M1-RobotAspirador

November 9, 2022

1 M1. Simulación de Robot Aspirador

Alfonso Pineda Cedillo | A01660394

El objetivo de la actividad es realizar una simulación de un sistema multiagente basado en un robot aspirador, para este primer ejercicio, el movimiento de cada uno de los agentes es completamente aleatorio. Para dar solución a la presente actividad, es neceserario crear un ambiente donde se encontrará el robot, dicho ambiente estará conformado por celdas, las cuales pueden tener un valor booleano de limpio o sucuio. Finalmente, se busca hacer un análisis de las estadísitcas generales con la información generada por los agentes.

Una vez considerado lo anterior, es necesario establecer una serie de parámetros iniciales:

- La habitación (ambiente) tiene un tamaño de M * N celdas.
- Contará con un número n de agentes (robots).
- Porcentaje de celdas inicialmente sucias.
- Tiempo máximo de ejecución.

Asimismo, contamos con consideraciones iniciales para la simulación:

- De acuerdo con el porcentaje previamente mencionado, se inicializarán las celdas sucias de manera aleatoria.
- Los agentes (robots) se inicializarán en la celda [1,1].
- Para cada paso de tiempo, los agentes deverán considerar lo siguiente:
  - Si la celda está sucia, entonces aspira.
  - Si la celda está limpia, el agente elije una dirección aleatoria para moverse (unas de las 8 celdas vecinas) y elije la acción de movimiento (si no puede moverse allí, permanecerá en la misma celda).
- Se ejecuta el tiempo máximo establecido.

1.1 Ambiente

De acuerdo con la información que tenemos disponible del problema, podemos inferir que contamos con un ambiente parcialmente observable. A pesar de que nosotros como usuarios tenemos visibilidad de todo el tablero, los agentes no cuentan con dicha capacidad, cada robot únicamente puede ver el estado de la celda en la que se encuentra ubicado; en otras palabras, los agentes no cuentan con información del resto de las celdas ni de su comportamiento a lo largo de la simulación.

Asimismo, podemos considerar que para este caso particular, contamos con un ambiente estocástico, lo que significa que tiene un comportamiento no determinista; es decir, debido a que el compor-

tamiento de los agentes es completamente al azar, se convierte en un proceso cuyo resultado no es predecible. Es por eso que a lo largo de la actividad, realizaremos diversas iteraciones de la simulación para poder obtener estadísticas generales de la misma.

Por otra parte, el ambiente se clasificaría de igual forma como episódico, debido a que existe una división de episodios independientes entre sí; el primero consiste en la percepción del agente (el análisis del estado de la celda en la que se encuentra), el segundo en la acción que se llevará a cabo (aspirar o moverse).

Por último, nuestro ambiente también se clasificaría como dinámico, esto debido a que el estado de las celdas puede cambiar a lo largo de la simulación, es decir, las celdas pueden pasar de estar sucias a limpias; este proceso de cambio puede llevarse a cabo incluso al momento en que un robot esté deliberando su próximo movimiento. Debido a lo anterior, es necesario hacer énfasis en que el sistema es multiagente, de manera simultánea existen diversos agentes que interactúan con el ambiente.

1.2 Agentes

Los agentes son los encargados de realizar las acciones dentro del ambiente, en este caso, los agentes son los robots aspiradores. Cada uno de estos robots tiene la capacidad de moverse y aspirar, además de que cada uno de ellos tiene la capacidad de percibir el estado de la celda en la que se encuentra ubicado. Debido a lo anterior, podemos considerar a los agentes como reactivos simples, puesto que sus decisiones se basan únicamente en la información que tienen disponible en el momento (precepto actual) y no en un historial de preceptos; como lo hemos visto anteriormente, el comportamiento de los agentes se basa en verificar si la celda en la que se encuentra esta limpia o sucia, si está limpia, entonces elige una dirección aleatoria para moverse, si está sucia, entonces aspira.

De acuerdo con la información previamente mencionada, podemos establecer que el precepto inicial de cada agente es su posición inicial; así que para cada paso posterior, el precepto únicamente será el estado de la celda en la que se encuentra ubicado, debido a que es el único criterio que necesita para tomar una decisión.

Por cuestiones de aplicación (para poder agregar una cantidad diferente de suciedad dependiendo de los parámetros iniciales del sistema), establecimos a la suciedad (trash) como un agente más, el cual tiene como precepto inicial estar sucio.

1.3 Simulación

Primero que nada, es necesario importar las librerías que son necesarias para la realización y análisis de la simulación. Para este ejercicio, haremos uso de la librería mesa como librería principal para la simulación de nuestro modelo basado en agentes, así como de librerías auxiliares.

[ ]:
```
import mesa
import random

# Contador de movimientos
movesCounter = 0
```

Definimos al agente que representa la suciedad (trash), el cual, al inicializarlo tiene como precepto su condición inicial previamente descrita, la cual es que debe estar sucio (Suciedad -> 1).

[ ]:
```python
class Trash(mesa.Agent):
    def __init__(self, unique_id, model):
        super().__init__(unique_id, model)
        self.state = 1

    def step(self):
        pass
```

Definimos al agente que representa a nuestro robot aspirador, dentro de la clase definimos las acciones que nuestro agente es capaz de realizar, las cuales son: aspirar, moverse y verificar si la celda en la que se encuentra es sucia o limpia. Tenemos la función step, la cual analiza el estado de la celda en la que se encuentra el agente, si está sucia, entonces aspira, si está limpia, entonces elige una dirección aleatoria para moverse; nuestra función move elige una dirección aleatoria para que el agente pueda moverse.

[ ]:
```python
class Robot(mesa.Agent):
    def __init__(self, unique_id, model):
        super().__init__(unique_id, model)
        self.state = "Cleaning"
        self.moves = 0

    def move(self):
        # Definimos la lista de posiciones posibles
        nextStep = self.model.grid.get_neighborhood(self.pos, moore=True,␣
↪include_center=False)

        # De forma aleatroia seleccionamos una posición de la lista
        new_position = self.random.choice(nextStep)
        self.model.grid.move_agent(self, new_position)
        self.moves += 1

    def step(self):
        self.move()
        cellmates = self.model.grid.get_cell_list_contents([self.pos])

        # En cada paso, si la celda está sucia, se limpia; si la cella está␣
↪limpia, se mueve.
        if any(isinstance(agent, Trash) for agent in cellmates):
            trash = next(
                agent for agent in cellmates if isinstance(agent, Trash))

            self.state = "Cleaning"
            self.model.grid.remove_agent(trash)

        else:
            self.state = "Moving"
```

Declaramos la clase para la simulación de nuestro modelo (ambiente), dentro de la función step

verificamos si se ha alcanzado el límite de pasos o no, para saber en qué momento detener la simulación. De igual forma definimos funciones auxiliares que nos ayudan a recopilar la información necesaria para realizar las gráficas (análisis estadístico).

[ ]:
```
class Simulation(mesa.Model):
    def __init__(self, M, N, n_agents, dirtyCells, max_steps=100):
        self.num_agents = n_agents
        self.grid = mesa.space.MultiGrid(M, N, True)
        self.schedule = mesa.time.RandomActivation(self)
        self.max_steps = max_steps
        self.running = True

        room = M * N

        dirtyCells = int(room * dirtyCells / 100)


        # Creamos los robots aspiradores (agentes)
        for i in range(self.num_agents):
            r = Robot(i, self)
            self.schedule.add(r)

            self.grid.place_agent(r, (1, 1)) # El agente se inicializa en la posición (1, 1)

        counter = 0

        # Creamos al agente que representa la suciedad
        while counter < dirtyCells:
            a = random.randrange(self.grid.width)
            b = random.randrange(self.grid.height)

            if self.grid.is_cell_empty((a, b)):
                trash = Trash(counter, self)
                self.grid.place_agent(trash, (a, b))
                counter += 1

        # Data Collector para obtener la cantidad de celdas sucias con el paso del tiempo
        self.TRData = mesa.DataCollector(
            {
                "Trash remaining": self.dirtyCells,
            }
        )

        self.TRData.collect(self)
```

```
        # Data Collector para obtener el porcentaje de celdas limpias con el paso del tiempo
        self.PCData = mesa.DataCollector(
            {
                "Percentage of clean cells": self.cleanCellsP,
            }
        )

        self.PCData.collect(self)

        # Data Collector para obtener el total de movimientos con el paso del tiempo
        self.TMData = mesa.DataCollector(
            {
                "Total Moves": self.totalMoves,
            }
        )
        self.TMData.collect(self)



    def step(self):
        # Si se alcanza el límite de pasos, se detiene la simulación
        if self.schedule.steps >= self.max_steps:
            self.running = False

        # Si aún no se alcanza el límite de pasos, se ejecuta el paso de la simulación a menos que no haya celdas sucias
        elif self.schedule.steps < self.max_steps:
            if self.dirtyCells() == 0:
                self.running = False

            else:
                self.schedule.step()

        self.TRData.collect(self)
        self.PCData.collect(self)
        self.TMData.collect(self)

    def run_model(self):
        # Mientras la simulación se esté ejecutando
        while self.running:
            # Ejecuta el paso de la simulación
            self.step()

    # Fucnión que nos permite obtener los movimeintos totales de los agentes
    def totalMoves(self):
```

```
        movesCounter = 0
        for agent in self.schedule.agents:
            movesCounter += agent.moves
        return movesCounter

    # Función que nos permite obtener el número de celdas limpias
    def cleanCells(self):
        cleanCellsCounter = 0
        for cell in self.grid.coord_iter():
            cell_content, x, y = cell
            if not any(isinstance(agent, Trash) for agent in cell_content):
                cleanCellsCounter += 1
        return cleanCellsCounter

    # Función que nos permite obtener el porcentaje de celdas limpias
    def cleanCellsP(self):
        return self.cleanCells() / (self.grid.width * self.grid.height) * 100

    # Función que nos permite obtener el número de celdas sucias
    def dirtyCells(self):
        dirtyCellsCounter = 0
        for cell in self.grid.coord_iter():
            cell_content, x, y = cell
            if any(isinstance(agent, Trash) for agent in cell_content):
                dirtyCellsCounter += 1
        return dirtyCellsCounter

    # Función que nos permite obtener el porcentaje de celdas sucias
    def dirtyCellsP(self):
        # Calcula el porcentaje de celdas Dirty
        return self.dirtyCells() / (self.grid.width * self.grid.height) * 100
```

Declaramos la función que nos permite visualizar la simulación de nuestro modelo (el tablero y las gráficas), en esta sección definimos nuestras condiciones iniciales.

[ ]:
```
def agent_portrayal(agent):
    portrayal = {"Shape": "circle",
                 "Filled": "true",
                 "Layer": 0,
                 "Color": "red",
                 "r": 0.5}

    if agent.state == 1:
        portrayal["Color"] = "black"
        portrayal["r"] = 0.2
    elif agent.state == "Cleaning":
        portrayal["Color"] = "blue"
```

```
    else:
        portrayal["Color"] = "blue"
    return portrayal

grid = mesa.visualization.CanvasGrid(agent_portrayal, 12, 12, 500, 500)

TRchart = mesa.visualization.ChartModule(
    [{"Label": "Trash remaining", "Color": "Blue"}],␣
 ↪data_collector_name="TRData"
)

PCchart = mesa.visualization.ChartModule(
    [{"Label": "Percentage of clean cells", "Color": "Green"}],␣
 ↪data_collector_name="PCData"
)

TMchart = mesa.visualization.ChartModule(
    [{"Label": "Total Moves", "Color": "Gray"}], data_collector_name="TMData"
)

server = mesa.visualization.ModularServer(
    Simulation, [grid,TRchart, PCchart, TMchart], "Envyroment", {"M": 12, "N":␣
 ↪12, "n_agents": 5, "dirtyCells": 30 }
    )

server.port = 8005
server.launch()
```

Para analizar los datos obtenidos, primero correremos la simulación con los siguientes parámetros establecidos, un espacio de 12 x 12 celdas, 30% de suciedad, 100 pasos de tiempo y 5 robots. Posteriormente, aumentaremos la cantidad de agentes (robots aspiradores) a 10 y observaremos los resultados obtenidos.

Gráfica de Celdas sucias restantes vs tiempo (pasos)

Nuestra primer gráfica nos muestra la cantidad de celdas sucias en cada paso de tiempo, podemos observar que a medida que los agentes van limpiando las celdas, la cantidad de celdas sucias va disminuyendo. En el primer caso (con 5 agentes) podemos observar que la cantidad de celdas sucias disminuye a un ritmo relativamente lento, y en muy pocos casos se termina de limpiar la habitación por completo, en la mayoría de los casos, las celdas sucias restantes se mantienen entre 3 y 6. Sin embargo, en el segundo caso (con 10 agentes) podemos observar que la cantidad de celdas sucias disminuye de manera más rápida, esto debido a que la cantidad de agentes es mayor, por lo que la cantidad de celdas sucias que pueden limpiar es mayor; debido a lo anterior, en todas las iteraciones realizadas, la simulación se termina debido a que se logra limpiar la habitación por completo y no por llegar al límite de pasos.

Gráfica de porcentaje de Celdas limpias vs tiempo (pasos)

En nuestra segunda gráfica, podemos analizar el porcentaje de celdas limpias en cada paso de

tiempo, podemos observar que a medida que los agentes van limpiando las celdas, el porcentaje de celdas limpias va aumentando. En el primer caso (con 5 agentes) podemos observar que el porcentaje de celdas limpias aumenta a un ritmo relativamente lento, y en muy pocos casos se logra llegar a un 100%, en la mayoría de los casos, el porcentaje de celdas limpias se mantiene entre 92% y 98%. Sin embargo, debido a las mismas razones expuestas en la gráfica anterior, en el segundo caso (con 10 agentes) podemos observar que el porcentaje de celdas limpias aumenta de manera más rápida; debido a lo anterior, en todas las iteraciones realizadas, el porcentaje de celdas limpias se mantiene en 100, es decir, se logra limpiar la habitación por completo antes de llegar al límite de pasos.

Total de movies vs tiempo (pasos)

Esta tercer gráfica nos uestra la cantidad de movimientos realizados por todos los agentes conforme pasa el tiempo; a diferencia de las gráficas anteriores, en esta última podemos observar un movimiento idéntico para ambos casos, debido a que cada agente se mueve al mismo tiempo, el comportamiento de la gráfica siempre es lineal. La única difernecia entre ambos casos es la cantidad de moviemientos realizados, siendo considerablemente menor cuando el número de agentes es 5, a pesar de que individualmente cada agente realiza menos movimientos (debido a que la habitación se limpia más rápido), la cantidad de movimientos totales es mayor debido a que hay más agentes.

Enlace al Repositorio de Github: https://github.com/AlfonsoPineda/SistemasMultiagentes.git