Campus Querétaro

Materia:

Modelación de sistemas multiagentes con gráficas computacionales

(Gpo 301)

Actividad Integradora

Profesor:

Pedro Perez

Alumno:

Carlos Isaac Dávalos Lomelí A01706041

Descripción

Felicidades! Eres el orgulloso propietario de 5 robots nuevos y un almacén lleno de cajas. El dueño anterior del almacén lo dejó en completo desorden, por lo que depende de tus robots organizar las cajas en algo parecido al orden y convertirlo en un negocio exitoso.

Cada robot está equipado con ruedas omnidireccionales y, por lo tanto, puede conducir en las cuatro direcciones. Pueden recoger cajas en celdas de cuadrícula adyacentes con sus manipuladores, luego llevarlas a otra ubicación e incluso construir pilas de hasta cinco cajas. Todos los robots están equipados con la tecnología de sensores más nueva que les permite recibir datos de sensores de las cuatro celdas adyacentes. Por tanto, es fácil distinguir si un campo está libre, es una pared, contiene una pila de cajas (y cuantas cajas hay en la pila) o está ocupado por otro robot. Los robots también tienen sensores de presión equipados que les indican si llevan una caja en ese momento.

Lamentablemente, tu presupuesto resultó insuficiente para adquirir un software de gestión de agentes múltiples de última generación. Pero eso no debería ser un gran problema ... ¿verdad? Tu tarea es enseñar a sus robots cómo ordenar su almacén. La organización de los agentes depende de ti, siempre que todas las cajas terminen en pilas ordenadas de cinco.

Estrategia

La estrategia principal utilizada en esta solución se basa en la colaboración y coordinación de los robots para recoger y apilar cajas de manera eficiente. Cada robot es capaz de moverse en las cuatro direcciones (arriba, abajo, izquierda y derecha) y puede recoger cajas de las celdas adyacentes. Los robots utilizan sensores para determinar el estado de las celdas adyacentes y tomar decisiones sobre dónde moverse y dónde apilar las cajas.

Cada robot utiliza sensores para detectar si las celdas adyacentes están vacías, contienen una caja, o forman parte de una pila, además de saber si llevan una caja en ese momento.

Movimiento y Recogida de Cajas:
Los robots se mueven aleatoriamente al principio, recogiendo cajas y dirigiéndose hacia el centro del almacén para comenzar a apilarlas.

Construcción de Pilas:
Las cajas se apilan en el centro del almacén, formando pilas de hasta cinco cajas. Una vez que una pila está completa, los robots evitan añadir más cajas a esa pila.

Terminación de la Simulación:
La simulación concluye cuando todas las cajas están apiladas en pilas de cinco, o se alcanza el tiempo máximo de ejecución para evitar bucles infinitos.

```python
# Importar Librerías
from mesa import Agent, Model
from mesa.space import MultiGrid
from mesa.time import SimultaneousActivation
from mesa.datacollection import DataCollector

import matplotlib.pyplot as plt
import matplotlib.animation as animation
plt.rcParams["animation.html"] = "jshtml"

import numpy as np
import pandas as pd
import random

import time
import datetime
```

Esta celda importa todas las librerías necesarias para la simulación, incluyendo Mesa para la simulación multiagente, NumPy para el manejo de matrices, y Matplotlib para la visualización.

```python
# Variables Globales y Funciones Auxiliares
                                    l funcionamiento del modelo
int: robot_movements
12393
robot_movements = 0

# Función que regresa la habitación o almacén
def get_room(model):
    room = np.zeros((model.grid.width, model.grid.height))
    for cell_content, (x, y) in model.grid.coord_iter():
        for content in cell_content:
            if isinstance(content, Robot):
                room[x][y] = content.robot_state
            else:
                room[x][y] = content.state
    return room
```

En esta celda definimos variables globales que nos ayudarán a llevar un registro del número de cajas, pilas y movimientos realizados por los robots. También se define la función get_room, que devuelve el estado actual del almacén.

```
[5]  # Agente Piso
     class Floor(Agent):
         FULL_STACK = 3
         STACK = 2
         BOX = 1
         EMPTY = 0

         # Constructor para inicializar el agente
         def __init__(self, pos, model, state=EMPTY):
             super().__init__(pos, model)
             self.x, self.y = pos
             self.state = state
             self.next_state = None
```

Esta celda define la clase Floor, que representa las celdas del almacén. La clase incluye un constructor para inicializar el estado de la celda.

```
0 s  # Modelo Habitación
     class Room(Model):
         global boxes_in_stack

         # Constructor para inicializar el modelo
         def __init__(self, m, n, num_agents, num_boxes):
             super().__init__()
             self.num_agents = num_agents
             self.num_boxes = num_boxes
             self.grid = MultiGrid(m, n, False)
             self.schedule = SimultaneousActivation(self)

             empty_cells_list = list(self.grid.empties)
             for _ in range(num_boxes):
                 empty_cell = random.choice(empty_cells_list)
                 floor = Floor(empty_cell, self)
                 floor.state = floor.BOX
                 self.grid.place_agent(floor, empty_cell)
                 self.schedule.add(floor)
                 empty_cells_list.remove(empty_cell)

             empty_cells_list = list(self.grid.empties)
             for cell in empty_cells_list:
                 floor = Floor(cell, self)
                 self.grid.place_agent(floor, cell)
                 self.schedule.add(floor)

             for i in range(num_agents):

✓ 4 min, 38 s   se ejecutó 12:47 p.m.
```

En esta celda se define la clase Room, que representa el almacén completo. El constructor inicializa la cuadrícula, coloca las cajas y los robots en posiciones aleatorias. La clase también incluye métodos para realizar los pasos de la simulación (step)

```python
# Ejecutar el Modelo
# Datos de la habitación:
M = 20
N = 20

# Número de agentes:
NUM_AGENTS = 5

# Número de cajas:
BOXES = 200

# Tiempo máximo de ejecución (segundos):
MAX_EXECUTION_TIME = 5

start_time = time.time()
model = Room(M, N, NUM_AGENTS, BOXES)

while (time.time() - start_time) < MAX_EXECUTION_TIME and not model.all_boxes_stacked():
    model.step()

# Imprimir el tiempo de ejecución y movimientos totales
execution_time = str(datetime.timedelta(seconds=(time.time() - start_time)))
print("Execution time: " + execution_time)
print("Total movements: " + str(robot_movements))

all_rooms = model.data_collector.get_model_vars_dataframe()
```

```
Execution time: 0:00:05.036697
Total movements: 12393
```

4 min, 38 s se ejecutó 12:47 p.m.

Esta celda configura y ejecuta la simulación. Inicializa el modelo con las dimensiones del almacén el número de robots y el número de cajas. Luego ejecuta la simulación hasta que todas las cajas estén apiladas correctamente o se alcance el tiempo máximo de ejecución. Finalmente, imprime el tiempo de ejecución y el número total de movimientos realizados por los robots.

```python
# Capturar y Animar el Gráfico
fig, axs = plt.subplots(figsize=(10, 10))
axs.set_xticks([])
axs.set_yticks([])
patch = plt.imshow(all_rooms.iloc[0][0], cmap='PuBuGn')
plt.colorbar(patch)

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

anim = animation.FuncAnimation(fig, animate, frames=len(all_rooms))

anim
```

```
WARNING:matplotlib.animation:Animation size has reached 20981885 bytes, exceeding the limit of 20971520.0.
```

En esta celda se genera una visualización de la simulación. Utiliza Matplotlib para crear una animación que muestra el estado del almacén en cada paso de la simulación.
Esta animación ayuda a visualizar cómo los robots organizan las cajas a lo largo del tiempo.

Reflexión

¿Existe una forma de reducir el número de pasos utilizados? Si es así, ¿cuál es la estrategia que se tendría en implementar?

Si, para reducir el número de pasos utilizados, podemos asignar zonas específicas del almacén a cada robot. Cada robot se encargará de recoger y apilar cajas solo dentro de su zona designada. Esto evitará movimientos innecesarios y redundantes, ya que cada robot trabajará en un área definida, mejorando la eficiencia y disminuyendo el tiempo total de la simulación.