Reporte de Actividad Integradora

Alumno:

Jorge Emiliano Pomar Mendoza | A01709338

Modelación de sistemas multiagentes con gráficas computacionales (Gpo 301)

Profesor:

Tecnológico de Monterrey, campus Querétaro

Febrero- Junio 2024

03 de junio del 2024

índice

1. Introducción

En esta actividad se simula un almacén con robots que deben apilar cajas en pilas de 5 cajas para de esta manera asegurarnos que el espacio del almacén quede lo más limpio posible. Para lograr esto, tengo a mi disposición 5 robots a los que les puedo enseñar a moverse en las 4 direcciones y pueden cargar una caja a la vez. El objetivo es que todas las cajas queden apiladas en pilas de no más de 5 cajas.

- - 20x20: Tamaño del almacén
- - 5: Número de cajas que contiene cada pila
- - 5: Número de robots (agentes)
- - 200 cajas posicionadas aleatoriamente en el almacén
- - Robots se mueven en las 4 direcciones de manera aleatoria
- - Robots pueden cargar una caja a la vez
- - La semilla para la generación de números aleatorios es 67890
- - Las cajas están en grupos de 1 a 3 cajas

Enlace al repositorio

https://github.com/pomaremiliano/ActIntegradora

2. Agentes Robot y Caja

Para esto se realizó un programa en python utilizando agentes de la librería de mesa. Cómo primer paso definí a los agentes. En este caso el agente robot de limpieza y el agente de caja. El agente robot empezó en un punto aleatorio del grid. Las cosas que podía saber el robot eran:

1. El vecindario de adyacencias
2. Los posibles movimientos
3. En qué celdas se encuentran las cajas
4. Las celdas que se encuentran vacías
5. Cómo cargar la caja
6. Cómo descargar la caja en otra celda

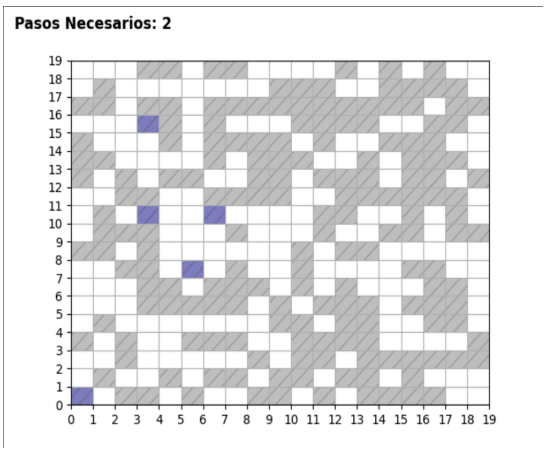
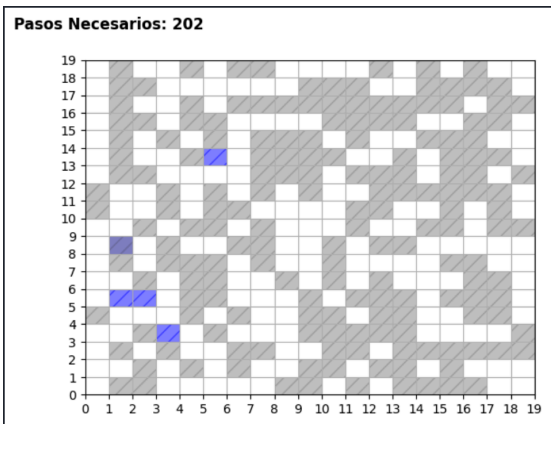
```python
# agente RobotLimpieza
class RobotLimpieza(Agent):
    def __init__(self, unique_id, model):
        super().__init__(unique_id, model)
        self.cargar = []

    # Función para moverse de manera aleatoria en un rango de posibles celdas
    def AvanzaCelda(self):
        possible_steps = self.model.grid.get_neighborhood(
            self.pos,
            moore=False,
            include_center=False)
        possible_steps = [step for step in possible_steps if self.model.grid.is_cell_empty(step)]
        if possible_steps:
            new_position = self.random.choice(possible_steps)
            self.model.grid.move_agent(self, new_position)

    # Función para encontrar la celda con cajas
    def CeldaCaja(self):
        for cell in self.model.grid.coord_iter():
            cell_content, (x, y) = cell
            if any(isinstance(obj, Caja) for obj in cell_content):
                return(x, y)
        return None

    # Función para encontrar la celda vacía p con menos de 5 cajas apiladas
    def CeldaVacia(self):
        for cell in self.model.grid.coord_iter():
            cell_content, (x, y) = cell
            nCajas = sum(1 for obj in cell_content if isinstance(obj, Caja))
            if nCajas < 5:
                return (x, y)
        return None

    # Función para encontrar y cargar cajas
    def CargarCaja(self):
        x, y = self.pos
        for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
            neighbor = (x + dx, y + dy)
            if self.model.grid.out_of_bounds(neighbor):
                continue
            neighbor_content = self.model.grid.get_cell_list_contents([(neighbor)])
            for obj in neighbor_content:
                if isinstance(obj, Caja) and len(self.cargar) < 5:
                    self.cargar.append(obj)
                    self.model.grid.remove_agent(obj)
                    return

    # Función para descargar cajas en celdas vacías
    def DescargarCaja(self):
        if len(self.cargar) > 0:
            caja = self.cargar.pop()
            self.model.grid.place_agent(caja, self.pos)

    # cada step, el robot se mueve, carga y descarga cajas
    def step(self):
        if len(self.cargar) == 0:
            self.CargarCaja()
            self.AvanzaCelda()
        else:
            self.AvanzaCelda()
            self.DescargarCaja()

# agente Caja
class Caja(Agent):
    def __init__(self, unique_id, model):
        super().__init__(unique_id, model)
```

3. Modelo del Grid

El grid 20x20 empezó con 200 cajas colocadas de manera aleatoria y 5 agentes colocados de manera aleatoria. Después, tuve que volver a definir las adyacencias, pero esta vez para saber las cajas que estaban todavía sin apilar o que no cumplían con el requisito de que tienen que ser mínimo 5 cajas apiladas. Y con el conocimiento de quienes en las adyacencias ya tienen las cajas apiladas, los robots entonces pueden llegar a eficientizar las cosas al no ir a recoger cajas donde ya se cumplió el requisito. Y al mismo tiempo, puedo recopilar la estadística de cuantas cajas no han sido apiladas en cada paso de la simulación.

```python
# modelo AlmacenModel que define el grid y la cantidad de agentes
class AlmacenModel(Model):
    def __init__(self, width, height, N, nCajas):
        self.num_agents = N
        self.grid = MultiGrid(width, height, False)
        self.schedule = RandomActivation(self)
        self.running = True
        self.steps = 0 # empezamos a contar los pasos

        # Creamos los agentes y los ponemos en el grid de manera aleatoria
        for i in range(self.num_agents):
            a = RobotLimpieza(i, self)
            self.schedule.add(a)
            x = self.random.randrange(self.grid.width)
            y = self.random.randrange(self.grid.height)
            while not self.grid.is_cell_empty((x, y)):
                x = self.random.randrange(self.grid.width)
                y = self.random.randrange(self.grid.height)
            self.grid.place_agent(a, (x, y))

        # Creamos las cajas y las ponemos en el grid de manera aleatoria
        for i in range(nCajas):
            c = Caja(i + self.num_agents, self)
            self.schedule.add(c)
            x = self.random.randrange(self.grid.width)
            y = self.random.randrange(self.grid.height)
            while not self.grid.is_cell_empty((x, y)):
                x = self.random.randrange(self.grid.width)
                y = self.random.randrange(self.grid.height)
            self.grid.place_agent(c, (x, y))

    # Función para contar las cajas adyacentes
    def adyacencias(self, x, y):
        count = 0
        for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
            neighbor = (x + dx, y + dy)
            if not self.grid.out_of_bounds(neighbor):
                neighbor_content = self.grid.get_cell_list_contents([neighbor])
                count += sum(1 for obj in neighbor_content if isinstance(obj, Caja))
        return count

    # utilizando la función adyacencias, se verifica si hay menos de 5 cajas apiladas
    def step(self):
        self.schedule.step()
        self.steps += 1
        self.no_apiladas = []
        for cell in self.grid.coord_iter():
            cell_content, (x, y) = cell
            nCajas = sum(1 for obj in cell_content if isinstance(obj, Caja))
            if (nCajas > 0 and self.adyacencias(x, y) < 5): # si hay cajas y menos de 5 cajas apiladas, se añade a la lista de cajas no apiladas
                self.no_apiladas.append((x, y))
        if len(self.no_apiladas) == 0: # si todas las cajas ya se apilaron en 5 o mas, se detiene la simulación
            self.running = False
```

4. Resultados

Inicio	Fin
	

```
Simulación en 202 pasos.
Robots restantes: 5
Cajas restantes: 196
```

En la tabla de arriba, se puede observar el comportamiento de los 5 agentes. Durante 200 pasos, los agentes pudieron cumplir con el actuar que se les enseñó en el código, es decir, pudieron llevar a cabo los movimientos aleatorios, pudieron recoger cajas, pudieron analizar los vecinos y el perímetro adyacente y pudieron colocar las cajas en otro punto. Igualmente, en la simulación que está disponible en el jupyter notebook, se puede observar que los agentes si llegan a respetar las pilas hechas y llegan a guardar distancias con el trabajo de otros agentes para poder hacer los movimientos más ágiles. Sin embargo, en 200 pasos (200 frames) no se logró que todas las celdas fueran apiladas con la condición de más de 5. Creo que esto se debe a los siguientes puntos:

1. Una mala implementación para la recopilación de datos de las cajas que no se han apilado. Ya que después de 200 pasos, los resultados de esa estadística son alarmantes y poco creíbles. No puedo creer que 196 cajas todavía no se han apilado, sobre todo porque la simulación permite ver pilas que sí han sido apiladas.

2. La tarea de limpiar esta cantidad de cajas definitivamente necesitaba de una implementación más eficiente. Aunque siempre busqué que los agentes cuidaran las cajas que ya estaban apiladas y que reconocieran lo mejor posibles el perímetro. Al final esto no dió los resultados esperados porque la simulación tardó mucho tiempo (3 minutos aproximadamente) y los agentes no lograron limpiar todo el grid.