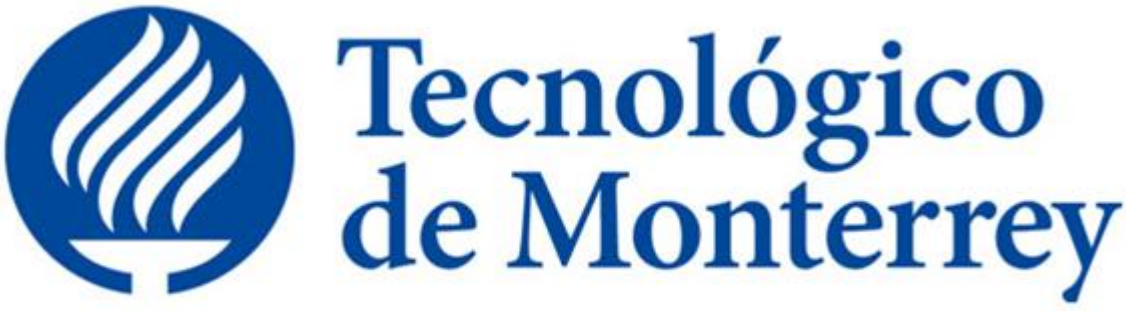

Actividad 1 - Reporte

Módulo 1

Arturo Díaz López A01709522

TC2008B.301

12/11/2023

Instrucciones

Para este problema, deberás entregar, **de manera individual**, un informe en PDF que estudie las estadísticas de un robot de limpieza reactivo, así como el enlace al repositorio en Github del código desarrollado para esta actividad. El código debe ajustarse al estilo solicita en el siguiente documento.

Dado:

- Habitación de MxN espacios.
- Número de agentes.
- Porcentaje de celdas inicialmente sucias.
- Tiempo máximo de ejecución.

Realiza la siguiente simulación:

- Inicializa las celdas sucias (ubicaciones aleatorias).
- Todos los agentes empiezan en la celda [1,1].
- En cada paso de tiempo:
    - 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.

Para un espacio de 100x100, considera los siguientes escenarios:

- Escenario 1: 1 agente, 90% de celdas sucias.
- Escenario 2. 2 agentes, 90% de celdas sucias.

Deberás resolver las siguientes preguntas:

- ¿Cuántos pasos de simulación toma limpiar todo el espacio?
- ¿Qué porcentaje de celdas sucias queda con los siguientes pasos de simulación: 100, 1000, 10000?

A continuación, determina cuál es la cantidad óptima de aspiradoras que debe de tener para realizar la limpieza en el menor tiempo posible. Considera que tenemos un máximo de 10 aspiradoras disponibles.

Solución

La simulación tiene como objetivo estudiar el comportamiento de agentes de limpieza reactiva en entornos sucios. Se consideran dos escenarios principales: uno con un solo agente y otro con dos agentes, ambos en un espacio de 100x100 celdas con un 90% de suciedad inicial. Se implementó un modelo de agentes de limpieza reactivo utilizando el framework Mesa. Cada agente se mueve en el entorno, aspirando si la celda está sucia y eligiendo una dirección aleatoria si está limpia. El tiempo de ejecución está limitado a un máximo de 10,000 pasos de simulación.

Para comenzar con la solución, se definió un agente de limpieza llamado CleaningAgent

```python
class CleaningAgent(Agent):
    def move(self):
        empty_cell = self.model.grid.get_neighborhood(self.pos, moore=True, include_center=False)
        empty_cell = [i for i in empty_cell if self.model.grid.is_cell_empty(i)]

        if empty_cell:
            new_position = random.choice(empty_cell)
            self.model.grid.move_agent(self, new_position)

    def step(self):
        x, y = self.pos

        if self.model.is_clean(x, y):
            self.move()

        else:
            self.model.clean(x, y)
            self.move()
```

Dentro de su definición, contamos con los siguientes métodos:

- move(self): Indica cómo se mueve el agente en su entorno. Obtiene las celdas vecinas que están vacías (empty_cells), y realiza la siguiente validación: si existe una celda vacía, escoge aleatoriamente una nueva celda y se mueve hacia ella.
- step(self): Define el comportamiento del agente en cada paso de tiempo. Primero, se obtiene su posición, después, si la celda en la que se encuentra está limpia, se llama al método move(). Si la celda en la que se encuentra está sucia, el agente llama al método clean() y move() para limpiar la celda y luego moverse.

Continuando con la solución presentada, contamos con la definición del modelo CleaningModel, el cual contiene la lógica principal para la ejecución de la simulación de limpieza.

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

        self.dirty = np.zeros((width, height))
        self.dirty_cells = int(width * height * dirty_cells)
        self.clean_cells = 0

        self.current_id = 0
        self.steps = 0
```

```
self.current_id = 0
self.steps = 0

for _ in range(num_agents):
    agent = CleaningAgent(self.next_id(), self)
    self.schedule.add(agent)
    x, y = self.random_empty_cell()
    self.grid.place_agent(agent, (x, y))

for _ in range(self.dirty_cells):
    x, y = self.random_empty_cell()
    self.dirty[x][y] = 1
```

```
def step(self):
    self.schedule.step()
    self.steps += 1

def clean(self, x, y):
    self.dirty[x][y] = 0
    self.clean_cells += 1

def is_clean(self, x, y):
    return self.dirty[x][y] == 0

def random_empty_cell(self):
    x, y = self.random_pos()
    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):
    return random.randrange(self.grid.width), random.randrange(self.grid.height)

def get_dirty_percentage(self):
    return (self.dirty_cells - self.clean_cells) / self.dirty_cells * 100
```

```
def next_id(self):
    self.current_id += 1
    return self.current_id
```

- __init__(self, width, height, num_agents, dirty_cells): Inicializa una instancia del modelo. Se crean e inicializan los atributos del modelo, como la cuadrícula (grid) y el programa de activación aleatoria (schedule). Además, se generan y colocan los agentes CleaningAgent en posiciones aleatorias, y se inicializa la suciedad en el entorno.
- step(self): Este método ejecuta un paso de tiempo en la simulación llamando al método step() de la clase RandomActivation, lo que activa a cada agente en un orden aleatorio.
- clean(self, x, y): Marca una celda como limpia.
- is_clean(self, x, y): Verifica si una celda está limpia.

- random_empty_cell(self): Genera una posición aleatoria que esté vacía y no sucia.
- random_pos(self): Genera una posición aleatoria en la cuadrícula.
- get_dirty_percentage(self): Calcula y devuelve el porcentaje de celdas sucias en el entorno.
- next_id(self): Aumenta el contador de id en 1 unidad.

Para lograr modelar nuestros escenarios declaramos las siguientes constantes:

```
WIDTH = 100
HEIGHT = 100
DIRT_PERCENTAGE_1 = 0.9
MAX_STEPS = 10000
```

Y finalmente, modelamos el primer escenario:

```
NUM_AGENTS_1 = 1

start_time_1 = time.time()
model_1 = CleaningModel(WIDTH, HEIGHT, NUM_AGENTS_1, DIRT_PERCENTAGE_1)

initial_dirty_percentage_1 = model_1.get_dirty_percentage()

for i in range(MAX_STEPS):
    if model_1.get_dirty_percentage() == 0:
        break
    model_1.step()

final_dirty_percentage_1 = model_1.get_dirty_percentage()
end_time_1 = time.time() - start_time_1

print("E S C E N A R I O  1 ")
print("\nInitial dirty percentage: ", initial_dirty_percentage_1)
print("Final dirty percentage: ", final_dirty_percentage_1)
print("Time taken: ", end_time_1)
print("Steps taken: ", model_1.steps)
```

En este bloque de código, se realiza la simulación del escenario 1. Se crea una instancia del modelo CleaningModel con un solo agente (NUM_AGENTS_1 = 1). Luego, se ejecuta un bucle de simulación (for i in range(MAX_STEPS)) que avanza en el tiempo llamando al método step() del modelo. Se verifica si el porcentaje de suciedad ha alcanzado cero, en cuyo caso se rompe el bucle.

Para el segundo escenario, el número de agentes es igual a 2, por lo que tendremos lo siguiente:

```
NUM_AGENTS_2 = 2

start_time_2 = time.time()
model_2 = CleaningModel(WIDTH, HEIGHT, NUM_AGENTS_2, DIRT_PERCENTAGE_1)

initial_dirty_percentage_2 = model_2.get_dirty_percentage()

for i in range(MAX_STEPS):
    if model_2.get_dirty_percentage() == 0:
        break
    model_2.step()

final_dirty_percentage_2 = model_2.get_dirty_percentage()
end_time_2 = time.time() - start_time_2

print("E S C E N A R I O  2 ")
print("\nInitial dirty percentage: ", initial_dirty_percentage_2)
print("Final dirty percentage: ", final_dirty_percentage_2)
print("Time taken: ", end_time_2)
print("Steps taken: ", model_2.steps)
```

Los resultados de ambas simulaciones serán los siguientes:

```
E S C E N A R I O  1

Initial dirty percentage:  100.0
Final dirty percentage:  67.58888888888889
Time taken:  0.10799837112426758
Steps taken:  10000
```

```
E S C E N A R I O  2

Initial dirty percentage:  100.0
Final dirty percentage:  56.97777777777778
Time taken:  0.17123818397521973
Steps taken:  10000
```

Los resultados incluyen estadísticas clave como el porcentaje de suciedad inicial y final, el tiempo de ejecución y la cantidad de pasos tomados en cada escenario. Estos resultados proporcionan información sobre la eficiencia del modelo de limpieza en diferentes condiciones. La simulación permite observar cómo la presencia de más agentes afecta la velocidad de limpieza del entorno y cómo factores como el porcentaje de suciedad inicial influyen en el rendimiento global del sistema de limpieza.

Preguntas

¿Cuántos pasos de simulación toma limpiar todo el espacio?

- Escenario 1: 273441
- Escenario 2: 175982

¿Qué porcentaje de celdas sucias queda con los siguientes pasos de simulación: 100, 1000, 10000?

- Escenario 1 (100 pasos): 99.37777777777778
- Escenario 1 (1000 pasos): 95.83333333333334
- Escenario 1 (10000 pasos): 69.16666666666667

- Escenario 2 (100 pasos): 98.92222222222222
- Escenario 2 (1000 pasos): 90.63333333333333
- Escenario 2 (10000 pasos): 50.3

Conclusiones

Esta actividad me ayudó a comprender mejor cómo modelar sistemas multiagentes, fue muy útil e interesante jugar con los valores para comprender de qué manera esto afectaba los resultados finales de la simulación.

Creo que la solución presentada podría optimizarse bastante y aunque no fue el propósito de la actividad, sí me quedo intrigado en pensar de qué manera puedo integrar algoritmos avanzados a la lógica de limpieza de los agentes para obtener un mucho mejor resultado en las simulaciones.