Tecnológico de Monterrey

# Almacén de Cajas

**Luis Gabriel Delfín Paulín**

**A01701482**

**03/06/2024**

## 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.

- Realiza la siguiente simulación:
  - Inicializa las posiciones iniciales de las K cajas. Todas las cajas están a nivel de piso, es decir, no hay pilas de cajas.
  - Todos los agentes empiezan en posición aleatorias vacías.
  - Se ejecuta el tiempo máximo establecido.
- Deberás recopilar la siguiente información durante la ejecución:
  - Tiempo necesario hasta que todas las cajas están en pilas de máximo 5 cajas.
  - Número de movimientos realizados por todos los robots.

## Codificación

Al ser un código extenso no vamos a ver cada parte del código, en caso particular de querer ver qué hace cada parte del código, se dejó un comentario con una pequeña explicación.

Por el momento vamos a explicar cada sección (celda) del código:

**Definir la clase del agente robot**

```
# Definir la clase del agente robot
class RobotAgent(Agent):
    def __init__(self, unique_id, model):
        super().__init__(unique_id, model)
        self.has_box = False
        self.box_stack = 0
        self.steps_taken = 0

    # Movimiento del agente robot mediante las celdas vecinas de Von Newmann
    def move(self):
        possible_steps = self.model.grid.get_neighborhood(
            self.pos,
            moore=False,
            include_center=False
        )
        new_position = self.random.choice(possible_steps)
        self.model.grid.move_agent(self, new_position)
        self.steps_taken += 1
```

En esta clase definimos los movimientos del agente robot a celdas vecinas de Von Neumann y recogemos y dejamos cajas, dependiendo de la cantidad de cajas que el agente tenga.

**Definir la clase del agente robot**

```
# Definir la clase del agente caja
class BoxAgent(Agent):
    def __init__(self, unique_id, model):
        super().__init__(unique_id, model)
```

Esta clase se hizo para tener mayor facilidad al manejar las cajas.

**Definir la clase del modelo**

```
# Definir la clase del modelo
class WarehouseModel(Model):
    def __init__(self, width, height, num_boxes, num_robots, max_stack_height, max_time_steps):
        super().__init__()
        self.width = width
        self.height = height
        self.num_boxes = num_boxes
        self.num_robots = num_robots
        self.max_stack_height = max_stack_height
        self.max_time_steps = max_time_steps
        self.num_stacks = num_boxes // max_stack_height
        self.grid = MultiGrid(width, height, True)
        self.schedule = RandomActivation(self)
        self.running = True
        self.total_moves = 0
        self.stacks_created = 0
```

En esta clase inicializamos los parámetros dados y agregamos los agentes de robots y cajas al modelo. Al igual que tenemos los contadores de movimiento.

**Parámetros de la simulación**

```
# Parámetros de la simulación
width = 10
height = 10
num_boxes = 20
num_robots = 5
max_stack_height = 5
max_time_steps = 100
```

Aquí es donde modificamos cualquier aspecto de nuestra simulación

**Ejecución de la simulación**

```
# Ejecutar la simulación
model = WarehouseModel(width, height, num_boxes, num_robots, max_stack_height, max_time_steps)

while model.running:
    model.step()

# Imprimir los resultados
print("Número total de movimientos realizados por todos los robots:", model.total_moves)
print("Número de montones creados:", model.stacks_created)
```

Finalmente damos los resultados obtenidos

**Resultados de la simulación**

```
Número total de movimientos realizados por todos los robots: 90
Número de montones creados: 4
```

Como podemos observar, se crearon 4 montones (5 cajas por montón, lo que nos da un total de las 20 cajas), lo que significa que aquí acabó nuestra simulación.

**Github**

https://github.com/Gabriel-Delfin/AlmacenCajas

**Reflexión**

Como reflexión me pongo a pensar en cómo podríamos disminuir el tiempo de esta simulación para que los robots acaben mucho antes. Una estrategia que se me ocurre es que cada que los robots visiten una celda e identifiquen las celdas vecinas, notifiquen a los demás robots que hasta el momento no hay cajas para visitar en esas celdas o que ya se recogieron, por lo que no valdría la pena visitarlas. Esto nos podría ahorrar tiempo de ejecución y hacer nuestros agentes más efectivos.