Modelación de Sistemas Multiagentes

Prof. Edgar Covantes Osuna

Prof. Jorge Cruz Duarte

Actividad Integradora

Esteban De la Maza Castro A01720548

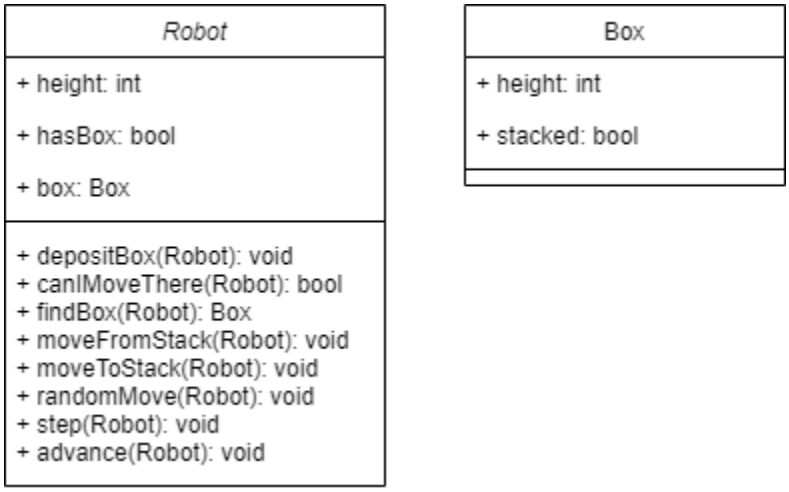
2 de septiembre del 2021

Esteban De la Maza Castro A01720548

Estrategia para Mejorar Rendimiento

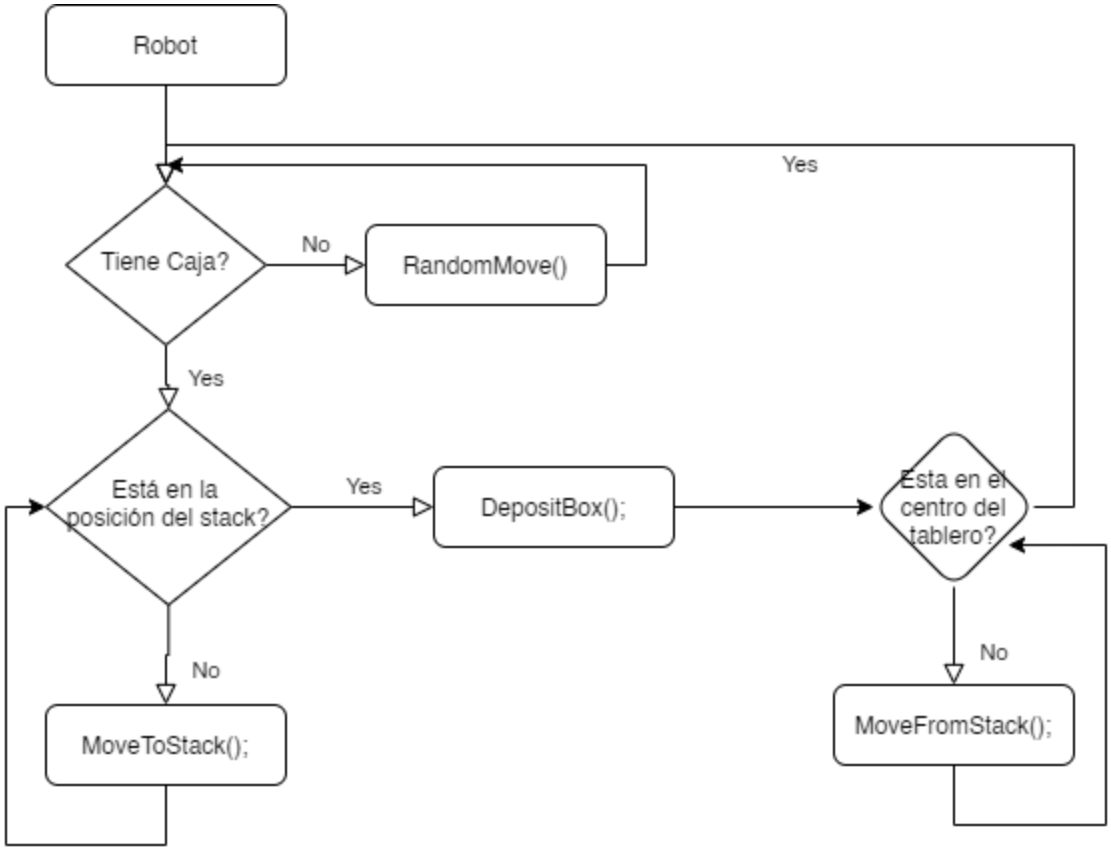
Tras haber realizado mi simulación y cumplido con todos los requisitos, creo que sí existen maneras para mejorar el rendimiento de la simulación. Lo primero que se me viene a la mente sería que los robots tengan la habilidad de leer y moverse en diagonal, pues hay muchos movimientos extras que se realizan debido a esta limitación que se nos dio. Otra parte importante que creo que mejoraría el sistema es poder agarrar más de una caja a la vez. Esta limitación causó que muchos de los robots pasen por al lado o sobre la posición de cajas que podría recoger, pero solo tienen la capacidad de 1. Creo que, si esta limitación se levantara, se reducirán los pasos. Finalmente, otro comportamiento que siento que ayudaría a la simulación sería que tuvieran la capacidad de encontrar cajas a mayor distancia. Debido a que solo tienen un radio de 1 caja, los robots pueden estar rodeando una caja varios turnos y no la logran captar debido a esta limitación. Si tuvieran mejor visión o sensores esta nueva capacidad ayudaría mucho al rendimiento.

Diagrama de Clases



Protocolo de Agentes

Diagrama de Flujo



Estrategia Cooperativa

El sistema se maneja de la siguiente manera. En un inicio, todas las cajas aparecen en posiciones aleatorias generadas por una matriz. Posteriormente, se crean 5 instancias de robots también ubicados en posiciones aleatorias. Las posiciones se asignan dentro de esta matriz. Si hay un 0 en alguna posición, no se inserta ningún agente, sin embargo si existe un 1, se inserta una caja. En el caso de que el valor sea un 2, se inserta un robot. De esta forma se asegura que no van a compartir posiciones en el Grid y todos van a aparecer en posiciones diferentes.

```
gridSize = width*height
# print(total)
arrayPositions = np.zeros((width,height), int)
x = 0
while x < K:
  pos = np.random.randint(0,width-1)
  pos2 = np.random.randint(0,height-1)
  if arrayPositions[pos][pos2] == 0:
    arrayPositions[pos][pos2] = 1
    x+=1

y = 0
while y < 5:
  pos = np.random.randint(0,width-1)
  pos2 = np.random.randint(0,height-1)
  if arrayPositions[pos][pos2] == 0:
    arrayPositions[pos][pos2] = 2
    y+=1
print(arrayPositions)
for (content, x, y) in self.grid.coord_iter():
  if arrayPositions[x][y] == 2:
    a = Robot((x, y), self, 6)
    self.grid.place_agent(a, (x,y))
    self.schedule.add(a)
  if arrayPositions[x][y] == 1:
    b = Box((x, y), self, 1)
    self.grid.place_agent(b, (x,y))
    self.schedule.add(b)
```

Posteriormente, los robots comienzan su búsqueda de cajas. Se estarán moviendo en posiciones aleatorias, para así tratar de encontrar una. Esta tarea se realiza a través del método de FindBox(), donde en caso de encontrarse con un agente tipo Caja en cualquiera de sus 4 sentidos (arriba, abajo, izquierda o derecha), capturará su información y lo igualará a su atributo de tipo Box. En caso de que no encuentre, este valor seguirá siendo igual a -1, para indicar que no logro identificar una caja. En cada iteración del modelo se pregunta si encuentra una caja si es que no tiene alguna caja actualmente.

```
def step(self):
  if self.hasBox == 0:
    self.box = self.findBox()
    if self.box != -1:
      self.hasBox = 1
      self.model.grid.remove_agent(self.box)
```

En caso de que ya tenga caja, el robot tendrá que desplazarse hacia la posición del stack que aún no está compuesto por 5 cajas y subir hasta la primera fila. Aquí es donde entra en vigor la función de MoveToStack() y posteriormente DepositBox(). Estas trabajan en conjunto para lograr que los robots suelten las cajas que poseen y sigan con su tarea de ordenar todas. Una

vez que se depositan las cajas, es donde se utiliza la función de MoveFromStack(). Lo que realiza es un desplazamiento hacia el centro del tablero para después comenzar a moverse de manera aleatoria de nuevo. Cada vez que se realiza un movimiento, anteriormente se realiza una verificación para ver si no hay algún robot en esa posición nueva actualmente a través de la función de canIMoveThere().

```
def advance(self):
    if self.hasBox == 1:
        self.moveToStack()
    elif self.hasBox == 2:
        self.moveFromStack()
    else:
        self.random_move()
```

Aquí es donde se realiza esa comprobación de en qué estado se encuentra el agente tipo Robot. Si su atributo de hasBox es 1, significa que aún tiene una caja que debe de ir a depositar, entonces se utiliza el método de moveToStack(). Si es igual a 2, significa que acaba de realizar este depósito y debe desplazarse hacia el centro del tablero, de lo contrario, deberá de seguirse moviendo de forma aleatoria.

Cada vez que se ordena una caja, se modifica el valor de una variable global que se utiliza posteriormente dentro del método de allStacked(). Este realiza una verificación para saber qué porcentaje de las cajas totales han sido ordenadas para así entregar este valor al usuario, o bien parar la simulación una vez que se ordenen el 100% de las cajas.

Finalmente, para mostrar el resultado de la simulación, se envía el Grid a través de la función de getGrid() y se grafica con la librería de matplot.