Instituto Tecnológico de Estudios Superiores de Monterrey
Campus Querétaro

Evidencia 1. Actividad Integradora

Arturo Sánchez Rodríguez - A01275427

Modelación de sistemas multiagentes con gráficas computacionales (Gpo 301)

Introducción

En este proyecto, se simulará el proceso de organización de un almacén utilizando cinco robots equipados con ruedas omnidireccionales y manipuladores para mover cajas. El objetivo es el almacén, que actualmente está en completo desorden, colocando todas las cajas en pilas de exactamente cinco cajas. Para lograrlo, se implementará un modelo multiagente utilizando la biblioteca 'Mesa' en Python.

Desarrollo

Descripción del Proyecto

Cada robot puede moverse en las cuatro direcciones y recoger cajas de celdas adyacentes. Pueden apilar cajas hasta una altura de cinco. La simulación debe distribuir inicialmente 200 cajas en grupos de 1 a 3 en posiciones aleatorias dentro de una cuadrícula de 20x20. Los robots también se colocan en posiciones aleatorias vacías.

La simulación finaliza cuando todas las cajas están apiladas en pilas de exactamente cinco cajas. Se medirán los pasos necesarios para completar la tarea, y se analizarán posibles estrategias para optimizar el número de pasos.

Código por Bloques

Importación de Bibliotecas y Configuración Inicial

```
# Importar las clases necesarias de Mesa y otras bibliotecas
from mesa import Agent, Model
from mesa.space import MultiGrid
from mesa.time import RandomActivation
from mesa.datacollection import DataCollector
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import random

# Establecer la semilla para la generación de números aleatorios
random.seed(67890)
np.random.seed(67890)
```

En este bloque, importamos las bibliotecas necesarias y configuramos la semilla para la generación de números aleatorios, asegurando la reproducibilidad de los resultados.

Definición de la clase robot

```
# Definir la clase de los robots
class Robot(Agent):
    def __init__(self, unique_id, model):
        super().__init__(unique_id, model)
        self.carrying = 0

    def step(self):
        self.move()
        self.pick_or_drop_box()

    def move(self):
        possible_steps = self.model.grid.get_neighborhood(
            self.pos,
            moore=True,
            include_center=False
        )
        new_position = self.random.choice(possible_steps)
        self.model.grid.move_agent(self, new_position)

    def pick_or_drop_box(self):
        x, y = self.pos
        cell_content = self.model.grid.get_cell_list_contents([self.pos])
        boxes_in_cell = [obj for obj in cell_content if isinstance(obj, Box)]

        if self.carrying == 0 and len(boxes_in_cell) > 0:
            # Recoger una caja
            self.carrying += 1
            self.model.grid.remove_agent(boxes_in_cell[0])
            print(f"Robot {self.unique_id} recogió una caja en ({x},{y})")
        elif self.carrying > 0 and len(boxes_in_cell) < 5:
            # Dejar una caja si hay menos de 5 en la celda
            self.carrying -= 1
            new_box = Box(self.model.next_id(), self.model)
            self.model.grid.place_agent(new_box, self.pos)
            print(f"Robot {self.unique_id} dejó una caja en ({x},{y})")
```

Este bloque define la clase ‘Robot’, que incluye métodos para moverse y recoger o dejar cajas. Los robots se mueven a celdas adyacentes y manipulan las cajas según su capacidad de carga

Definición de la Clase Box

```
# Definir la clase de las cajas
class Box(Agent):
    def __init__(self, unique_id, model):
        super().__init__(unique_id, model)
```

Este bloque define la clase 'Box', que representa una caja en el almacén.

Definición de la clase del modelo del almacén

```
# Definir la clase del modelo del almacén
class WarehouseModel(Model):
    def __init__(self, width, height, num_robots, num_boxes):
        super().__init__()  # Inicializar la clase base Model
        self.current_id = 0  # Inicializar el current_id
        self.num_agents = num_robots
        self.grid = MultiGrid(width, height, True)
        self.schedule = RandomActivation(self)
        self.datacollector = DataCollector(model_reporters={"Grid": self.get_grid})
        self.running = True  # Inicializar running

        # Crear robots
        for i in range(self.num_agents):
            robot = Robot(self.next_id(), self)
            x, y = self.get_empty_position()
            self.grid.place_agent(robot, (x, y))
            self.schedule.add(robot)

        # Crear cajas
        boxes_to_place = num_boxes
        while boxes_to_place > 0:
            group_size = min(random.randint(1, 3), boxes_to_place)
            x, y = self.get_empty_position()
            for _ in range(group_size):
                box = Box(self.next_id(), self)
                self.grid.place_agent(box, (x, y))
            boxes_to_place -= group_size

    def get_empty_position(self):
        while True:
            x = self.random.randrange(self.grid.width)
            y = self.random.randrange(self.grid.height)
            if self.grid.is_cell_empty((x, y)):
                return (x, y)

    def step(self):
```

```
        self.datacollector.collect(self)
        self.schedule.step()
        if self.all_boxes_stacked():
            self.running = False

    def all_boxes_stacked(self):
        for (content, (x, y)) in self.grid.coord_iter():
            boxes_in_cell = [obj for obj in content if isinstance(obj, Box)]
            if len(boxes_in_cell) not in (0, 5):
                return False
        return True

    def get_grid(self):
        grid = np.zeros((self.grid.width, self.grid.height))
        for (content, (x, y)) in self.grid.coord_iter():
            grid[x][y] = len([obj for obj in content if isinstance(obj, Box)])
        return grid

    def print_grid(self):
        for (content, (x, y)) in self.grid.coord_iter():
            boxes_in_cell = len([obj for obj in content if isinstance(obj, Box)])
            if boxes_in_cell > 0:
                print(f"Cell ({x},{y}) has {boxes_in_cell} boxes")
```

Este bloque define la clase 'WarehouseModel', que inicializa e modelo del almacén, crea robots y cajas, y define la lógica para verificar si todas las cajas están correctamente apiladas.

Ejecución de la Simulación

```
# Parámetros de la simulación
width = 20
height = 20
num_robots = 5
num_boxes = 200
max_steps = 500

# Inicializar el modelo
model = WarehouseModel(width, height, num_robots, num_boxes)

# Imprimir el estado inicial
print("Estado inicial del almacén:")
model.print_grid()

# Ejecutar la simulación
for i in range(max_steps):
    model.step()
```

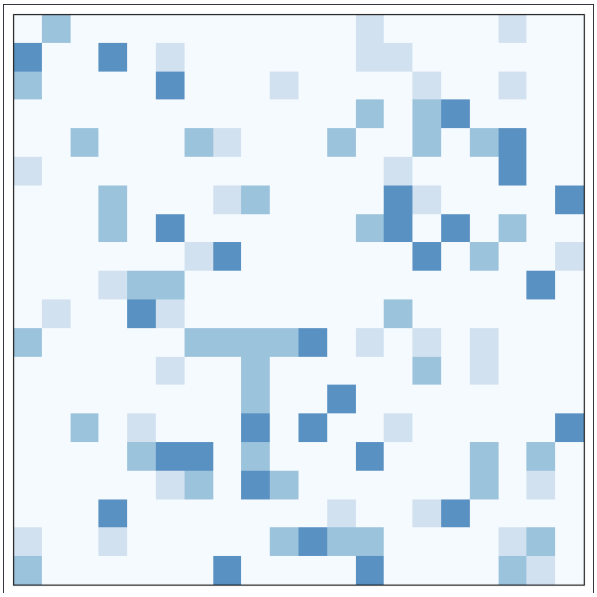
```
    if not model.running:
        break

# Obtener los datos de la simulación
all_grid = model.datacollector.get_model_vars_dataframe()

# Imprimir el estado final
print("\nEstado final del almacén:")
model.print_grid()

# Verificar si todas las pilas tienen exactamente 5 cajas
if model.all_boxes_stacked():
    print("\nTodas las cajas están correctamente apiladas en pilas de 5 cajas.")
else:
    print("\nNo todas las cajas están apiladas correctamente.")

# Visualizar la simulación
fig, ax = plt.subplots(figsize=(8, 8))
ax.set_xticks([])
ax.set_yticks([])
patch = plt.imshow(all_grid.iloc[0][0], cmap=plt.cm.Blues, vmin=0, vmax=5)

def animate(i):
    patch.set_data(all_grid.iloc[i][0])

anim = animation.FuncAnimation(fig, animate, frames=len(all_grid))
plt.show()
```

En este bloque se establecen los parámetros de la simulación, se inicializa el modelo y se ejecuta la simulación hasta que todas las cajas están apiladas correctamente o se alcance el número máximo de pasos. También se visualiza el proceso de la simulación.

Conclusión

El proyecto demuestra cómo utilizar un sistema multiagente para organizar un almacén desordenado. Se logró implementar una simulación donde cinco robots mueven y apilan cajas en un almacén de 20x20 celdas, con el objetivo de apilar todas las cajas en grupos de cinco. La estrategia implementada permitió verificar la eficiencia del sistema y explorar posibles mejoras para reducir el número de pasos necesarios para completar la tarea. Esta simulación puede servir como base para futuros desarrollos en sistemas multiagentes y aplicaciones de logística y organización de almacenes.

Salidas



```
Estado inicial del almacén:
Cell (0,1) has 2 boxes
Cell (0,12) has 1 boxes
Cell (0,17) has 1 boxes
Cell (1,0) has 3 boxes
Cell (1,3) has 3 boxes
Cell (1,5) has 1 boxes
Cell (1,12) has 1 boxes
Cell (1,13) has 1 boxes
Cell (2,0) has 2 boxes
Cell (2,5) has 3 boxes
Cell (2,9) has 1 boxes
Cell (2,14) has 1 boxes
Cell (2,17) has 1 boxes
Cell (3,12) has 2 boxes
Cell (3,14) has 2 boxes
Cell (3,15) has 3 boxes
Cell (4,2) has 2 boxes
Cell (4,6) has 2 boxes
Cell (4,7) has 1 boxes
Cell (4,11) has 2 boxes
Cell (4,14) has 2 boxes
Cell (4,16) has 2 boxes
Cell (4,17) has 3 boxes
Cell (5,0) has 1 boxes
...
Cell (19,16) has 1 boxes
Cell (19,18) has 1 boxes
```

Continua…

Referencias

Bordini, R. H., Hübner, J. F., & Wooldridge, M. (2007). Programming multi-agent systems in AgentSpeak using Jason. Wiley.

Bonabeau, E. (2002). Agent-based modeling: Methods and techniques for simulating human systems. Proceedings of the National Academy of Sciences, 99(suppl 3), 7280-7287.

Gilbert, N., & Troitzsch, K. G. (2005). Simulation for the social scientist. McGraw-Hill Education (UK).

Macal, C. M., & North, M. J. (2010). Tutorial on agent-based modelling and simulation. Journal of Simulation, 4(3), 151-162.

Russell, S. J., & Norvig, P. (2016). Artificial intelligence: A modern approach. Pearson.

Schelling, T. C. (1971). Dynamic models of segregation. Journal of Mathematical Sociology, 1(2), 143-186.

Wilensky, U., & Rand, W. (2015). An introduction to agent-based modeling: Modeling natural, social, and engineered complex systems with NetLogo. MIT Press.

Yang, Y., Guo, X., & Shu, H. (2016). A multi-agent-based simulation system for intralogistics in an industrial warehouse. International Journal of Simulation Modelling, 15(2), 328-339.