Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Querétaro

TC2008B.301

Modelación de sistemas multiagentes con gráficas computacionales

Actividad 1: Agentes de limpieza

Profesores

- Denisse Maldonado
- Alejandro Fernandez
- Pedro O. Perez

Presenta

- Daniel Felipe Hurtado Giraldo
- A01707774

Querétaro, Querétaro Miércoles 8, Noviembre 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.

Desarrollar un informe con lo observado.

In [ ]:
```
from mesa import Agent, Model
from mesa.space import MultiGrid
from mesa.time import SimultaneousActivation
from mesa.datacollection import DataCollector

# Importamos los siguientes paquetes para el mejor manejo de valores numéric
import numpy as np
import pandas as pd

# matplotlib lo usaremos crear una animación de cada uno de los pasos del mo
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.animation as animation
plt.rcParams["animation.html"] = "jshtml"
matplotlib.rcParams['animation.embed_limit'] = 2**128

# Definimos otros paquetes que vamos a usar para medir el tiempo de ejecució
import time
import datetime
```

Agente

In [ ]:
```
class CleaningRobot(Agent):
    def __init__(self, unique_id, model):
        super().__init__(unique_id, model)
```

```
    def step(self):
        # Si la celda está sucia, la limpia y luego intenta moverse
        if self.model.dirty(self.pos):
            self.model.clean(self.pos)
        self.move()

    def move(self):
        # Obtiene las celdas vecinas
        possible_moves = self.model.grid.get_neighborhood(self.pos, moore=Tr

        # Filtra las celdas que están vacías
        empty_moves = [move for move in possible_moves if self.model.grid.is

        # Si hay celdas vacías disponibles, elige una al azar y se mueve
        if empty_moves:
            next_move = self.random.choice(empty_moves)
            self.model.grid.move_agent(self, next_move)
        # Si no hay celdas vacías, el agente se queda en su posición actual
        else:
            pass  # El agente no se mueve
```

In [ ]:
```
DIRTY = 1  # Representa una celda sucia
CLEAN = 0  # Representa una celda limpia
```

Modelo

In [ ]:
```
class CleaningModel(Model):
    def __init__(self, width, height, initial_dirt, n_agents, max_steps):
        self.grid = MultiGrid(width, height, torus=False)
        self.schedule = SimultaneousActivation(self)
        self.cleaned = 0
        self.num_agents = n_agents
        self.max_steps = max_steps
        self.dirty_cells = np.zeros((width, height))

        # Iniciamos celdas sucias
        dirt_count = 0
        while dirt_count < initial_dirt:
            x = self.random.randrange(self.grid.width)
            y = self.random.randrange(self.grid.height)
            if self.dirty_cells[x, y] != DIRTY:
                self.dirty_cells[x, y] = DIRTY
                dirt_count += 1

        # Iniciamos agentes
        for i in range(n_agents):
            robot = CleaningRobot(i, self)
            self.grid.place_agent(robot, (1,1))
            self.schedule.add(robot)

        self.datacollector = DataCollector(
            model_reporters={"Cleaned": lambda m: m.cleaned}
        )
```

```
    def step(self):
        self.datacollector.collect(self)
        self.schedule.step()
        if self.schedule.steps > self.max_steps:
            self.running = False

    # Celda esta sucia
    def dirty(self, pos):
        x, y = pos
        return self.dirty_cells[x, y] == DIRTY

    # Celda esta limpia
    def clean(self, pos):
        x, y = pos
        if self.dirty(pos):
            self.dirty_cells[x, y] = CLEAN
            self.cleaned += 1

    def dirty_cells_remaining(self):
        return np.count_nonzero(self.dirty_cells == DIRTY)
```

Escenario 1

1 Agente. 90% de las celdad sucias

In [ ]:
```
# Parámetros para el escenario 1
WIDTH = 100
HEIGHT = 100
INITIAL_DIRT_1 = int(WIDTH * HEIGHT * 0.90)  # 90% de las celdas están sucia
N_AGENTS_1 = 1  # 1 agente de limpieza
MAX_STEPS_1 = 1000000  # Número máximo de pasos para la simulación
ITERATIONS = [100, 1000, 10000]

def process(model, iterations):
    total_cells = WIDTH * HEIGHT
    results = {}
    for steps in iterations:
        while model.schedule.steps < steps:
            model.step()

        dirty_cells_remaining = model.dirty_cells_remaining()
        dirty_percentage = (dirty_cells_remaining / total_cells) * 100
        results[steps] = dirty_percentage
        print(f"Iteración {steps}: {dirty_percentage:.2f}% celdas sucias res

    return results

def complete_cleaning(model):
    while model.dirty_cells_remaining() > 0 and model.schedule.steps < MAX_S
        model.step()

    return model.schedule.steps

# Ejecución del escenario 1
```

```
start_time = time.time()
model_1 = CleaningModel(WIDTH, HEIGHT, INITIAL_DIRT_1, N_AGENTS_1, MAX_STEPS
results = process(model_1, ITERATIONS)

# Determina el número de pasos para completar la limpieza
total_steps = complete_cleaning(model_1)

# Tiempo de ejecución del escenario 1
elapsed_time = datetime.timedelta(seconds=(time.time() - start_time))
print('Tiempo de ejecución del Escenario 1:', elapsed_time)

# Verificar si todas las celdas fueron limpiadas
total_cleaned = model_1.dirty_cells_remaining() == 0
print(f"¿Todas las celdas fueron limpiadas? {'Sí' if total_cleaned else 'No'
print(f"Número de pasos para completar la limpieza: {total_steps}")
```

```
Iteración 100: 89.50% celdas sucias restantes.
Iteración 1000: 85.72% celdas sucias restantes.
Iteración 10000: 61.68% celdas sucias restantes.
Tiempo de ejecución del Escenario 1: 0:00:01.211277
¿Todas las celdas fueron limpiadas? Sí
Número de pasos para completar la limpieza: 228510
```

Escenario 2

2 Agentes. 90% de las celdad sucias

In [ ]:

```
# Parámetros para el escenario 2
WIDTH = 100
HEIGHT = 100
INITIAL_DIRT_2 = int(WIDTH * HEIGHT * 0.90)  # 90% de las celdas están sucia
N_AGENTS_2 = 2  # 2 agente de limpieza
MAX_STEPS_2 = 1000000  # Número máximo de pasos para la simulación
ITERATIONS = [100, 1000, 10000]

def process(model, iterations):
    total_cells = WIDTH * HEIGHT
    results = {}
    for steps in iterations:
        while model.schedule.steps < steps:
            model.step()

        dirty_cells_remaining = model.dirty_cells_remaining()
        dirty_percentage = (dirty_cells_remaining / total_cells) * 100
        results[steps] = dirty_percentage
        print(f"Iteración {steps}: {dirty_percentage:.2f}% celdas sucias res

    return results

def complete_cleaning(model):
    while model.dirty_cells_remaining() > 0 and model.schedule.steps < MAX_S
        model.step()

    return model.schedule.steps
```

```
# Ejecución del escenario 2
start_time = time.time()
model_2 = CleaningModel(WIDTH, HEIGHT, INITIAL_DIRT_2, N_AGENTS_2, MAX_STEPS
results = process(model_2, ITERATIONS)

# Determina el número de pasos para completar la limpieza
total_steps = complete_cleaning(model_2)

# Tiempo de ejecución del escenario 2
elapsed_time = datetime.timedelta(seconds=(time.time() - start_time))
print('Tiempo de ejecución del Escenario 2:', elapsed_time)

# Verificar si todas las celdas fueron limpiadas
total_cleaned = model_2.dirty_cells_remaining() == 0
print(f"¿Todas las celdas fueron limpiadas? {'Sí' if total_cleaned else 'No'
print(f"Número de pasos para completar la limpieza: {total_steps}")
```

```
Iteración 100: 89.03% celdas sucias restantes.
Iteración 1000: 83.93% celdas sucias restantes.
Iteración 10000: 46.16% celdas sucias restantes.
Tiempo de ejecución del Escenario 2: 0:00:01.302430
¿Todas las celdas fueron limpiadas? Sí
Número de pasos para completar la limpieza: 167060
```

Determinando el numero optimo de agentes

In [ ]:

```
# Parámetros generales
WIDTH = 100
HEIGHT = 100
INITIAL_DIRT = int(WIDTH * HEIGHT * 0.90)  # 90% de las celdas están sucias
MAX_STEPS = 100000  # Número máximo de pasos para la simulación

def run_simulation(n_agents):
    model = CleaningModel(WIDTH, HEIGHT, INITIAL_DIRT, n_agents, MAX_STEPS)
    start_time = time.time()
    while model.schedule.steps < MAX_STEPS and model.dirty_cells_remaining()
        model.step()
    elapsed_time = time.time() - start_time
    total_steps = model.schedule.steps
    total_cleaned = model.dirty_cells_remaining() == 0
    dirty_percentage = (model.dirty_cells_remaining() / (WIDTH * HEIGHT)) *
    return elapsed_time, total_steps, total_cleaned, dirty_percentage

# Variables para registrar la configuración óptima
optimal_time = float('inf')
optimal_steps = MAX_STEPS
optimal_agents = 0

for n_agents in range(1, 11):  # De 1 a 10 aspiradoras
    elapsed_time, total_steps, all_cleaned, dirty_percentage = run_simulatic
    print(f"{n_agents} aspiradoras: Tiempo = {elapsed_time:.2f} segundos, Pa

    # Actualizar la configuración óptima si se limpia completamente con menc
    if all_cleaned and elapsed_time < optimal_time and total_steps < optimal
        optimal_time = elapsed_time
```

```
        optimal_steps = total_steps
        optimal_agents = n_agents

# Mostrar la configuración óptima
if optimal_agents > 0:
    print(f"Óptimo número de aspiradoras: {optimal_agents}, con {optimal_ste
else:
    print("No se encontró un óptimo número de aspiradoras para completar la
```

```
1 aspiradoras: Tiempo = 0.56 segundos, Pasos = 100000, Limpieza completa = N
o, Porcentaje sucio = 2.61%
2 aspiradoras: Tiempo = 0.73 segundos, Pasos = 100000, Limpieza completa = N
o, Porcentaje sucio = 0.25%
3 aspiradoras: Tiempo = 0.75 segundos, Pasos = 83087, Limpieza completa = S
í, Porcentaje sucio = 0.00%
4 aspiradoras: Tiempo = 1.08 segundos, Pasos = 100000, Limpieza completa = N
o, Porcentaje sucio = 0.14%
5 aspiradoras: Tiempo = 0.62 segundos, Pasos = 42165, Limpieza completa = S
í, Porcentaje sucio = 0.00%
6 aspiradoras: Tiempo = 0.74 segundos, Pasos = 50791, Limpieza completa = S
í, Porcentaje sucio = 0.00%
7 aspiradoras: Tiempo = 0.50 segundos, Pasos = 30259, Limpieza completa = S
í, Porcentaje sucio = 0.00%
8 aspiradoras: Tiempo = 0.85 segundos, Pasos = 47347, Limpieza completa = S
í, Porcentaje sucio = 0.00%
9 aspiradoras: Tiempo = 0.76 segundos, Pasos = 37445, Limpieza completa = S
í, Porcentaje sucio = 0.00%
10 aspiradoras: Tiempo = 0.68 segundos, Pasos = 29309, Limpieza completa = S
í, Porcentaje sucio = 0.00%
Óptimo número de aspiradoras: 7, con 30259 pasos en 0.50 segundos
```

Informe de Simulación de Limpieza con Agentes Autónomos

Resumen

Se llevaron a cabo simulaciones de limpieza en un espacio de 100x100 con diferentes cantidades de agentes de limpieza (aspiradoras) y distintos niveles de suciedad inicial. Se exploraron dos escenarios específicos y luego se realizó una optimización para determinar la cantidad ideal de agentes para la limpieza eficiente del espacio.

Esceario 1

Condiciones iniciales

- 1 agente
- 90% de celdas sucias

Resultados

- Suciedad tras 100 pasos: 89.50%
- Suciedad tras 1000 pasos: 85.72%
- Suciedad tras 10000 pasos: 61.68%
- Tiempo Total de Ejecución: 1.21 segundos
- Limpieza Completa: Sí
- Pasos para la Limpieza Completa: 228,510

Observaciones

Con un solo agente, se observa una disminución gradual pero lenta del porcentaje de suciedad. Aunque finalmente se logra una limpieza completa, el proceso requiere un número significativo de pasos, reflejando una eficiencia relativamente baja.

Esceario 2

Condiciones iniciales

- 1 agente
- 90% de celdas sucias

Resultados

- Suciedad tras 100 pasos: 89.03%
- Suciedad tras 1000 pasos: 83.93%
- Suciedad tras 10000 pasos: 46.16%
- Tiempo Total de Ejecución: 1.30 segundos
- Limpieza Completa: Sí
- Pasos para la Limpieza Completa: 167,060

Observaciones

El aumento a dos agentes de limpieza mejora la eficiencia en comparación con un solo agente, reduciendo el número de pasos necesarios para una limpieza completa. Sin embargo, todavía se requiere una cantidad considerable de tiempo y pasos para alcanzar el objetivo.

Optimización con hasta 10 agentes

Metodo

Se realizó una serie de simulaciones incrementando el número de agentes de 1 a 10 para encontrar la combinación óptima que minimiza tanto el tiempo como el número de pasos necesarios para una limpieza completa.

Resultados optimos

- Número Óptimo de Aspiradoras: 7
- Pasos Óptimos: 30,259
- Tiempo Óptimo: 0.50 segundos

Observaciones generales

- Con 3 a 10 aspiradoras, se logra la limpieza completa dentro del límite de pasos.
- La eficiencia mejora notablemente con más agentes hasta un punto óptimo con 7 aspiradoras, donde se logra el equilibrio entre el menor número de pasos y el tiempo más corto.
- Más allá de 7 agentes, no se observan mejoras significativas en términos de tiempo o pasos, sugiriendo una saturación en la eficiencia debido a la posible congestión o interacciones entre agentes.

Conclusiones

La simulación demuestra que la eficiencia en la limpieza autónoma mejora con el aumento del número de agentes hasta un punto óptimo. En este caso, 7 agentes proporcionan la mejor eficiencia en términos de tiempo y pasos para una limpieza completa. Este hallazgo subraya la importancia del equilibrio entre la cantidad de recursos (agentes) y la eficiencia operativa en tareas de limpieza autónoma.

Enlace al repositorio

https://github.com/DHurtado714-itesm/tc2008b