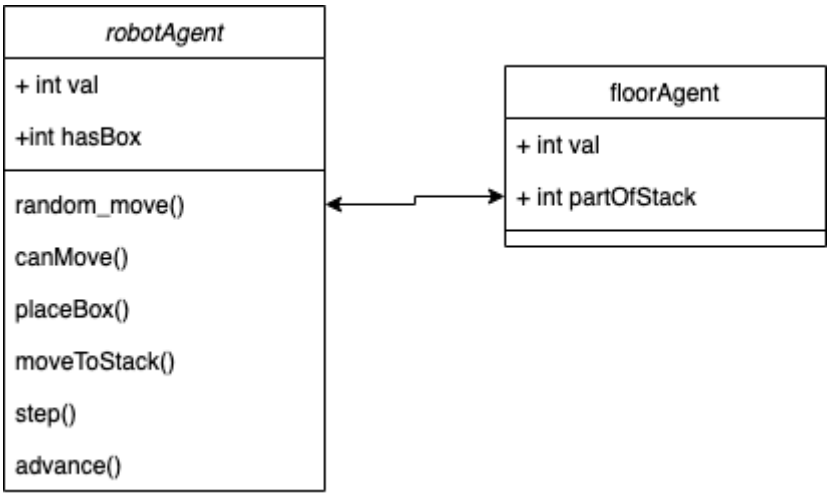
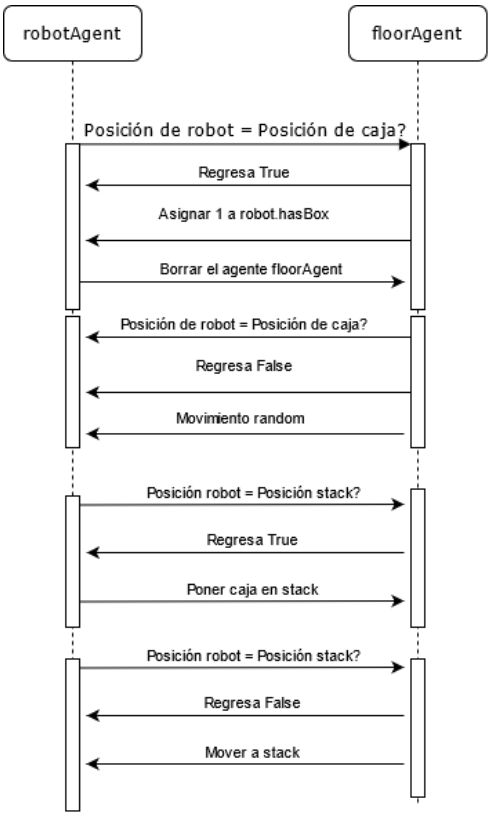
Ricardo Jasso Azzario A01720623

Actividad Integradora

DIAGRAMAS DE CLASES



PROTOCOLO DE AGENTES



Ricardo Jasso Azzario A01720623

ESTRATEGIA PARA LA SOLUCIÓN

Para esta Actividad Integradora, llegué a la solución utilizando 2 agentes. El primer agente es el de Robot y el segundo el de Piso. Primeramente, lo que hice fue, tomando en cuenta los parámetros de número de robots y número de cajas, crear una lista de agentes que se llenaría de manera aleatoria con robots y cajas, asignando valores para cada uno en la lista.

```
[[1 0 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 0 6 0 0]
 [6 0 1 0 0 1 1 1 0 0]
 [0 0 1 0 1 0 0 0 0 0]
 [0 1 6 0 0 0 0 1 0 0]
 [1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 1 0 0 0 0 0 0 0]
 [0 0 1 0 0 6 1 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]]
```

(El valor de 1 indica que el agente es una caja, 6 indica que el agente es un robot.)

Una vez teniendo esta lista, se crearon los agentes y se colocaron en el grid.

Para la simulación, lo que hice fue primero checar el valor de **hasBox** del robot. Si el valor es 0, significa que aún no tiene una caja y por consecuencia se llama la función de **random_move**() para que se mueva aleatoriamente hasta encontrar una caja.

Una vez que la encuentra, **hasBox** cambia a 1, se elimina el agente de la caja que agarró y se llama la función de **moveToStack**().

```
for neighbor in neighbours:
  if neighbor.val == 1 and neighbor.partOfStack == 0 and self.hasBox == 0:
    self.hasBox = 1
    self.model.grid.remove_agent(neighbor)
```

```
def advance(self):
    #if i have a box, go deposit, else random
      if(self.hasBox == 1):
        self.moveToStack()
      else:
        self.random_move()
```

moveToStack() hace que el robot se mueva a donde está el stack. Sabemos dónde está el stack por una variable global de tipo entero llamada **currentStack**. **currentStack** está inicializada en 0 y tomamos este valor como la posición en 'y' de la lista. Entonces, el robot sabe que debe moverse hasta que su posición sea [0][**currentStack**].

```
def moveToStack(self):
  global currentStack
  newPosx = self.pos[0]
  newPosy = self.pos[1]
  if self.pos[0] != 0 and self.canMove((self.pos[0] - 1, self.pos[1])):
    newPosx = self.pos[0] - 1
    self.model.grid.move_agent(self, (newPosx, newPosy))
  if self.pos[1] != currentStack and self.canMove((self.pos[0], self.pos[1] - 1)):
    newPosy = self.pos[1] - 1
    self.model.grid.move_agent(self, (newPosx, newPosy))
  if self.pos[0] == 0 and self.pos[1] == currentStack:
    self.placeBox()
```

Una vez que el robot llega a esta posición, se llama la función de **placeBox**() la cual crea un agente en esa misma posición y se le suma uno a **countOfBoxes** la cual mantiene el control del stack. Cuando **countOfBoxes** es igual a 5, se le suma 1 a **currentStack**, se le asigna 0 a **countOfBoxes** y vuelve a empezar todo el algoritmo pero con la nueva posición de [0][**currentStack**+1]. También, se le va sumando a **stackedBoxes**, una variable global que va contando el total de cajas puestas en cualquier stack, para tener una manera de terminar la simulación cuando esta variable sea igual al número de cajas inicializadas.

```
def placeBox(self):
  global currentStack, stackedBoxes, countOfBoxes
  self.hasBox = 0
  b = floorAgent(self.pos, self, countOfBoxes+1)
  b.partOfStack = 1
  self.model.grid.place_agent(b, self.pos)
  countOfBoxes+=1
  stackedBoxes+=1
  if countOfBoxes == 5:
    currentStack += 1
    countOfBoxes = 0
```

Finalmente, la simulación termina cuando **stackedBoxes** es igual al número de cajas iniciales y el tiempo de ejecución llega al asignado.

```
def stacked(self):
  global stackedBoxes
  if stackedBoxes == self.numBoxes:
    return True
  return False
```

```
while(time.time()-start_time) < MAX_RUN and model.stacked() == False:
  movimientos += 1
  model.step()
```

Este while loop nos dice que mientras el tiempo sea menor que el tiempo de ejecución, y aún no se juntan todas las cajas en stacks, siga corriendo y avanzando el programa.