M1. Actividad

Alumno:

Jorge Emiliano Pomar Mendoza | A01709338

Modelación de sistemas multiagentes con gráficas computacionales (Gpo 301)

Profesor:

Tecnológico de Monterrey, campus Querétaro

Febrero- Junio 2024

20 de mayo del 2024

índice

1. Introducción **2**
2. Agente **2**
3. Modelo de Oficina **3**
4. Simulaciones y Resultados **5**
4.1 Simulación con 1 agente 5
4.2 Simulación con 5 agentes: 6
4.3 Simulación con 10 agentes: 7

1. Introducción

Se requería para esta actividad de una simulación de un espacio de oficina en el que hay mucha basura (80% inicialmente) colocada de manera aleatoria. De igual manera se colocan de manera aleatoria un número determinado de aspiradoras inteligentes que recorren la oficina en un límite de tiempo definido (en este caso 100). Las aspiradoras recorren el espacio y toman la decisión en cada celda de si seguir avanzando o aspirar. Ya que si se posicionan en una celda marcada como sucia, aspiran para quitar la suciedad y posteriormente siguen a la siguiente celda.

Enlace al repositorio
https://github.com/pomaremiliano/TC2008B/tree/main/MultiAgentes/Actividad1

2. Agente

Para esto se realizó un programa en python utilizando agentes de la librería de mesa. Primero se definieron las aspiradoras, que en este caso son agentes que están instanciados de manera aleatoria en el tablero y cuando se instancian tienen conocimiento del vecindario de celdas que hay a su alrededor. Sin embargo, no conocen cuales son las celdas sucias, porque igualmente se instancian las celdas sucias de manera aleatoria en el tablero.

```
# Definir el agente aspiradora que se moverá por el grid
class Aspiradoras(Agent):
    def __init__(self, unique_id, model):
        super().__init__(unique_id, model)

    # Metodo que cuenta la cantidad de celdas sucias en el vecindario
y posiciona la aspiradora en una celda sucia
    def step(self):
        x, y = self.pos
```

```
        if self.model.grid.is_cell_empty((x, y)):
            self.model.grid.place_agent(None, (x, y), 0)
        else:
            pasos_posibles =
self.model.grid.get_neighborhood(self.pos, moore=True,
include_center=False)
            # Seleccionar una celda random en donde se encuentre
suciedad para mover la aspiradora
            nueva_posicion = random.choice(pasos_posibles)
            self.model.grid.move_agent(self, nueva_posicion)
```

3. Modelo de Oficina

Después, para poder generar el tablero, se hizo un modelo de oficina en el que se define el espacio, se colocan los agentes, se colocan las celdas sucias y además se recopila la información de cada paso que da la aspiradora.

```
# Definir el modelo de la oficina que tiene las aspiradoras
class OficinaModel(Model):
    # Inicializar el modelo con el tamaño del grid, el número de
aspiradoras, el porcentaje de celdas sucias, el tiempo máximo y los
pasos en los que se calculará el porcentaje de limpieza
    def __init__(self, X, Y, num_agents, porc_celdas_sucias,
max_tiempo, paso1, paso2, paso3):
        super().__init__()
        self.num_agents = num_agents
        self.grid = MultiGrid(X, Y, False)
        self.schedule = RandomActivation(self)
        self.running = True
        self.max_tiempo = max_tiempo
        self.tiempo = 0
```

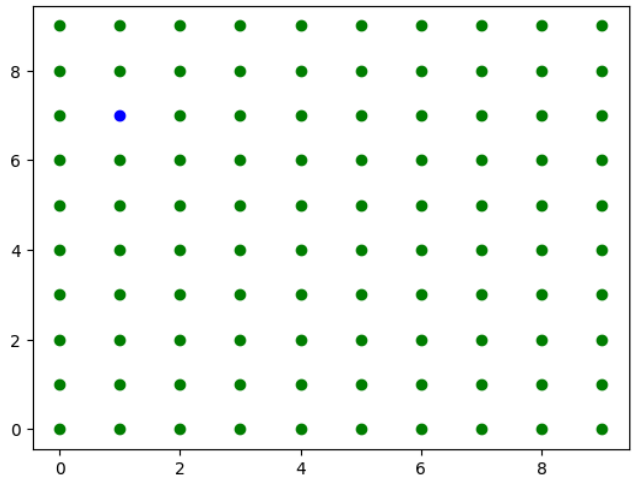
```
        self.paso1 = paso1
        self.paso2 = paso2
        self.paso3 = paso3
        self.porcentajes_limpio = []
        self.datacollector = DataCollector(
            model_reporters={"Particles":get_particles}
        )

        # Aqui se itera sobre el grid y se usa random para colocar
agentes
        for i in range(num_agents):
            a = Aspiradoras(i, self)
            x = random.randrange(self.grid.width)
            y = random.randrange(self.grid.height)
            self.schedule.add(a)
            self.grid.place_agent(a, (x, y))

    # Aqui se define un metodo de pasos, y cada paso se recorre el
schedule y se llama al metodo step de cada agente, ademas se
incrementa el tiempo y se recolectan los datos de porcentaje de
limpio en cada paso
    def step(self):
        self.schedule.step()
        self.tiempo += 1
        self.datacollector.collect(self)
        if self.tiempo in self.paso1:

self.porcentajes_limpio.append(self.calcula_porcentaje_limpio())
        if self.tiempo in self.paso2:

self.porcentajes_limpio.append(self.calcula_porcentaje_limpio())
        if self.tiempo in self.paso3:

self.porcentajes_limpio.append(self.calcula_porcentaje_limpio())
        if self.tiempo >= self.max_tiempo:
            self.running = False

    # Metodo que calcula el porcentaje de celdas limpias en el grid y
```

```
regresa el porcentaje de celdas limpias en el paso
    def calcula_porcentaje_limpio(self):
        limpio = 0
        for y in range(self.grid.height):
            for x in range(self.grid.width):
                if self.grid.is_cell_empty((x, y)):
                    limpio += 1
        return ((limpio/(self.grid.width*self.grid.height)) * 100)
```

Por último, se hizo la simulación del espacio y las aspiradoras. Se consideraron tres escenarios principales para esto. El primer escenario en el que el número de agentes es igual a 1, 5 y 10. Dentro de este escenario, se consideró la variación de los porcentajes de celdas limpias después de x pasos. En este caso, el porcentaje de celdas limpias después de los pasos es de 500, 1000, y 1500.
A continuación, los resultados de las simulaciones.

4. Simulaciones y Resultados

4.1 Simulación con 1 agente



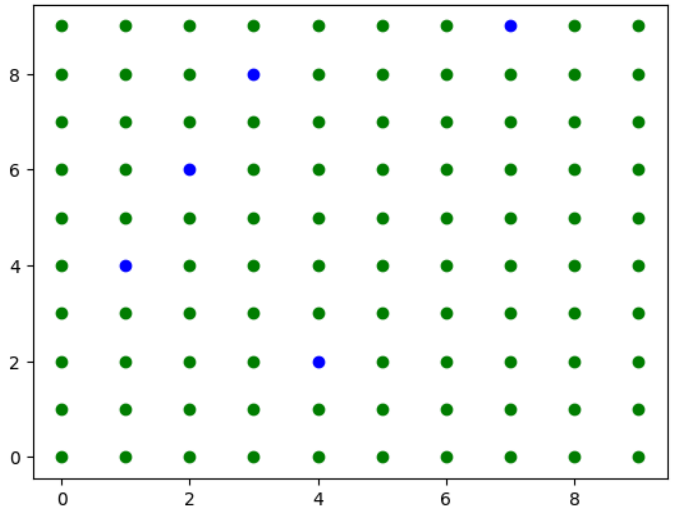
99.0% celdas limpias en paso [500] con 1 aspiradoras en 100 tiempo respectivamente.

99.0% celdas limpias en paso [1000] con 1 aspiradoras en 100 tiempo respectivamente.

99.0% celdas limpias en paso [1500] con 1 aspiradoras en 100 tiempo respectivamente.

Analizando los resultados, me sorprendió bastante que al ser un solo agente, pudiera terminar en el tiempo máximo. Sin embargo, tras un poco más de observación, se podría decir que el hecho de que un solo agente esté aspirando en el tablero, es probable que se reduzcan las posibles colisiones con otros agentes que causan vueltas innecesarias y retrasos. En este caso, podemos ver en el gráfico que la aspiradora (representada por el punto azul) terminó después del tiempo máximo de limpiar todas las celdas sucias (en este caso se representan de color rojo y las celdas verdes son limpias cuando la aspiradora ya limpió).

4.2 Simulación con 5 agentes:



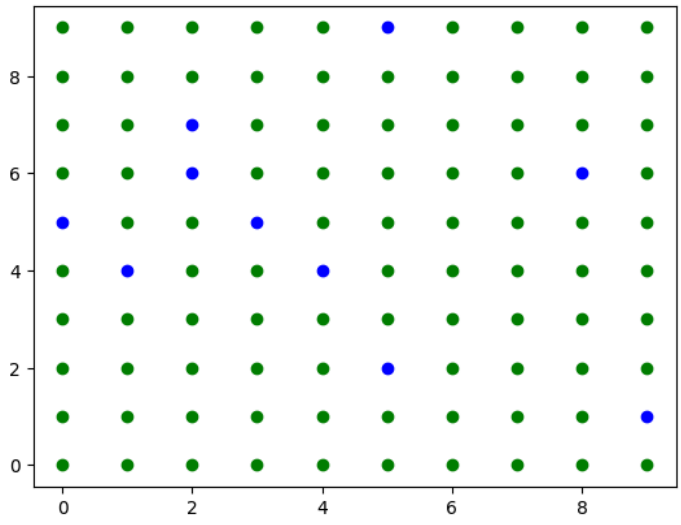
95.0% celdas limpias en paso [500] con 5 aspiradoras en 100 tiempo respectivamente.

95.0% celdas limpias en paso [1000] con 5 aspiradoras en 100 tiempo respectivamente.

95.0% celdas limpias en paso [1500] con 5 aspiradoras en 100 tiempo respectivamente.

En este caso, vemos en el gráfico que las 5 aspiradoras representadas con los puntos azules terminaron en lugares dispersos y lograron limpiar las celdas sucias en el tiempo máximo. Por otra parte, para las tres recolecciones de datos en los pasos, vemos que considerablemente bajó el porcentaje de lo que se llegó a limpiar en cada uno de los pasos en comparación con la simulación anterior. Igualmente, partiendo desde la premisa anterior, puede ser que el hecho de que son más agentes recorriendo el tablero, puede ser que los agentes entre sí se pongan en el camino de los otros y esto ocasiona que existan retrasos y termine siendo un poco menos eficiente.

4.3 Simulación con 10 agentes:



90.0% celdas limpias en paso [500] con 10 aspiradoras en 100 tiempo respectivamente.

90.0% celdas limpias en paso [1000] con 10 aspiradoras en 100 tiempo respectivamente.

90.0% celdas limpias en paso [1500] con 10 aspiradoras en 100 tiempo respectivamente.

De nuevo volvemos a ver que dentro del tiempo máximo, los agentes lograron limpiar toda la oficina. Y partiendo desde la teoría anteriormente mencionada, parece ser que el hecho de aumentar los agentes, ocasionó que la eficiencia disminuyera 5% con respecto al anterior.