

Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Puebla

Modelación de sistemas multiagentes con gráficas computacionales

(Gpo 1)

TC2008B.1

Luciano García Bañuelos

Jose Eduardo Ferrer Cruz

Actividad Integradora

Sahid Emmanuel Rosas Maas

A01734211

29 de noviembre 2021

Actividad Integradora (Almacén)

Introducción

Para la situación problema se nos fue asignado un almacén en desorden, y nuestro deber cómo dueños es usar agentes inteligentes para reorganizarlo y empezar nuestro negocio. Lamentablemente, no hubo presupuesto para comprar el software necesario que nos ayude a realizar esta tarea, por lo cual nuestro objetivo principal es crear un programa que nos permita localizar las cajas dispersas en el almacén y mover a los robots de tal manera que apilen las cajas en los lugares designados.

Con el propósito de cumplir con los parámetros antes mencionados se usaron las librerías Mesa y pathfinding de Python al igual que el motor gráfico Unity para ilustrar los resultados del programa. Se registraron variables como el número de pasos de cada robot y el tiempo que les toma recoger y apilar todas las cajas.

Repositorio de Github

<https://github.com/SahidDev/Warehouse-Sim>

Videos

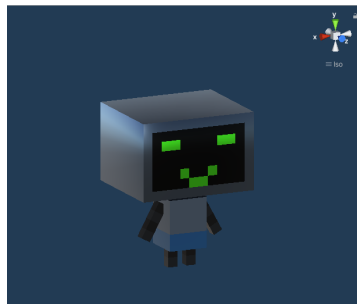
Evidencia de Mesa

<https://youtu.be/DCBYgjKYWQY>

Evidencia de Unity

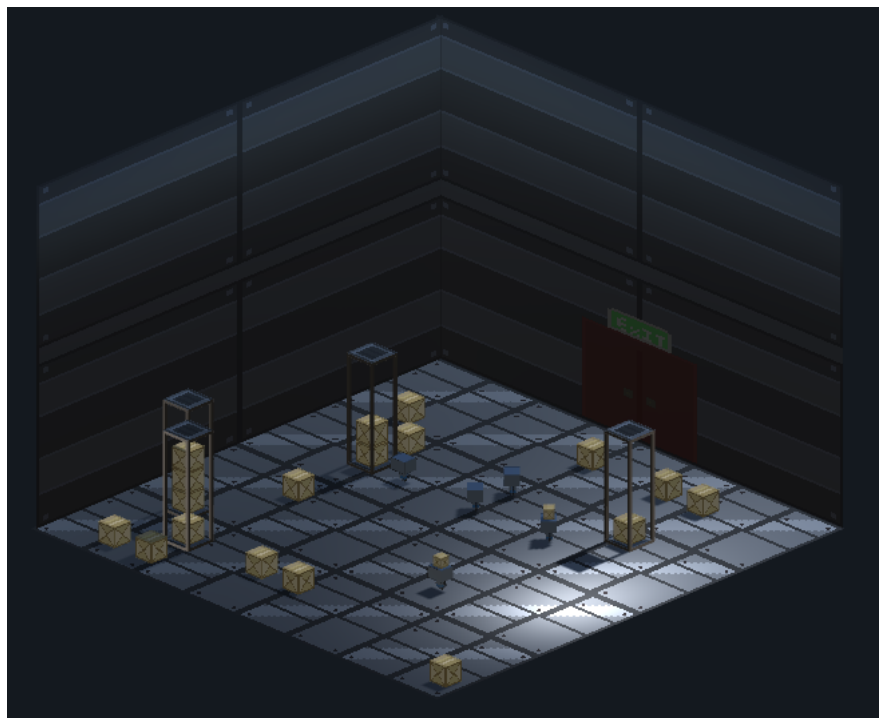
<https://youtu.be/HIZbHKy2e0U>

Agentes

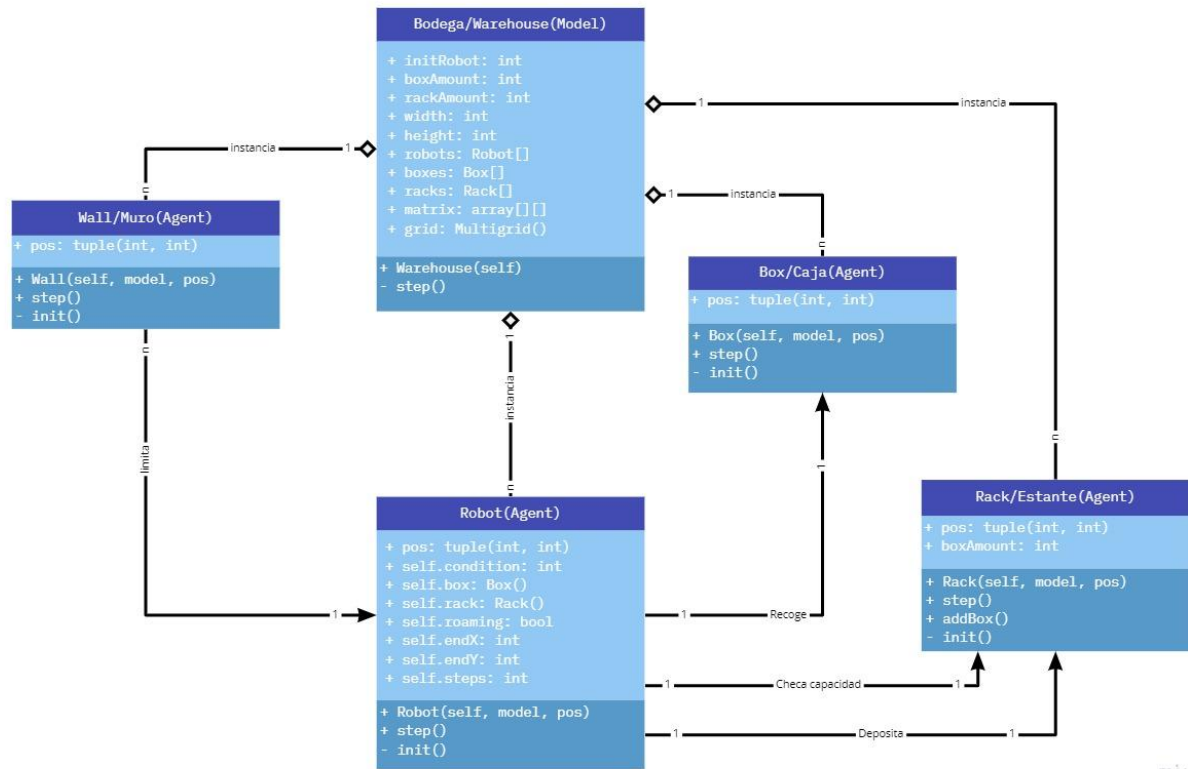
Agente	Modelo 3D
Robot	

Caja	
Estante	
Muro	

Modelo (Bodega)

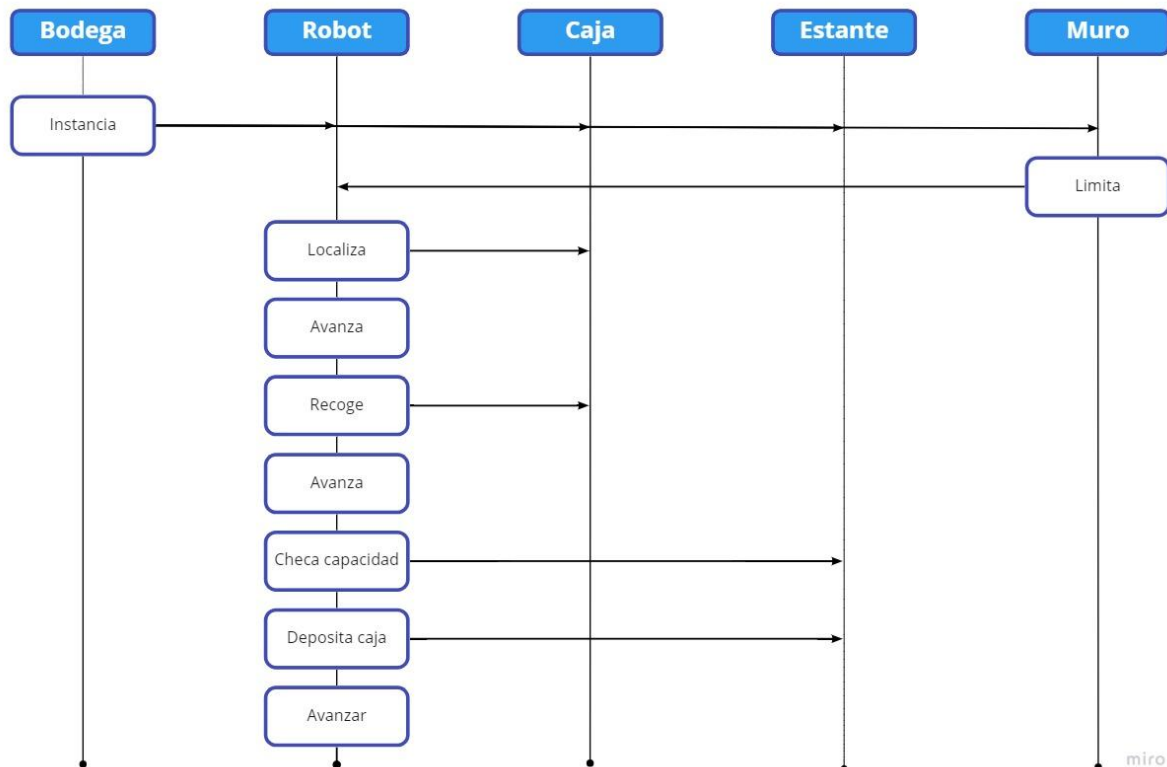


Diagramas de clase



miro

Protocolos de agentes



Recopilación de datos

Tiempo total y número de pasos por cada robot:

```
127.0.0.1 - - [28/Nov/2021 15:46:34] "GET /games/b9fa91b7-cf46-454e-93d2-91986f8a151f HTTP/1.1" 200 -
Tiempo para recoger cajas: 45
Robot con id 69 dio 34 pasos
Robot con id 70 dio 27 pasos
Robot con id 71 dio 28 pasos
Robot con id 72 dio 31 pasos
Robot con id 73 dio 30 pasos
```

Estrategia cooperativa

Descripción

Con el propósito de conseguir la cooperación y coordinación de todas las entidades del modelo, se tuvo que tomar en cuenta cada interacción posible entre ellos y las sucesión de eventos relevantes, tales como la recolección de cajas y su entrega a cada estante disponible. Los agentes relevantes para el funcionamiento esencial del modelo son el agente robot, el agente estante y el agente caja. El espacio dónde estos se desplazan se construye a partir de una función simple que define los bordes del almacén.

```
def initMaxtrix(n, m):
    matrix = [[1 for i in range(n)] for j in range(m)]
```

```

for i in range(n):
    matrix[0][i] = 0
    matrix[m - 1][i] = 0

for i in range(m):
    matrix[i][0] = 0
    matrix[i][n - 1] = 0

```

Siguiendo los diagramas de interacción antes mostrados, se creó el agente robot con el propósito de navegar a través de un cuarto de medidas NxM, y con la capacidad de sostener una caja y llevarla a un estante vacío. Su inteligencia primitiva le permite saber cuando está cargando una caja, la ubicación de un estante con capacidad para llevarla, y el estado del estante cuando llega a este, tanto así que el robot puede buscar algún otro estante vacío si al que llegó ya está lleno.

```

class Robot(Agent):
    NORMAL = 0
    LOADED = 1

    def __init__(self, model, pos):
        super().__init__(model.next_id(), model)
        self.pos = pos
        self.condition = self.NORMAL
        self.box = None
        self.rack = None
        self.roaming = False
        self.endX = 1
        self.endY = 1
        self.steps = 0

```

El robot es capaz de reconocer su estado actual y proseguir con la limpieza si hay cajas disponibles. También, en caso de tener una caja, trata de buscar estantes disponibles para empezar a moverse hacia ellos. Si ha escogido una caja, el robot empieza a moverse hacia su posición para recogerla y la quita del array de cajas disponibles para informarle a los demás agentes que la caja ya no está disponible.

```

def step(self):
    if(self.box == None):
        if(self.condition != self.LOADED and len(self.model.bboxes) > 0):
            i = random.randint(0, len(self.model.bboxes) - 1)
            self.box = self.model.bboxes.pop(i)

    else:
        if(self.pos == self.box.pos):
            self.condition = self.LOADED
            self.box = None
            agents = self.model.grid.get_cell_list_contents(self.pos)
            for agent in agents:
                if (type(agent) == Box):

                    self.model.grid.remove_agent(agent)
                    agent.pos = (0,0)
                    agents.remove(agent)
                    self.model.matrix[self.pos[1]][self.pos[0]] == 1

```

```

if (self.roaming == False and self.box != None):
    self.endX = self.box.pos[0]
    self.endY = self.box.pos[1]
    self.roaming = True

if (self.rack==None):
    if(len(self.model.racks) != 0):
        i = random.randint(0, len(self.model.racks) - 1)
        self.rack = self.model.racks[i]

if (self.condition == self.LOADED and self.rack != None):
    self.endX = self.rack.pos[0]
    self.endY = self.rack.pos[1]
    self.roaming = True

if (self.condition == self.LOADED):
    for neighbor in self.model.grid.neighbor_iter(self.pos, moore=True):
        if (type(neighbor) == Rack and self.condition == self.LOADED):
            if(neighbor == self.rack):
                if(neighbor.boxAmount < 5):
                    self.model.usedBoxes += 1
                    self.condition = self.NORMAL
                    neighbor.addBox()

                self.endX = self.pos[0]
                self.endY = self.pos[1]
                self.rack = None

```

Se usa el algoritmo A* de la librería 'pathfinding' para mover el Robot, de tal forma que se coloca en el espacio siguiente designado por su camino viable. En caso de no tener a dónde moverse se actualiza su estado para representar el hecho.

```

pathGrid = PathGrid(matrix=self.model.matrix)

start = pathGrid.node(self.pos[0], self.pos[1])
end = pathGrid.node(self.endX, self.endY)

finder = AStarFinder(diagonal_movement=DiagonalMovement.always)
path, runs = finder.find_path(start, end, pathGrid)

if (len(path) > 1):
    self.steps += 1
    next_move = path[1]
    self.model.grid.move_agent(self, next_move)

else:
    self.roaming = False
    pathGrid.cleanup()

return

```

Los estantes y cajas por su parte no son tan inteligentes, el primero de estos dos solo conoce el número de cajas que están alberga, mientras que el segundo, la caja, solo existe para ser movida. Aún así, su presencia es esencial para el funcionamiento de la bodega ya que estos deben ser detectados por los robots.

```

class Rack(Agent):
    def __init__(self, model, pos):
        super().__init__(model.next_id(), model)

```

```

        self.pos = pos
        self.boxAmount = 0

    def step(self):
        return

    def addBox(self):
        self.boxAmount += 1

class Box(Agent):
    def __init__(self, model, pos):
        super().__init__(model.next_id(), model)
        self.pos = pos

    def step(self):
        return

class Wall(Agent):
    def __init__(self, model, pos):
        super().__init__(model.next_id(), model)
        self.pos = pos

```

Mejoras

Pensando en maneras de mejorar los tiempos de recolección y la eficiencia de los robots para disminuir el número de pasos, se pueden tomar en cuenta un par de cosas. Primeramente, se podría cambiar la manera en la que cada robot selecciona la caja a recolectar, ya que en su modo actual se escogen de forma aleatoria, por lo que una buena propuesta sería encontrar la caja más cercana a la posición actual del robot. La segunda mejora a considerar es parecida a la primera, dado que la decisión sobre cual estante el robot escoge también es aleatoria. Así, se escogería el estante más cercano a la posición de la caja a recoger, y con ello se obtiene una ruta, en promedio, más corta a las que se toman actualmente.

Las mejoras no solo vienen de parte del agente robot, también pueden venir de la construcción de la bodega misma. Así, se podría contemplar el posicionamiento de los estantes para que no se ubiquen en posiciones al azar, y en su lugar que estén posicionados juntos, en un grupo, para evitar caminos a larga distancia de un extremo de la bodega al otro. Aplicando las mejoras antes mencionadas se disminuirían el número de pasos tanto para terminar la simulación como los pasos totales para cada robot, así mejorando la eficiencia de la bodega y reduciendo el costo de operación.

Conclusión

El modelo refleja muy bien el comportamiento real de una bodega automatizada y se asemeja bastante a la realidad. Siento que la incorporación de Mesa con las otras librerías que se nos fueron recomendadas para la realización de la actividad en realidad ayudaron mucho a enfocarnos en los aspectos que de verdad importaban, cómo la inteligencia de los agentes, y la conexión Python-Unity. El uso de modelos 3D especialmente fue innovador y me permitió observar el comportamiento de nuestro programa en un ambiente más realista.