



**Tecnológico
de Monterrey**

**Instituto Tecnológico y de Estudios
Superiores de Monterrey**

Campus Puebla

Modelación de sistemas multiagentes con gráficas computacionales

(Gpo 1)

TC2008B.1

Luciano García Bañuelos

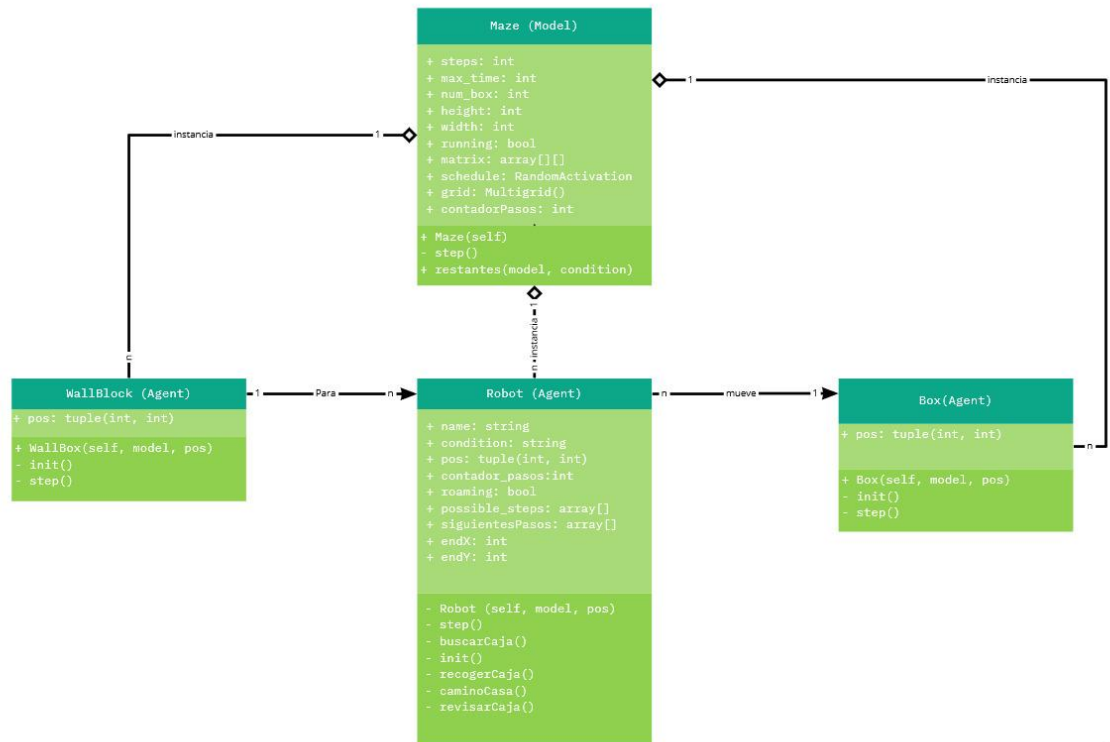
Actividad Integradora

Daniel Munive Meneses

A01734205

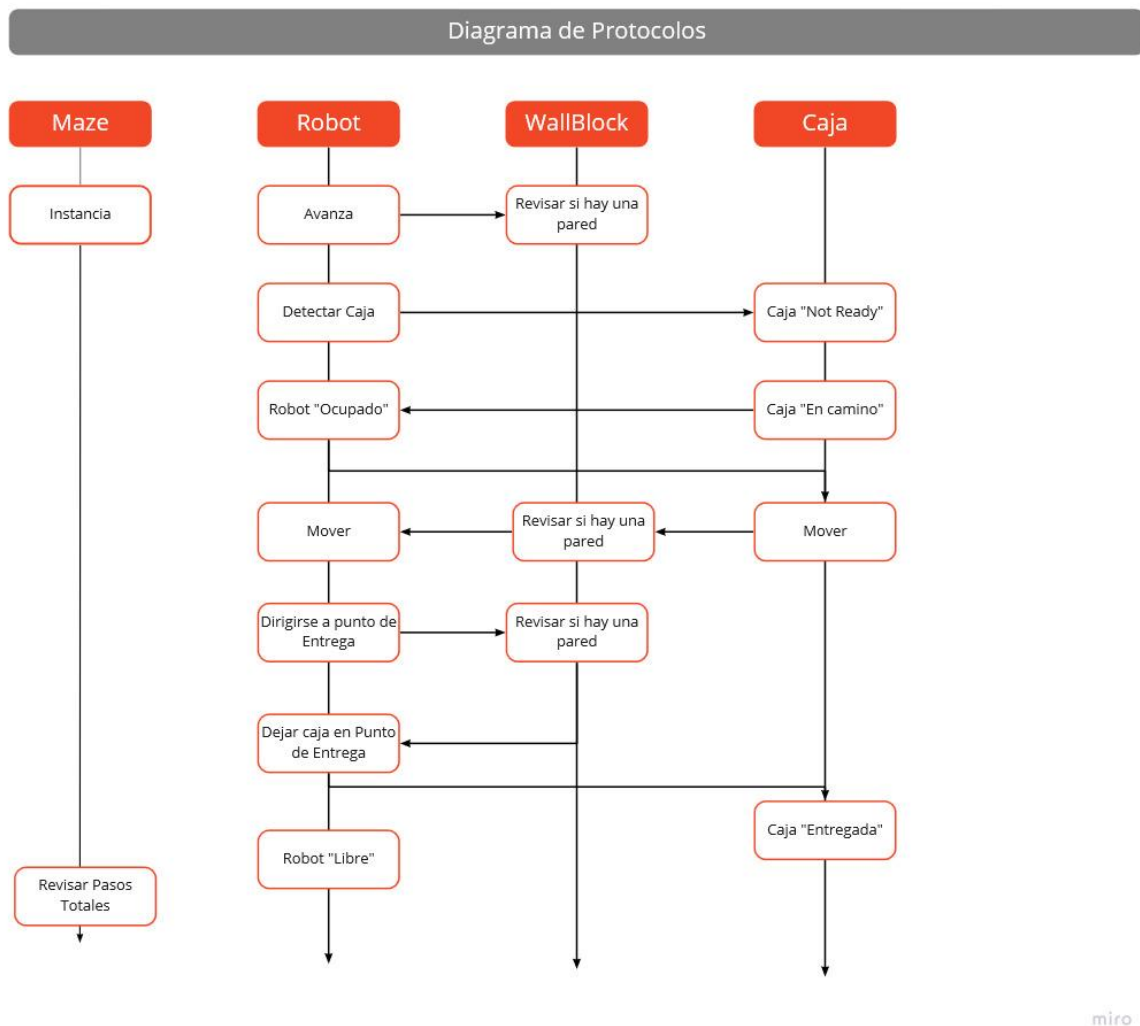
27 de noviembre 2021

Diagrama de Clase



miro

Protocolo de Agentes



Estrategia cooperativa

En esta actividad integradora se nos planteó el realizar la simulación de un software de 5 robots con ruedas omnidireccionales que puedan recoger cajas en celdas de cuadrícula adyacentes con sus manipuladores para posteriormente llevarlas a otra ubicación e incluso construir pilas de hasta cinco cajas.

Para la solución de este problema se realizó la simulación por medio de mesa y Unity, primeramente en Mesa se realizó la programación de los agentes sus interacciones y funcionalidad de la siguiente forma:

```
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

    return matrix
```

Primeramente se definió una función que creará una matriz del tamaño que se deseara para el modelo, y con esto inicializar los distintos agentes.

```
class Robot(Agent):
    def __init__(self, model, pos):
        super().__init__(model.next_id(), model)
        self.name = "robot"
        self.condition = "libre"
        self.pos = pos
        self.contador_pasos = 0
        self.roaming = False
        self.possible_steps = []
        self.siguietesPasos = []
        self.endX = 1
        self.endY = 1
```

Se crea el Agente Robot, con sus variables necesarias

```

108 def step(self):
109     zeros = []
110     self.grid = MultiGrid(15, 15, torus=False)
111     self.model.matrix
112     zeros = [(x, y) for y in range(self.model.height) for x in range
113             (self.model.width) if y in [0, self.model.height-1] or x in [0,
114             self.model.width - 1]]
115
116     next_moves = self.model.grid.get_neighborhood(self.pos, moore=False)
117     next_move = self.random.choice(next_moves)
118
119     if next_move in zeros:
120         next_move = self.random.choice(next_moves)
121     else:
122         if self.condition == "libre":
123             if self.buscarCaja():
124                 print("CAJA")
125                 self.recogerCaja()
126             else:
127                 self.contador_pasos += 1
128                 self.model.grid.move_agent(self, next_move)
129         elif self.condition == "Ocupado":
130             print("Transportando Caja")
131             self.revisarCaja()

```

Funcion step(), primeramente revisa las posiciones en las que hay paredes para ignorar estas posiciones, posteriormente los robot empiezan a moverse de manera random, hasta el momento en que encuentran una caja y al encontrarla, la recogen y se dirigen al punto de entrega.

```

def buscarCaja(self):
    for neighbor in self.model.grid.neighbor_iter(self.pos, moore=False):
        if neighbor.condition == "Not Ready":
            return True
    return False

def recogerCaja(self):
    self.condition = "Ocupado"
    for neighbor in self.model.grid.neighbor_iter(self.pos):
        if neighbor.condition == "Not Ready":
            self.caja = neighbor
            self.caja.condition = "En camino"
            break
    self.model.grid.move_agent(self.caja, self.pos)

def caminoCasa(self, posX, posY):
    pathGrid = PathGrid(matrix=self.model.matrix)

    if(self.roaming == False):
        self.endX = posX
        self.endY = posY
        self.roaming = True

    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)
    #print(path)
    if(len(path) > 1):
        next_move = path[1]
        self.contador_pasos += 1
        self.model.grid.move_agent(self, next_move)
        self.model.grid.move_agent(self.caja, next_move)

    else:
        self.roaming = False

    pathGrid.cleanup()

def revisarCaja(self):
    if (self.pos == (1,1) or self.pos == (2,1) or self.pos == (3,1) or self.pos == (4,1) or self.pos == (5,
1) or self.pos == (6,1) or self.pos == (7,1)):
        self.condition = "libre"
        self.caja.condition = "Entregada"
    else:

        self.posicionFinal = (1,random.randrange(1,7))
        self.caminoCasa(random.randrange(1,7),1)

```

Funciones utilizadas en el Step(), función buscarCaja() que revisa si hay cajas esperando ser entregadas en las posiciones vecinas del robot, recogerCaja(), junta por así decir a los agentes caja y robot y hace que los dos vayan cambiando de posición juntos, caminoCasa() utilizando pathfinding lleva al robot que esté cargando una caja

hacia los distintos posibles puntos de entrega de las cajas, `revisarCaja()` revisa las distintas posiciones de entrega posibles y elige una random para entregar las cajas.

```
class Box(Agent):
    def __init__(self, model, pos):
        super().__init__(model.next_id(), model)
        self.pos = pos
        if (self.pos == (1,1) or self.pos == (2,1) or self.pos == (3,1) or self.pos == (4,1) or self.pos == (5,1) or self.pos == (6,1) or self.pos == (7,1)):
            self.condition = "Entregado"
        else:
            self.condition = "Not Ready"

    def step(self):
        if (self.pos == (1,1) or self.pos == (2,1) or self.pos == (3,1) or self.pos == (4,1) or self.pos == (5,1) or self.pos == (6,1) or self.pos == (7,1)) :
            self.condition = "Entregado"
        else:
            self.condition = "Not Ready"
```

Se crea el agente caja y ya que esta aparecen de manera aleatoria, se revisa si aparecieron en algún punto de entrega desde el principio o no, para poder definir su condición, si ya están entregadas o no, así mismo esto se revisa en cada paso para evitar que las cajas se queden atrapadas o trabadas, fuera de los puntos de entrega

```
# Se define el agente de pared
class WallBlock(Agent):
    def __init__(self, model):
        super().__init__(self, model)
        self.condition = "Wall"

    def step(self):
        pass
```

Se crean los agente de las paredes como `WallBlock`

```
# Se define el modelo Maze con los atributos dentro del constructor
class Maze(Model):
    def __init__(self, height=15, width=15, boxN=15, max_time=500):
        super().__init__()
        self.steps = 0
        self.max_time = max_time
        self.num_box = boxN
        self.height = height
        self.width = width
        self.running = True
        self.matrix = initMaxtrix(width,height);
        self.schedule = RandomActivation(self)
        self.grid = MultiGrid(width, height, torus=False)
        self.contadorPasos = 0
```

Se crea el modelo de Maze, con sus respectivos atributos, estos tuvieron que ser definidos de manera estática, ya que al realizar el código de BackEnd y la conexión para Unity se encontró con la limitación de que no era posible usar Sliders para permitir que los usuarios definieran los datos, sin embargo el código está diseñado de forma que si se quisiera implementar esto en Mesa no supondría un problema.


```

muro = [(x, y) for y in range(height) for x in range(width) if y in [0,
height-1] or x in [0, width - 1]]

for pos in muro:
    wall = WallBlock(self)
    self.schedule.add(wall)
    self.grid.place_agent(wall, pos)

for i in range(5):
    def pos_robot(w, h):
        for _,x,y in self.grid.coord_iter():
            return (self.random.randrange(2,w), self.random.randrange(2,h))

    pos = pos_robot(self.grid.width, self.grid.height)

    while (not self.grid.is_cell_empty(pos)):
        pos = pos_robot(self.grid.width, self.grid.height)

    new_robot = Robot(self, pos)
    self.grid._place_agent(pos, new_robot)
    self.schedule.add(new_robot)

#Agregar Cajas
for i in range(self.num_box):
    def pos_box(w, h):
        return (self.random.randrange(w), self.random.randrange(h))

    pos = pos_box(self.grid.width, self.grid.height)

    while (not self.grid.is_cell_empty(pos)):
        pos = pos_box(self.grid.width, self.grid.height)

    new_box = Box(self, pos)
    self.grid._place_agent(pos, new_box)
    self.schedule.add(new_box)

```

En esta parte del código se inicializan las paredes alrededor de toda la matriz, se inicializan los 5 robot en posiciones random vacías, y se inicializan las cajas definidas (en este caso 15), también de manera random en espacios libres donde no haya ningún otro agente.

```

def step(self):
    self.contadorPasos += 1
    if (self.max_time <= 0 or self.restantes(self, "Entregado") == self.num_box):
        print("Final de simulacion")
        print("Número de movimientos realizados por todos los robots: ",
              self.contadorPasos*5)
        print("Tiempo total:", self.contadorPasos)
        self.running = False
    self.max_time -= 1
    self.schedule.step()

    @staticmethod
    def restantes(model, condition):
        count = 0
        for agent in model.schedule.agents:
            if agent.condition == condition:
                count += 1
        return count

```

En el step revisa si ya se entregaron todas las cajas, o si ya se llegó al tiempo límite terminado, esto para saber cuando terminar la simulación, así mismo al final se imprimen (esto en mesa), los pasos totales tomados por cada robot y el tiempo total tomado.

```

def agent_portrayal(agent):
    if(type(agent) == Robot):
        return {"Shape": "robot.png", "Layer": 1}
    elif (type(agent) == Box):
        return {"Shape": "box.png", "Layer": 0}
    elif (type(agent) == WallBlock):
        return {"Shape": "wall12.png", "Layer": 0}

    # Se definen las variables de grid, chart, y server
    grid = CanvasGrid(agent_portrayal, 15, 15, 450, 450)

    server = ModularServer(Maze, [grid], "Robot")

    server.port = 8522
    server.launch()

```

En esta parte de mesa se definen las imágenes determinadas de los agentes, el tamaño de los grid y el server para correr la simulación de forma local.

Posteriormente se realizó la Union del código para Unity

```
1 import flask
2 from flask.json import jsonify
3 import uuid
4 from ActInt import Maze
5
6 games = {}
7
8 app = flask.Flask(__name__)
9
10 @app.route("/games", methods=["POST"])
11 def create():
12     global games
13     id = str(uuid.uuid4())
14     games[id] = Maze()
15     return "ok", 201, {'Location': f"/games/{id}"}
16
17
18 @app.route("/games/<id>", methods=["GET"])
19 def queryState(id):
20     global model
21     model = games[id]
22     model.step()
23     Robot1 = model.schedule.agents[56]
24     Robot2 = model.schedule.agents[57]
25     Robot3 = model.schedule.agents[58]
26     Robot4 = model.schedule.agents[59]
27     Robot5 = model.schedule.agents[60]
28
29     Robot6 = model.schedule.agents[61]
30     Robot7 = model.schedule.agents[62]
31     Robot8 = model.schedule.agents[63]
32     Robot9 = model.schedule.agents[64]
33     Robot10 = model.schedule.agents[65]
34     Robot11 = model.schedule.agents[66]
35     Robot12 = model.schedule.agents[67]
36     Robot13 = model.schedule.agents[68]
37     Robot14 = model.schedule.agents[69]
38     Robot15 = model.schedule.agents[70]
39     Robot16 = model.schedule.agents[71]
40     Robot17 = model.schedule.agents[72]
41     Robot18 = model.schedule.agents[73]
42     Robot19 = model.schedule.agents[74]
43     Robot20 = model.schedule.agents[75]
```

```
Robot6 = model.schedule.agents[61]
Robot7 = model.schedule.agents[62]
Robot8 = model.schedule.agents[63]
Robot9 = model.schedule.agents[64]
Robot10 = model.schedule.agents[65]
Robot11 = model.schedule.agents[66]
Robot12 = model.schedule.agents[67]
Robot13 = model.schedule.agents[68]
Robot14 = model.schedule.agents[69]
Robot15 = model.schedule.agents[70]
Robot16 = model.schedule.agents[71]
Robot17 = model.schedule.agents[72]
Robot18 = model.schedule.agents[73]
Robot19 = model.schedule.agents[74]
Robot20 = model.schedule.agents[75]

return jsonify({ "Items": [{"x": Robot1.pos[0], "y": Robot1.pos[1]}, {"x": Robot2.pos[0], "y": Robot2.pos[1]},
{"x": Robot3.pos[0], "y": Robot3.pos[1]}, {"x": Robot4.pos[0], "y": Robot4.pos[1]}, {"x": Robot5.pos[0], "y": Robot5.pos[1]},
{"x": Robot6.pos[0], "y": Robot6.pos[1]}, {"x": Robot7.pos[0], "y": Robot7.pos[1]}, {"x": Robot8.pos[0], "y": Robot8.pos[1]},
{"x": Robot9.pos[0], "y": Robot9.pos[1]}, {"x": Robot10.pos[0], "y": Robot10.pos[1]}, {"x": Robot11.pos[0], "y": Robot11.pos[1]},
{"x": Robot12.pos[0], "y": Robot12.pos[1]}, {"x": Robot13.pos[0], "y": Robot13.pos[1]}, {"x": Robot14.pos[0], "y": Robot14.pos[1]},
{"x": Robot15.pos[0], "y": Robot15.pos[1]}, {"x": Robot16.pos[0], "y": Robot16.pos[1]}, {"x": Robot17.pos[0], "y": Robot17.pos[1]},
{"x": Robot18.pos[0], "y": Robot18.pos[1]}, {"x": Robot19.pos[0], "y": Robot19.pos[1]}, {"x": Robot20.pos[0], "y": Robot20.pos[1]} ] } })

app.run()
```

Donde se revise cada posición de los agentes por paso y se convierten los datos tipo json

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4 using UnityEngine.Networking;
5
6 public static class JsonHelper
7 {
8     public static T[] FromJson<T>(string json)
9     {
10         Wrapper<T> wrapper = JsonUtility.FromJson<Wrapper<T>>(json);
11         return wrapper.Items;
12     }
13
14     public static string ToJson<T>(T[] array)
15     {
16         Wrapper<T> wrapper = new Wrapper<T>();
17         wrapper.Items = array;
18         return JsonUtility.ToJson(wrapper);
19     }
20
21     public static string ToJson<T>(T[] array, bool prettyPrint)
22     {
23         Wrapper<T> wrapper = new Wrapper<T>();
24         wrapper.Items = array;
25         return JsonUtility.ToJson(wrapper, prettyPrint);
26     }
27
28     [System.Serializable]
29     private class Wrapper<T>
30     {
31         public T[] Items;
32     }
33 }
34
35 [System.Serializable]
36 class MyRobot
37 {
38     public int x;
39     public int y;
40
41     override public string ToString()
42     {
43         return "X: " + x + ", Y: " + y;
44     }
45 }
```

```

public class MoverModelo : MonoBehaviour
{
    string simulationURL = null;
    private float waitTime = 0.3f;
    private float timer = 0.0f;
    public GameObject[] Robot;

    // Start is called before the first frame update
    void Start()
    {
        StartCoroutine(ConnectToMesa());
    }

    IEnumerator ConnectToMesa()
    {
        WWWForm form = new WWWForm();

        using (UnityWebRequest www = UnityWebRequest.Post("http://localhost:5000/games", form))
        {
            yield return www.SendWebRequest();

            if (www.result != UnityWebRequest.Result.Success)
            {
                Debug.Log(www.error);
            }
            else
            {
                simulationURL = www.GetResponseHeader("Location");
                Debug.Log("Connected to simulation through Web API");
                Debug.Log(simulationURL);
            }
        }
    }
}

```

```

IEnumerator UpdatePositions()
{
    using (UnityWebRequest www = UnityWebRequest.Get(simulationURL))
    {
        if (simulationURL != null)
        {
            // Request and wait for the desired page.
            yield return www.SendWebRequest();

            //Debug.Log(www.downloadHandler.text);
            Debug.Log("Data has been processed");
            MyRobot[] robots = JsonHelper.FromJson<MyRobot>(www.downloadHandler.text);

            //Debug.Log(Robot[0].ToString());

            //Debug.Log(robots[4].ToString());

            Robot[0].transform.position = new Vector3(robots[0].x, 1, robots[0].y);
            Robot[1].transform.position = new Vector3(robots[1].x, 1, robots[1].y);
            Robot[2].transform.position = new Vector3(robots[2].x, 1, robots[2].y);
            Robot[3].transform.position = new Vector3(robots[3].x, 1, robots[3].y);
            Robot[4].transform.position = new Vector3(robots[4].x, 1, robots[4].y);

            Robot[5].transform.position = new Vector3(robots[5].x, 0, robots[5].y);
            Robot[6].transform.position = new Vector3(robots[6].x, 0, robots[6].y);
            Robot[7].transform.position = new Vector3(robots[7].x, 0, robots[7].y);
            Robot[8].transform.position = new Vector3(robots[8].x, 0, robots[8].y);
            Robot[9].transform.position = new Vector3(robots[9].x, 0, robots[9].y);
            Robot[10].transform.position = new Vector3(robots[10].x, 0, robots[10].y);
            Robot[11].transform.position = new Vector3(robots[11].x, 0, robots[11].y);
            Robot[12].transform.position = new Vector3(robots[12].x, 0, robots[12].y);
            Robot[13].transform.position = new Vector3(robots[13].x, 0, robots[13].y);
            Robot[14].transform.position = new Vector3(robots[14].x, 0, robots[14].y);
            Robot[15].transform.position = new Vector3(robots[15].x, 0, robots[15].y);
            Robot[16].transform.position = new Vector3(robots[16].x, 0, robots[16].y);
            Robot[17].transform.position = new Vector3(robots[17].x, 0, robots[17].y);
            Robot[18].transform.position = new Vector3(robots[18].x, 0, robots[18].y);
            Robot[19].transform.position = new Vector3(robots[19].x, 0, robots[19].y);
        }
    }
}

// Update is called once per frame
void Update()
{
    timer += Time.deltaTime;
    if (timer > waitTime)
    {
        StartCoroutine(UpdatePositions());
        timer = timer - waitTime;
    }
}
}

```

Tomando el ejemplo dado en clase y la librerías de unity se realizó la conexión de los datos obtenidos por la simulación de mesa y se les dio a los modelos 3d de Unity para con estos realizar la simulación en 3d.

Link de Video de Simulación corriendo en mesa: <https://youtu.be/ztJYE8Urrj0>

Resultados Obtenidos:

```
{"type": "get_step", "step": 359}
Final de simulacion
Número de movimientos realizados por todos los robots: 1795
Tiempo total: 359
Transportando Caja
{"type": "get_step", "step": 360}
```

Código implementado

```
from mesa import Agent, Model
from mesa.space import MultiGrid
from mesa.time import RandomActivation
from mesa.time import BaseScheduler
from mesa.visualization.modules import CanvasGrid
from mesa.visualization.ModularVisualization import ModularServer
from pathfinding.core.diagonal_movement import DiagonalMovement
from pathfinding.core.grid import Grid as PathGrid
from pathfinding.finder.a_star import AStarFinder
from mesa.visualization.UserParam import UserSettableParameter
from mesa.datacollection import DataCollector
from mesa.visualization.modules import ChartModule
from mesa.visualization.ModularVisualization import VisualizationElement
import random
from math import sqrt

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
    return matrix

class Robot(Agent):
    def __init__(self, model, pos):
```

```

    super().__init__(model.next_id(), model)
    self.name = "robot"
    self.condition = "libre"
    self.pos = pos
    self.contador_pasos = 0
    self.roaming = False
    self.possible_steps = []
    self.siguientesPasos = []
    self.endX = 1
    self.endY = 1

def buscarCaja(self):
    for neighbor in self.model.grid.neighbor_iter(self.pos, moore=False):
        if neighbor.condition == "Not Ready":
            return True
    return False

def recogerCaja(self):
    self.condition = "Ocupado"
    for neighbor in self.model.grid.neighbor_iter(self.pos):
        if neighbor.condition == "Not Ready":
            self.caja = neighbor
            self.caja.condition = "En camino"
            break
    self.model.grid.move_agent(self.caja, self.pos)

def caminoCasa(self, posX, posY):

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

    if(self.roaming == False):
        self.endX = posX
        self.endY = posY
        self.roaming = True

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

```



```

        self.model.grid.move_agent(self.caja, next_move)

    else:
        self.roaming = False

        pathGrid.cleanup()

    def revisarCaja(self):
        if (self.pos == (1,1) or self.pos == (2,1) or self.pos == (3,1) or
self.pos == (4,1) or self.pos == (5,1) or self.pos == (6,1) or self.pos ==
(7,1)):
            self.condition = "libre"
            self.caja.condition = "Entregada"
        else:

            self.posicionFinal = (1,random.randrange(1,7))
            self.caminoCasa(random.randrange(1,7),1)

            #for i in range(5):
            #    self.caminoCasa(i,1)

    def step(self):

        zeros = []
        self.grid = MultiGrid(15, 15, torus=False)
        self.model.matrix
        zeros = [(x, y) for y in range(self.model.height) for x in
range(self.model.width) if y in [0,self.model.height-1] or x in [0,
self.model.width - 1]]

        next_moves = self.model.grid.get_neighborhood(self.pos, moore=False)
        next_move = self.random.choice(next_moves)

        if next_move in zeros:
            next_move = self.random.choice(next_moves)
        else:
            if self.condition == "libre":
                if self.buscarCaja():
                    print("CAJA")
                    self.recogerCaja()
                else:
                    self.contador_pasos += 1
                    self.model.grid.move_agent(self, next_move)

```

```

        elif self.condition == "Ocupado":
            print("Transportando Caja")
            self.revisarCaja()

class Box(Agent):
    def __init__(self, model, pos):
        super().__init__(model.next_id(), model)
        self.pos = pos
        if (self.pos == (1,1) or self.pos == (2,1) or self.pos == (3,1) or
self.pos == (4,1) or self.pos == (5,1) or self.pos == (6,1) or self.pos ==
(7,1)):
            self.condition = "Entregado"
        else:
            self.condition = "Not Ready"

    def step(self):

        if (self.pos == (1,1) or self.pos == (2,1) or self.pos == (3,1) or
self.pos == (4,1) or self.pos == (5,1) or self.pos == (6,1) or self.pos ==
(7,1)) :
            self.condition = "Entregado"
        else:
            self.condition = "Not Ready"

# Se define el agente de pared
class WallBlock(Agent):
    def __init__(self, model):
        super().__init__(self, model)
        self.condition = "Wall"

    def step(self):
        pass

# Se define el modelo Maze con los atributos dentro del constructor
class Maze(Model):
    def __init__(self, height=15, width=15, boxN=15, max_time=500):
        super().__init__()
        self.steps = 0
        self.max_time = max_time
        self.num_box = boxN
        self.height = height
        self.width = width

```

```

self.running = True
self.matrix = initMaxtrix(width,height);
self.schedule = RandomActivation(self)
self.grid = MultiGrid(width, height, torus=False)
self.contadorPasos = 0

    muro = [(x, y) for y in range(height) for x in range(width) if y in
[0, height-1] or x in [0, width - 1]]

    for pos in muro:
        wall = WallBlock(self)
        self.schedule.add(wall)
        self.grid.place_agent(wall, pos)

    for i in range(5):

        def pos_robot(w, h):
            for _,x,y in self.grid.coord_iter():
                return (self.random.randrange(2,w),
self.random.randrange(2,h))

        pos = pos_robot(self.grid.width, self.grid.height)

        while (not self.grid.is_cell_empty(pos)):
            pos = pos_robot(self.grid.width, self.grid.height)

        new_robot = Robot(self, pos)
        self.grid._place_agent(pos, new_robot)
        self.schedule.add(new_robot)

#Agregar Cajas
for i in range(self.num_box):

    def pos_box(w, h):
        return (self.random.randrange(w), self.random.randrange(h))

    pos = pos_box(self.grid.width, self.grid.height)

    while (not self.grid.is_cell_empty(pos)):
        pos = pos_box(self.grid.width, self.grid.height)

    new_box = Box(self, pos)
    self.grid._place_agent(pos, new_box)
    self.schedule.add(new_box)

```

```

def step(self):
    self.contadorPasos += 1
    if (self.max_time <= 0 or self.restantes(self, "Entregado") ==
self.num_box):
        print("Final de simulacion")
        print("Número de movimientos realizados por todos los robots:
",self.contadorPasos*5)
        print("Tiempo total:", self.contadorPasos)
        self.running = False
        self.max_time -= 1
        self.schedule.step()

    @staticmethod
    def restantes(model, condition):
        count = 0
        for agent in model.schedule.agents:
            if agent.condition == condition:
                count += 1
        return count

def agent_portrayal(agent):
    if(type(agent) == Robot):
        return {"Shape": "robot.png", "Layer": 1}
    elif (type(agent) == Box):
        return {"Shape": "box.png", "Layer": 0}
    elif (type(agent) == WallBlock):
        return {"Shape": "wall12.png", "Layer": 0}

    # Se definen las variables de grid, chart, y server
    grid = CanvasGrid(agent_portrayal, 15, 15, 450, 450)

    server = ModularServer(Maze, [grid], "Robot")

    server.port = 8522
    server.launch()

```

Unity

(Para poder correr el código de manera correcta es necesario tener el paquete de Probuilder dentro de su proyecto de unity)

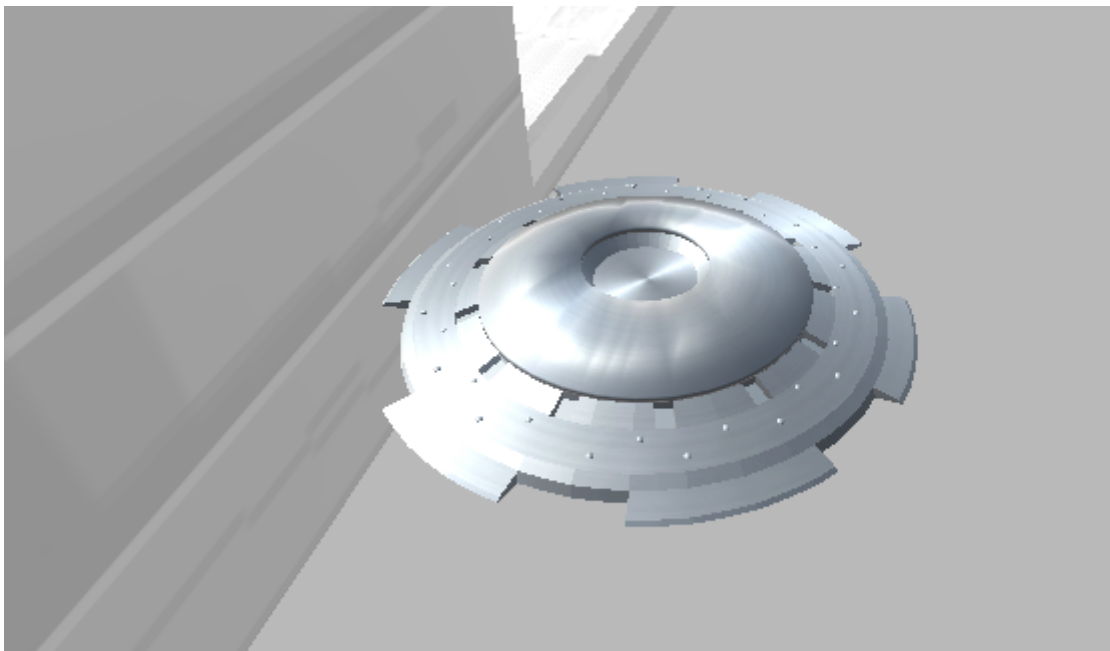
Link de Video a Simulación 3D corriendo:

<https://youtu.be/FPbf5GNWFMk>

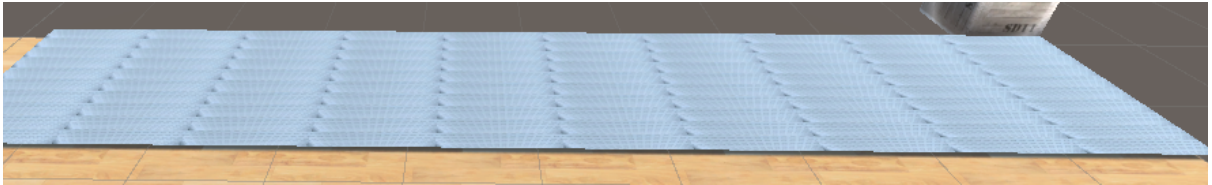
Modelo de caja:



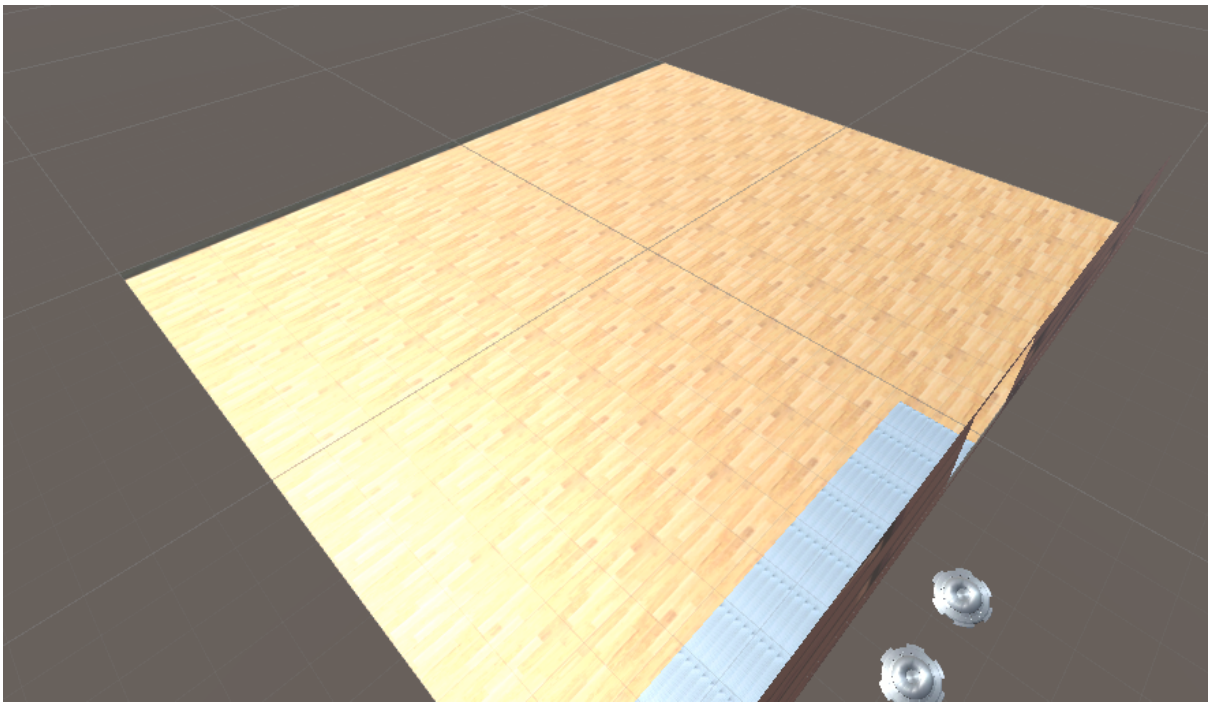
Modelo de Robot:



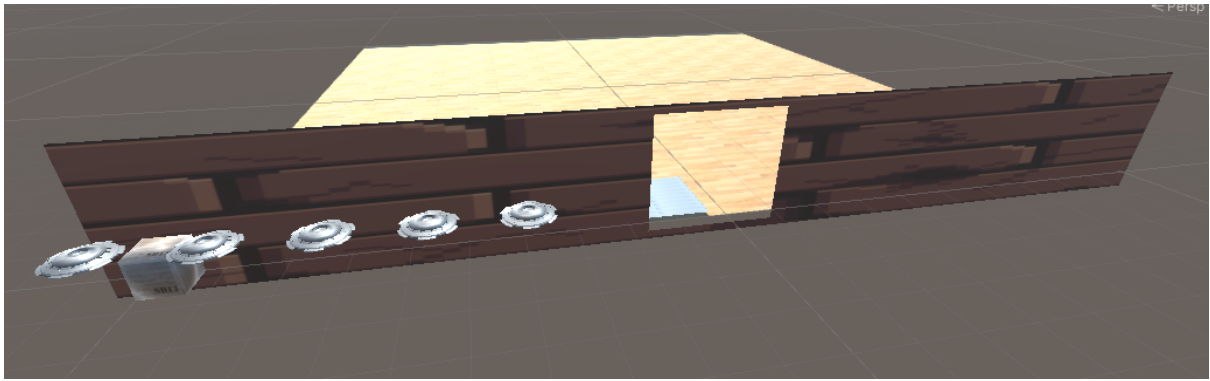
Modelo de punto de entrega:



Modelo de suelo:



Modelo de paredes:



Link de GitHub:

<https://github.com/DanielMuni/Actividad-Integradora-Paqueteria>

Implicaciones Éticas

Durante el desarrollo de esta actividad integradora me guíe en base al Código de Ética de la Ingeniería de Software y al del Tecnológico de Monterrey, además del código realizado cumple con los ocho principios: sociedad, cliente, producto, juicio, administración, profesión, colegas, y personal, así mismo me guíe por lo códigos éticos internacionales definidos por organizaciones de prestigio internacional en el campo de la informática, como el Institute of Electrical and Electronics Engineers (IEEE).