

M1. Actividad

Profesores:

Pedro Oscar Pérez Murueta

Denisse Lizbeth Maldonado Flores

Alejandro Fernández Vilchis

Presenta:

José Emiliano Riosmena Castañón – A01704245

Fecha:

Domingo, 12 de noviembre del 2023

Implementación

Para poder modelar nuestros robots de limpieza, es necesario conocer los comportamientos básicos que tendrían estos robots.

- Limpiar: Si el agente se encuentra una celda sucia limpiarla
- Moverse: Si el agente no se encuentra en una celda sucia, moverse a otra celda
- Resolver conflictos: Si una celda esta ocupada por otro agente, entonces este deberá buscar otra celda disponible

Dicho lo anterior, vamos a proceder con la importación de las librerías que necesitamos para modelar nuestros robots.

```
# Mesa imports
from mesa import Agent, Model
from mesa.time import RandomActivation
from mesa.space import MultiGrid
from mesa.datacollection import DataCollector

# Statistic libraries imports
import numpy as np
import pandas as pd

# Other imports
import time
import datetime
import random
```

La primera sección de las librerías a importar, son del conjunto de la librería mesa, las cuales son la base para poder construir nuestro agente, en ella tenemos desde al propio agente y el modelo, la secuencia de activación, el espacio en el que se encontrara y los datos que el agente nos va a mostrar.

La segunda sección son las librerías para realizar cálculos y para poder convertir los datos que nos entrega el agente en información, la cual nos permitirá interpretar el comportamiento y los resultados de la simulación

Y por último, tenemos otras librerías que nos van a servir para medir el tiempo que le toma a los agentes terminar sus actividades y también para permitirles a los agentes tomar sus propias decisiones.

Una vez que tenemos definidas nuestras librerías y tenemos listas nuestras herramientas con las que vamos a construir esta simulación. Entonces podemos empezar a definir nuestra primera clase, la cual es el agente. En este caso lo definiremos como *CleaningAgent*. Sus parámetros son los siguientes:

- Id: El identificador único para cada agente
- Model: El modelo en el cual el agente estará trabajando y donde se basará su comportamiento.

Para la clase del agente, vamos a definir sus siguientes métodos:

- move(): En este le permitiremos al agente poder moverse a otra posición en el espacio
- step(): Este método es muy importante, porque aquí realizaremos cada iteración en el comportamiento del agente, en donde implicara el movimiento del agente, o si tiene que limpiar la celda en la que se encuentra.

```python
class CleaningAgent(Agent):
    def __init__(self, id, model):
        super().__init__(id, model)

    def move(self):
        # Get neighboring positions (Moore neighborhood) excluding the current position
        possible_steps = self.model.grid.get_neighborhood(self.pos, moore=True, include_center=False)
        # Filter out positions that are not empty
        possible_steps = [i for i in possible_steps if self.model.grid.is_cell_empty(i)]

        # Check if there are any possible steps
        if possible_steps:
            # Choose a random position from the list of possible steps
            new_position = random.choice(possible_steps)
            # Move the agent to the new position
            self.model.grid.move_agent(self, new_position)

    def step(self):
        # Get current position
        x, y = self.pos

        # Check if the current position is clean
        if self.model.is_clean(x, y):
            # If clean, move to a new position
            self.move()
        else:
            # If not clean, clean the current position and then move to a new position
            self.model.clean(x, y)
            self.move()
```

Ya tenemos modelado a nuestro agente, pero ahora como vamos a determinar su comportamiento para esta simulación. Pues para eso, construimos el modelo en el cual trabajara el agente. Este se llamará *CleaningModel.* Aquí determinaremos los siguientes parámetros:

- Ancho: Aquí simplemente le pasamos las medidas del ancho del escenario en el cual trabajaran nuestros agentes
- Largo: Similar al anterior, le pasamos las medidas del largo del escenario en donde trabajaran nuestros agentes
- Número de agentes: Aquí definimos el número de agentes que trabajaran en una simulación, en donde dependiendo de su número, el resultado de la simulación cambiara.
- Celdas sucias: Aquí definimos como un porcentaje, que tanto del escenario estará cubierto con celdas sucias. En donde mientras más agreguemos, más pasos y tiempo les tomara a nuestros agentes completar la tarea o llegar al límite de pasos permitidos

Del mismo modo, vamos a definir los métodos en los que trabajara el agente:

- clean(): La principal función del agente, limpiar la celda en la que se encuentra y actualizar el número de celdas limpias
- is_clean(): Función para identificar si la celda en la que se encuentra el agente esta limpia o esta sucia.
- random_empty_cell(): Seleccionar una celda aleatoria para moverse, siempre y cuando no este ocupada por otro agente.
- random_pos(): Seleccionar una celda aleatoria siempre y cuando este dentro de los límites del escenario
- get_steps(): Obtener constantemente el número de pasos permitidos a los agentes, para saber cuanto llevan y si ya se pasaron del límite
- get_dirty_cells(): Obtener en número inicial de celdas sucias y actualizarse conforme se vayan limpiando las celdas
- get_clean_sells(): Obtener el número inicial de celdas limpias y actualizarse conforme se vayan limpiando las celdas
- get_dirty_percentage(): Obtener el porcentaje inicial de celdas sucias y actualizarse hasta que llegue a 0 o hasta que se alcance el límite de pasos permitidos
- step(): Similar a la clase del agente, aquí realizaremos cada iteración del agente en donde este se estarán realizando las funciones mencionadas anteriormente.

```python
class CleaningModel(Model):
    def __init__(self, width, height, num_agents, dirty_cells):
        # Initialize the grid and scheduler
        self.grid = MultiGrid(width, height, torus=False)
        self.schedule = RandomActivation(self)

        # Initialize counters for dirty and clean cells, steps, and set the number of dirty cells
        self.dirty = np.zeros((width, height))
        self.dirty_cells = 1
        self.clean_cells = 0
        self.steps = 0

        # Initialize agent IDs starting from 0
        id = 0

        # Create CleaningAgent instances, add them to the schedule, and place them on the grid
        for i in range(num_agents):
            agent = CleaningAgent(id, self)
            self.schedule.add(agent)
            self.grid.place_agent(agent, (1, 1))
            id += 1

        # Calculate the total number of cells and set the number of dirty cells
        total_cells = width * height
        self.dirty_cells = int(total_cells * dirty_cells)

        # Randomly set dirty cells on the grid
        for i in range(self.dirty_cells):
            x, y = self.random_empty_cell()
            self.dirty[x][y] = 1

    def step(self):
        # Advance the model by one step, updating agents and model state
        self.schedule.step()
        self.steps += 1
```

```python
    def clean(self, x, y):
        # Mark a cell as clean and update clean cell count
        self.dirty[x][y] = 0
        self.clean_cells += 1

    def is_clean(self, x, y):
        # Check if a cell is clean
        return self.dirty[x][y] == 0

    def random_empty_cell(self):
        # Get a random empty cell
        x, y = self.random_pos()

        # Keep selecting a random position until an empty cell is found
        while not self.grid.is_cell_empty((x, y)) or self.dirty[x][y] == 1:
            x, y = self.random_pos()

        return x, y

    def random_pos(self):
        # Generate a random position within the grid boundaries
        return random.randrange(self.grid.width), random.randrange(self.grid.height)

    def get_steps(self):
        # Get the current number of steps taken
        return self.steps

    def get_dirty_cells(self):
        # Get the total number of dirty cells
        return self.dirty_cells

    def get_clean_cells(self):
        # Get the total number of clean cells
        return self.clean_cells

    def get_dirty_percentage(self):
        # Calculate and return the percentage of dirty cells in the grid
        return round(100 - (self.clean_cells / self.dirty_cells) * 100, 2)
```

Listo, ya hemos construido nuestro agente y el modelo en el que estarán basados nuestros agentes para poder trabajar.

Ahora vamos a construir nuestra simulación. Vamos a asignarle los parámetros a nuestros agentes de modo que podamos darles diferentes escenarios para trabajar, y obtendremos nuestros resultados.

```
# Set the dimensions of the environment
WIDTH = 100
HEIGHT = 100
# Set the number of cleaning agents
NUM_AGENTS = 2
# Set the percentage of dirty cells in the environment
DIRT_PERCENTAGE = 0.9
# Set the maximum number of steps for the cleaning process
MAX_STEPS = 10000

# Record the start time to measure the execution time of the cleaning process
start_time = time.time()

# Call the clean function with the specified parameters
model = CleaningModel(WIDTH, HEIGHT, NUM_AGENTS, DIRT_PERCENTAGE)

# Print the initial percentage of dirty cells
print("Initial dirty percentage: ", model.get_dirty_percentage())

# Run the cleaning process until the environment is completely clean or the maximum number of steps is reached
for i in range(MAX_STEPS):
    if model.get_dirty_percentage() == 0:
        break

    model.step()

# Calculate the elapsed time for the cleaning process
end_time = time.time() - start_time

# Print the results, including the time taken and the final result of the cleaning process
print("Final dirty percentage: ", model.get_dirty_percentage())
print("Time taken: ", end_time)
print("Steps taken: ", model.get_steps())
```

Aquí hemos definido un escenario de 100x100, en donde tendremos 2 agentes, en este caso dos robots de limpieza, en un escenario donde el 90% de las celdas están sucias. También hemos asignado como límite de pasos que los agentes podrán realizar de 10000. Vamos a observar nuestros resultados:

```
Initial dirty percentage:  100.0
Final dirty percentage:  53.51
Time taken:  0.21248459815979004
Steps taken:  10000
```

En esta simulación podemos ver que de todas las celdas sucias que había, con dos agentes y 10000 pasos permitidos, tenemos un resultado final de 53.51% de todas las celdas sucias que había.

En este caso podemos ver que, con estos parámetros, nuestros agentes no han sido capaces de limpiar todo el escenario. Entonces, vamos a considerar, con uno y dos agentes, cuantos pasos nos tomara para que estos puedan completar la limpieza. Pues si corremos el código añadiéndole más pasos, más le permitiremos a los agentes a terminar la limpieza. Los resultados obtenidos para uno y dos agentes son los siguientes:

- Un agente: 274080 pasos
- Dos agentes: 154180 pasos.

Del mismo modo, podremos preguntarnos como se desempeñarían uno o dos agentes con 100 pasos permitidos, o con 1000, o con 10000. Los resultados para estas simulaciones son las siguientes:

- 100: 99.44% (1 agente), 98.82% (2 agentes)
- 1000: 96.02% (1 agente), 92.34% (2 agentes)
- 10000: 70.6% (1 agente), 53.51% (2 agentes)

Por último, podremos entonces preguntarnos, ¿cuál sería la cantidad optima de robots necesarios para completar la limpieza en el menor tiempo posible? Y que si bien, es lógico pensar que con 10 robots tendremos la menor cantidad de pasos posibles, no necesariamente es el que tiene el menor tiempo posible. Entonces si corremos la simulación con parámetros de 1 a 10 agentes, podremos obtener los tiempos que les toma para terminar la limpieza.

Si corremos la simulación para cada parámetro de número de agentes del 1 al 10, obtenemos que más o menos el mejor escenario es utilizar 7 agentes, tardando aproximadamente 1.21 segundos en completar la simulación.

Para cerrar, podemos concluir que mediante esta actividad pudimos poner en práctica la modelación de agentes para diferentes escenarios, en este caso para limpieza. Pudimos observar las capacidades que tenían los agentes para tomar sus propias decisiones y los comportamientos que desempeñaron en las simulaciones. Y también pudimos dar una pequeña aproximación a lo que sería si tuviéramos robots que limpiaran por ejemplo nuestros hogares, de modo que les permita a todos los robots, poder tomar decisiones sobre que habitaciones limpiar, si ya están limpias o si saber si ya hay otro agente en esa habitación.

Link al repositorio:

https://github.com/Riosmena/TC2008B/blob/main/Scripts/CleaningModel.ipynb