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

Juan Pablo Armendariz Salas A01734010

29 de noviembre 2021

1 Introducción

Fue asignado un grupo de cinco robots nuevos y un almacén lleno de cajas. Al presentar un gran desorden, se asignó a cada uno de los robots organizar de alguna manera las

cajas. Cada robot puede moverse en cuatro dimensiones y pueden recoger cajas en celdas de cuadrículas específicas. Los robots tienen la habilidad de llevar una caja a otra ubicación y construir pilas de 5 cajas. Estos están equipados con sensores que permiten detectar dónde se encuentra un agente en un plano. Por esto, los robots pueden distinguir si un campo esta libre, es una pared, contiene una pila, o tiene una caja. Además, cada robot presenta un sensor de presión que permite indicar si tiene o no una caja, marcando su estado actual.

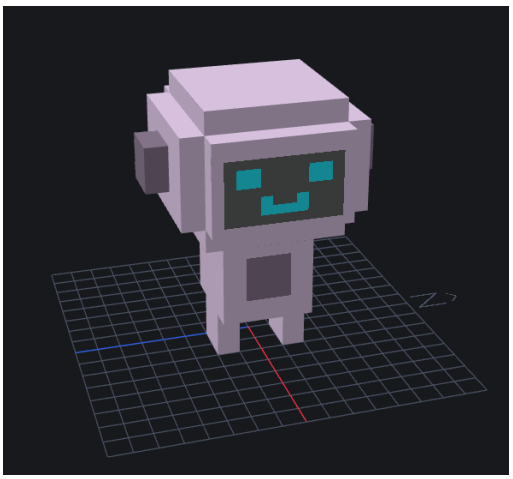
Al no presentar un presupuesto suficiente, no se pudo adquirir un software de gestión de agentes múltiples, por lo que la tarea principal del trabajo es generar una simulación que enseñe a los robots a ordenar cajas en pilas de cinco en un almacén. La simulación inicializa las posiciones iniciales de K cajas a nivel de piso, posiciona los agentes en celdas vacías de manera aleatoria, y se ejecuta en un tiempo específico.

El objetivo del trabajo fue realizar una simulación que sea capaz de recopilar información sobre el tiempo necesario hasta que todas las cajas esten apiladas y el número de movimientos realizados por cada robot. Además, el modelo despliega dicha simulación en tres dimensiones en un diseño que tiene modelos con materiales y texturas y una animación correcta. Todo esto para demostrar el funcionamiento de sistemas computacionales y generar modelos.

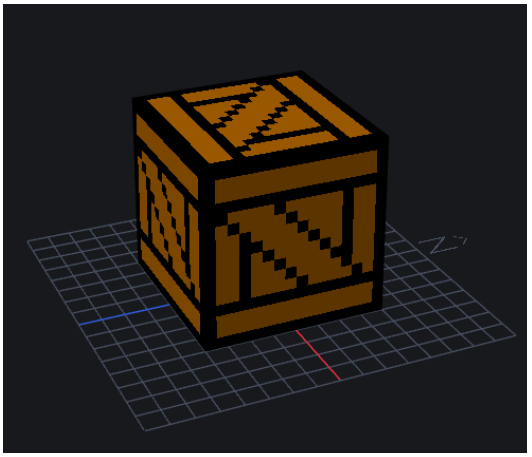
2 Implementación

2.1 Agentes y modelo

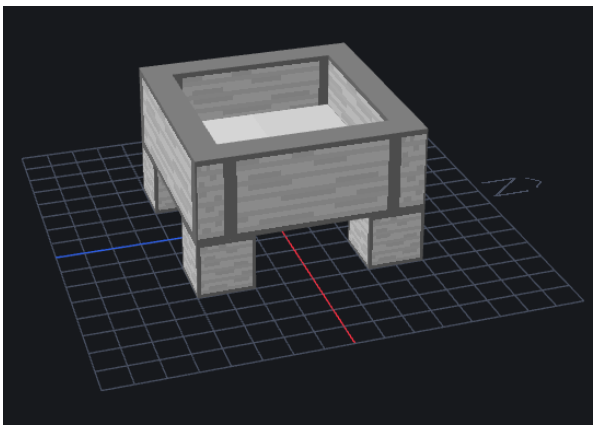
- **Robot (agente):** Agente móvil que busca un agente Box dentro de un espacio y las mueve hacia un agente pila. Este agente presenta los atributos de posición, modelo, y la pila. Los robots hacen uso del algoritmo A* para moverse dentro del plano, y se detienen una vez que hayan recolectado cinco cajas en una pila o si ya no hay cajas disponibles.



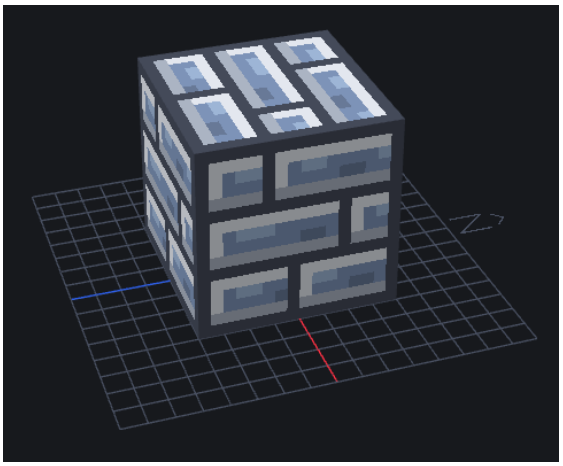
- **Box (agente):** Agente estático que tiene la función de una caja, los cuales están posicionados de manera aleatoria dentro del plano. Este agente tiene como atributos el modelo y su posición. Su objetivo es interactuar con los agentes Robot una vez que estos los recogen y son asignados a los mismos. Dentro de la matriz se representa como un número 2.



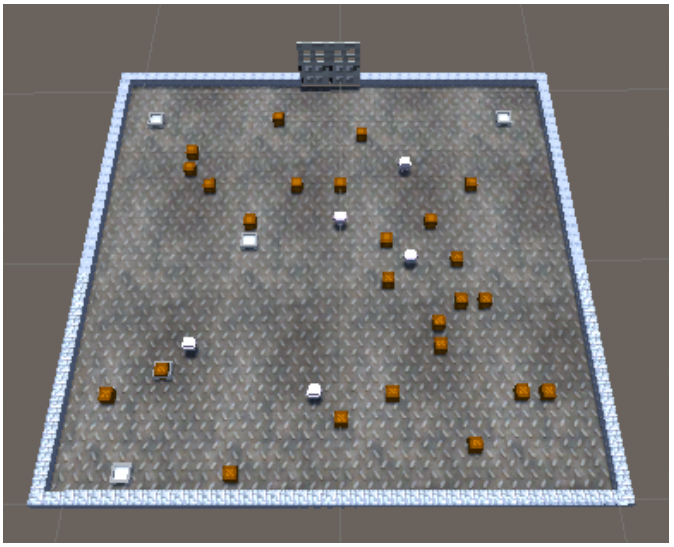
- **Pile (agente):** Agente estático que tiene la función de simular una pila o espacio de recolección, los cuales se encuentran distribuidos de manera aleatoria en el plano. Este agente tiene los atributos del modelo y su posición. Su meta es la de formar parte de un robot y guardar las cajas recolectadas por el mismo. Dentro de la matriz se representa como un número 3.



- **MetalBlock (agente):** Agente estático que simula los límites del plano. Este agente permite que todos los agentes Robot no sobrepasen los límites de la matriz. Dentro de la matriz se representa como un 1.



- **WareHouse (modelo):** Representa el escenario general en donde se realizan todas las interacciones entre los agentes. El modelo se encarga de inicializar las posiciones de los agentes Robot, Box, Pile, y MetalBlock, al igual que los espacios vacíos dentro de la matriz. Presenta dos funciones, una para asignar un robot a una caja, y otro para remover la caja de la matriz. El modelo también se encarga de culminar la simulación una vez que todas las cajas están en una pila.



2.2 Estrategia

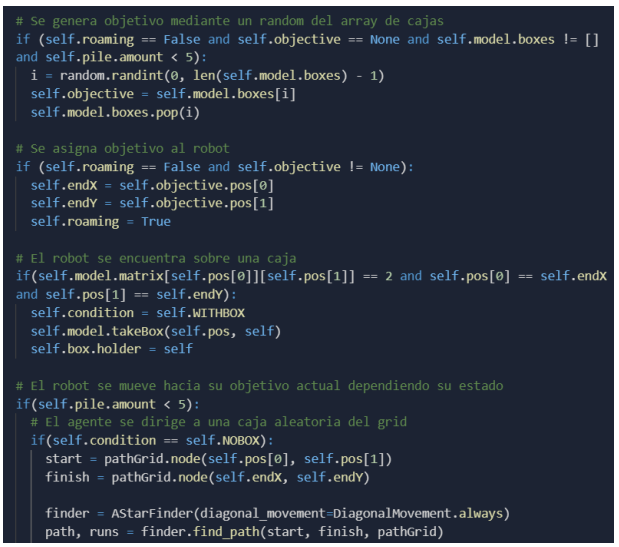
Para realizar la actividad integradora se hizo un enfoque en dos partes principales, la simulación del sistema multiagentes en Mesa con las herramientas de Python y las gráficas computacionales mediante los modelados y animaciones de Unity.

Primeramente se generó un programa en Python que simulara el sistema multiagentes, basado en los trabajos previamente hechos y en la situación problema.

```python
def createMatrix(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
```

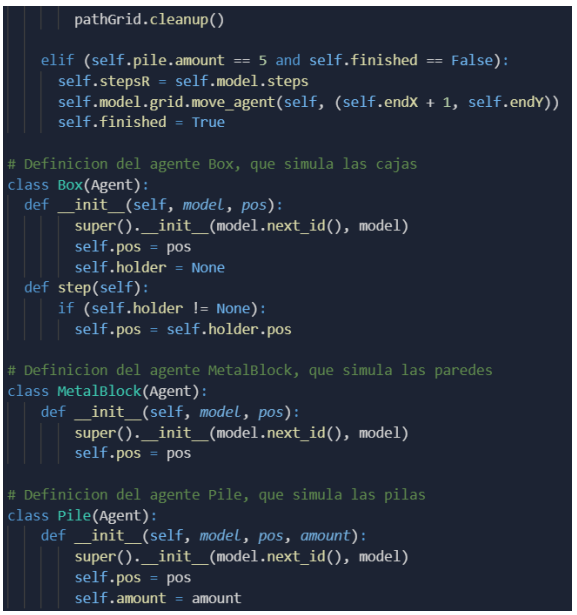
Se generó una función createMatrix para generar una matriz de cualquier tamaño. Esta recibe el valor de ancho y alto, y llena todos los espacios con 0 y 1 en base a lo requerido.

```python
class Robot(Agent):
    NOBOX = 0
    WITHBOX = 1

    def __init__(self, model, pos, pile):
        super().__init__(model.next_id(), model)
        self.condition = self.NOBOX
        self.pos = pos
        self.roaming = False
        self.objective = None
        self.endX = 1
        self.endY = 1
        self.pile = pile
        self.box = None
        self.stepsR = 0
        self.finished = False

    def step(self):
        # Se asgina una pila al robot
        if(self.pile == None):
            i = random.randint(0, len(self.model.piles) - 1)
            self.pile = self.model.piles[i]
            self.model.piles.remove(self.pile)

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



```python
        # Se genera objetivo mediante un random del array de cajas
        if (self.roaming == False and self.objective == None and self.model.boxes != []
        and self.pile.amount < 5):
            i = random.randint(0, len(self.model.boxes) - 1)
            self.objective = self.model.boxes[i]
            self.model.boxes.pop(i)

        # Se asigna objetivo al robot
        if (self.roaming == False and self.objective != None):
            self.endX = self.objective.pos[0]
            self.endY = self.objective.pos[1]
            self.roaming = True

        # El robot se encuentra sobre una caja
        if(self.model.matrix[self.pos[0]][self.pos[1]] == 2 and self.pos[0] == self.endX
        and self.pos[1] == self.endY):
            self.condition = self.WITHBOX
            self.model.takeBox(self.pos, self)
            self.box.holder = self

        # El robot se mueve hacia su objetivo actual dependiendo su estado
        if(self.pile.amount < 5):
            # El agente se dirige a una caja aleatoria del grid
            if(self.condition == self.NOBOX):
                start = pathGrid.node(self.pos[0], self.pos[1])
                finish = pathGrid.node(self.endX, self.endY)

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

```python
                if(len(path) > 1):
                    newMove = path[1]
                    self.model.grid.move_agent(self, newMove)
                else:
                    self.roaming = False
                    self.objective = None

                pathGrid.cleanup()

            # El agente se dirige hacia su pila asignada
            elif(self.condition == self.WITHBOX):

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

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

                if(len(path) > 1):
                    newMove = path[1]
                    self.model.grid.move_agent(self, newMove)
                else:
                    self.roaming = False
                    self.objective = None
                    self.condition = self.NOBOX
                    self.pile.amount += 1
                    self.model.boxesInPile += 1
                    self.box.holder = self.pile
```



```python
                pathGrid.cleanup()

            elif (self.pile.amount == 5 and self.finished == False):
                self.stepsR = self.model.steps
                self.model.grid.move_agent(self, (self.endX + 1, self.endY))
                self.finished = True

    # Definicion del agente Box, que simula las cajas
    class Box(Agent):
        def __init__(self, model, pos):
            super().__init__(model.next_id(), model)
            self.pos = pos
            self.holder = None
        def step(self):
            if (self.holder != None):
                self.pos = self.holder.pos

    # Definicion del agente MetalBlock, que simula las paredes
    class MetalBlock(Agent):
        def __init__(self, model, pos):
            super().__init__(model.next_id(), model)
            self.pos = pos

    # Definicion del agente Pile, que simula las pilas
    class Pile(Agent):
        def __init__(self, model, pos, amount):
            super().__init__(model.next_id(), model)
            self.pos = pos
            self.amount = amount
```

Después se generaron los agentes y el modelo. Los agentes que se generaron fueron Robot, Box, Pile, y MetalBlock. El agente Robot es se mueve dentro del plano mediante el algoritmo A* de la librería Pathfinding. El agente toma de manera aleatoria una de las cajas de la matriz y la convierte en su objetivo. Una vez que llegue, a la caja se le asigna ese robot y fija su objetivo hacia una pila. La pila se genera de manera aleatoria en el modelo y es asignada al robot una vez que se inicializa. Cuando el objetivo de la caja sea la pila, su estado cambia para demostrar que tiene una caja, y vuelve a cambiar cuando deja la caja en la pila. La caja va actualizando su posición en base a la posición del robot y convierte su posición al de la pila una vez que el robot la deposita. Este proceso se

repite hasta que el robot haya ordenado cinco cajas en la pila o si ya no quedan cajas en el plano.

```python
# Definicion del modelo WareHouse
class WareHouse(Model):
  def __init__(self):
      super().__init__()

      self.height = 20
      self.width = 20
      self.boxA = 25
      self.totalBoxes = 25
      self.robotN = 5
      self.pileN = 5
      self.boxes = []
      self.piles = []
      self.robots = []
      self.boxesInPile = 0
      self.over = False
      self.steps = 0

      # Se genera la matriz y el grid
      self.matrix = createMatrix(self.width, self.height)
      self.schedule = RandomActivation(self)
      self.grid = MultiGrid(self.width, self.height, torus=False)

      # Se agregan las cajas al grid en base a boxA
      while(self.boxA > 0):
        box = Box(self, (self.random.randint(1, self.width-2),
        self.random.randint(1, self.height-2)))
        if (self.matrix[box.pos[0]][box.pos[1]] != 2 and self.matrix[box.pos
        [0]][box.pos[1]] != 3):
          self.grid.place_agent(box, box.pos)
          self.schedule.add(box)
          self.matrix[box.pos[0]][box.pos[1]] = 2
          self.boxes.append(box)
        else:
          self.boxA += 1
        self.boxA -= 1

      # Se agregan los robots al grid en base a robotN
      while (self.robotN > 0):
        robot = Robot(self, (self.random.randint(1, self.width-2),
        self.random.randint(1, self.height-2)), None)
        if (self.matrix[robot.pos[0]][robot.pos[1]] != 2 and self.matrix
        [robot.pos[0]][robot.pos[1]] != 3):
          self.grid.place_agent(robot, robot.pos)
          self.schedule.add(robot)
          self.robots.append(robot)
        else:
          self.robotN += 1
        self.robotN -= 1

      # Se agregan las pilas al grid en base a pileN
      while (self.pileN > 0):
        pile = Pile(self, (self.random.randint(1, self.width-2),
        self.random.randint(1, self.height-2)), 0)
        if (self.matrix[pile.pos[0]][pile.pos[1]] != 2 and self.matrix[pile.pos
        [0]][pile.pos[1]] != 3):
          self.grid.place_agent(pile, pile.pos)
          self.schedule.add(pile)
          self.matrix[pile.pos[0]][pile.pos[1]] = 3
          self.piles.append(pile)
        else:
          self.pileN += 1
        self.pileN -= 1
```

```python
def step(self):
    self.schedule.step()
    self.steps += 1

    if(self.over == True):
      for i in range (len(self.robots)):
        print("Total de pasos dados por robot: ", self.robots[i].stepsR)
      print("Tiempo total: ", self.steps)
      self.running = False

    if(self.boxesInPile > self.totalBoxes - 1):
      self.over = True

# Funcion que asigna al agente caja un robot
# Complejidad: O(1)
def assignRobot(self, box, robot):
  robot.box = box
  box.robot = robot

# Funcion que cambia las caracteristicas de la celda en base a la posicion actual del robot
# Complejidad: O(n)
def takeBox(self, pos, robot):
    agents = self.grid.get_cell_list_contents(pos)
    for agent in agents:
      if(type(agent) == Box):
        self.grid.remove_agent(agent)
        agents.remove(agent)
        self.matrix[pos[1]][pos[0]] == 1
        self.assignRobot(agent, robot)
```

El modelo fue el encargado de instanciar a cada uno de los agentes dentro del Grid y de la matriz. Además presenta dos funciones assignBox y takeBox, que se encargan de asignar una robot a una caja y de quitar la caja de la matriz. El modelo también genera los arrays de cada agente para que puedan ser utilizados en cualquier parte del programa.

Dentro de la actividad integradora se generaron dos códigos de simulación, uno para que pueda correr desde Unity, y otro que pueda correr mediante Mesa solamente. Este

segundo programa mencionado presenta un bloque de código más que permite observar la simulación desde Python y con sus propios archivos PNG.

```python
#Funcion de mesa que define las apariencias de los agentes
#Complejidad: O(1)
def agent_portrayal(agent):

  if(type(agent) == Robot):
    if(agent.condition == agent.NOBOX):
      return {"Shape": "robot.png", "Layer": 0}
    elif(agent.condition == agent.WITHBOX):
      return {"Shape": "robotBox.png", "Layer": 0}
  elif (type(agent) == Box):
    return {"Shape": "box.png", "Layer": 0}
  elif (type(agent) == MetalBlock):
    return {"Shape": "metalBlock.png", "Layer": 0}
  elif (type(agent) == Pile):
    return {"Shape": "rect", "w": 1, "h": 1, "Filled": "true", "Color": "Grey", "Layer": 0}

# Se definen las variables de grid y server
grid = CanvasGrid(agent_portrayal, 20, 20, 450, 450)

server = ModularServer(WareHouse, [grid], "Box collector", {})

server.port = 8522
server.launch()
```

Video de simulación en Mesa: https://www.youtube.com/watch?v=YUQpZ3fE_o4

Ahora, para la parte de las animaciones y el modelo en Unity utilizó el código generado con Python y otros dos programas que fueron de utilidad para poder unificar la parte de Mesa con Unity.

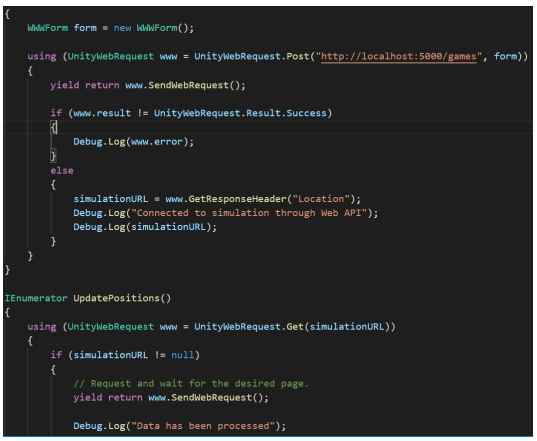
backend.py > queryState

```python
import flask
from flask.json import jsonify
import uuid
from paquetes import WareHouse

games = {}

app = flask.Flask(__name__)

@app.route("/games", methods=["POST"])
def create():
    global games
    id = str(uuid.uuid4())
    games[id] = WareHouse()
    return "ok", 201, {'Location': f"/games/{id}"}


@app.route("/games/<id>", methods=["GET"])
def queryState(id):
    global model
    model = games[id]
    model.step()

    robot1 = model.schedule.agents[25]
    robot2 = model.schedule.agents[26]
    robot3 = model.schedule.agents[27]
    robot4 = model.schedule.agents[28]
    robot5 = model.schedule.agents[29]

    box1 = model.schedule.agents[0]
    box2 = model.schedule.agents[1]
    box3 = model.schedule.agents[2]
    box4 = model.schedule.agents[3]
    box5 = model.schedule.agents[4]
    box6 = model.schedule.agents[5]
    box7 = model.schedule.agents[6]
    box8 = model.schedule.agents[7]
    box9 = model.schedule.agents[8]
    box10 = model.schedule.agents[9]
    box11 = model.schedule.agents[10]
    box12 = model.schedule.agents[11]
    box13 = model.schedule.agents[12]
    box14 = model.schedule.agents[13]
    box15 = model.schedule.agents[14]
    box16 = model.schedule.agents[15]
    box17 = model.schedule.agents[16]
    box18 = model.schedule.agents[17]
    box19 = model.schedule.agents[18]
    box20 = model.schedule.agents[19]
    box21 = model.schedule.agents[20]
    box22 = model.schedule.agents[21]
    box23 = model.schedule.agents[22]
    box24 = model.schedule.agents[23]
    box25 = model.schedule.agents[24]

    pile1 = model.schedule.agents[30]
    pile2 = model.schedule.agents[31]
    pile3 = model.schedule.agents[32]
    pile4 = model.schedule.agents[33]
```

```
pile5 = model.schedule.agents[34]

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": box1.pos[0], "y": box1.pos[1]},
    {"x": box2.pos[0], "y": box2.pos[1]},
    {"x": box3.pos[0], "y": box3.pos[1]},
    {"x": box4.pos[0], "y": box4.pos[1]},
    {"x": box5.pos[0], "y": box5.pos[1]},
    {"x": box6.pos[0], "y": box6.pos[1]},
    {"x": box7.pos[0], "y": box7.pos[1]},
    {"x": box8.pos[0], "y": box8.pos[1]},
    {"x": box9.pos[0], "y": box9.pos[1]},
    {"x": box10.pos[0], "y": box10.pos[1]},
    {"x": box11.pos[0], "y": box11.pos[1]},
    {"x": box12.pos[0], "y": box12.pos[1]},
    {"x": box13.pos[0], "y": box13.pos[1]},
    {"x": box14.pos[0], "y": box14.pos[1]},
    {"x": box15.pos[0], "y": box15.pos[1]},
    {"x": box16.pos[0], "y": box16.pos[1]},
    {"x": box17.pos[0], "y": box17.pos[1]},
    {"x": box18.pos[0], "y": box18.pos[1]},
    {"x": box19.pos[0], "y": box19.pos[1]},
    {"x": box20.pos[0], "y": box20.pos[1]},
    {"x": box21.pos[0], "y": box21.pos[1]},
    {"x": box22.pos[0], "y": box22.pos[1]},
    {"x": box22.pos[0], "y": box22.pos[1]},
    {"x": box23.pos[0], "y": box23.pos[1]},
    {"x": box24.pos[0], "y": box24.pos[1]},
    {"x": box25.pos[0], "y": box25.pos[1]},
    {"x": pile1.pos[0], "y": pile1.pos[1]},
    {"x": pile2.pos[0], "y": pile2.pos[1]},
    {"x": pile3.pos[0], "y": pile3.pos[1]},
    {"x": pile4.pos[0], "y": pile4.pos[1]},
    {"x": pile5.pos[0], "y": pile5.pos[1]}
    ]})

app.run()
```

El primer código fue el de backend.py que permite tomar los datos generados por Mesa y los guarda en un espacio en formato JSON. Aquí se toman los agentes del array que genera la parte de mesa y se inserta dentro de los ítems tomando en cuenta su posición en X y Y.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;

public static class JsonHelper
{
    public static T[] FromJson<T>(string json)
    {
        Wrapper<T> wrapper = JsonUtility.FromJson<Wrapper<T>>(json);
        return wrapper.Items;
    }

    public static string ToJson<T>(T[] array)
    {
        Wrapper<T> wrapper = new Wrapper<T>();
        wrapper.Items = array;
        return JsonUtility.ToJson(wrapper);
    }

    public static string ToJson<T>(T[] array, bool prettyPrint)
    {
        Wrapper<T> wrapper = new Wrapper<T>();
        wrapper.Items = array;
        return JsonUtility.ToJson(wrapper, prettyPrint);
    }

    [System.Serializable]
    private class Wrapper<T>
    {
        public T[] Items;
    }
}

[System.Serializable]
class MyObject
{
    public int x;
    public int y;

    override public string ToString()
    {
        return "X: " + x + ", Y: " + y;
    }
}

public class MoveObject : MonoBehaviour
{
    string simulationURL = null;
    private float waitTime = 0.2f;
    private float timer = 0.0f;
    public GameObject[] myObjects;

    // 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("Data has been processed");
```

```
            Debug.Log("Data has been processed");
            MyObject[] agente = JsonHelper.FromJson<MyObject>(www.downloadHandler.text);

            for(int i = 0; i < 35; i++){
                myObjects[i].transform.position = new Vector3(agente[i].x, 0, agente[i].y);
            }
        }
    }
}

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

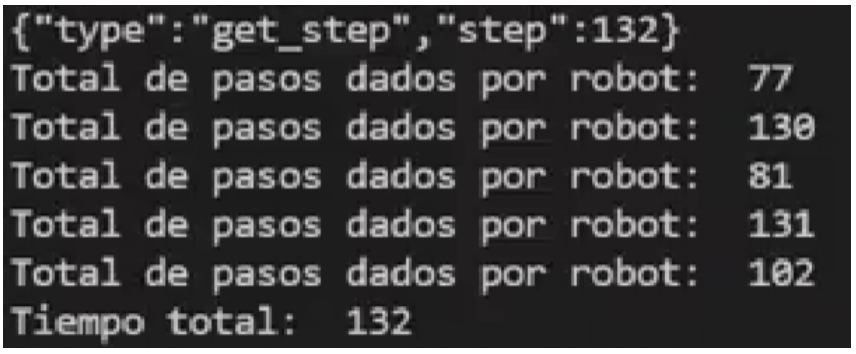
El segundo código es un script de C# dentro de Unity denominado MoveObject.cs, que toma los valores que distribuye la parte del backend y los guarda en un array. Una vez con cada uno de los agentes, se les añade de manera ordenada dentro de un array de GameObjects para que puedan ser asignados dentro de la simulación.

Una vez unificado y corriendo todos los códigos generados se pude dar play a la simulación dentro de Unity para observar como los objetos 3D van realizando las interacciones de un sistema multiagentes.

Video de simulación en Unity: https://www.youtube.com/watch?v=-D-tMKvrxr4

2.3 Información recopilada

Tomando como referencia la simulación anterior, se pudieron recopilar la información sobre el tiempo necesario hasta que todas las cajas esten apiladas y el número de movimientos realizados por cada robot. Los resultados se despliegan en la consola tal como se ve en el video y se muestran a continuación.



```
{"type":"get_step","step":132}
Total de pasos dados por robot:  77
Total de pasos dados por robot:  130
Total de pasos dados por robot:  81
Total de pasos dados por robot:  131
Total de pasos dados por robot:  102
Tiempo total:  132
```

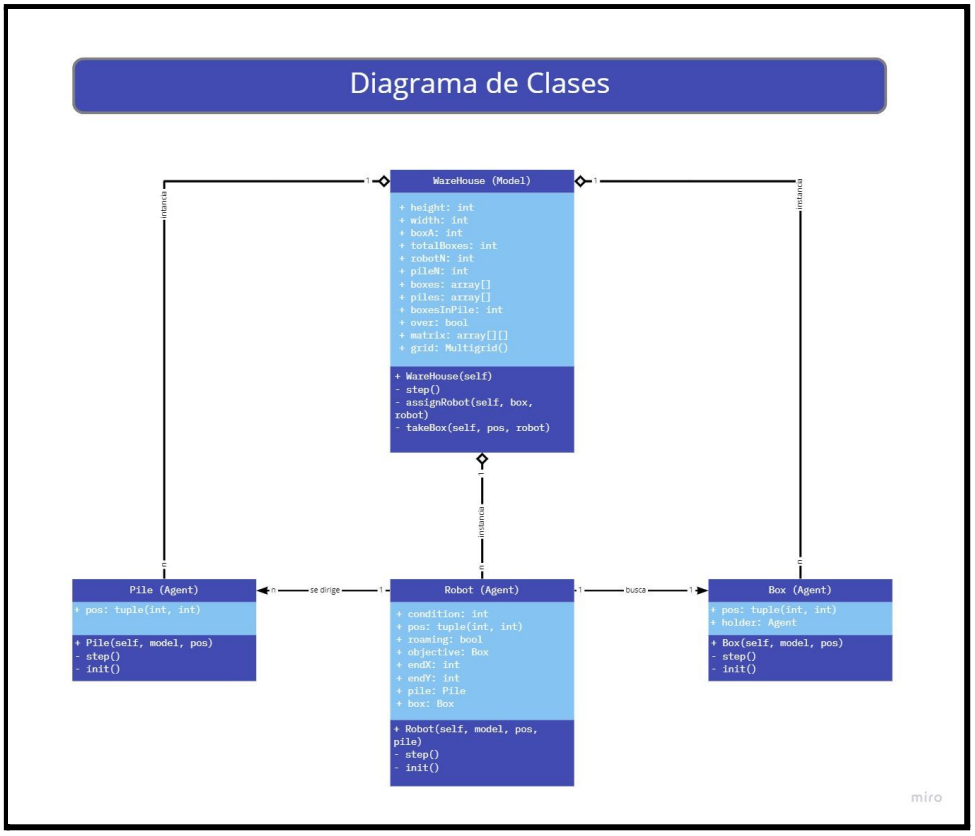
2.4 Áreas de oportunidad

Aunque se pudo generar una solución óptima que se apegaba a todo lo solicitado por la actividad integradora, existen áreas de oportunidad que pueden mejorar la solución en diferentes ámbitos.

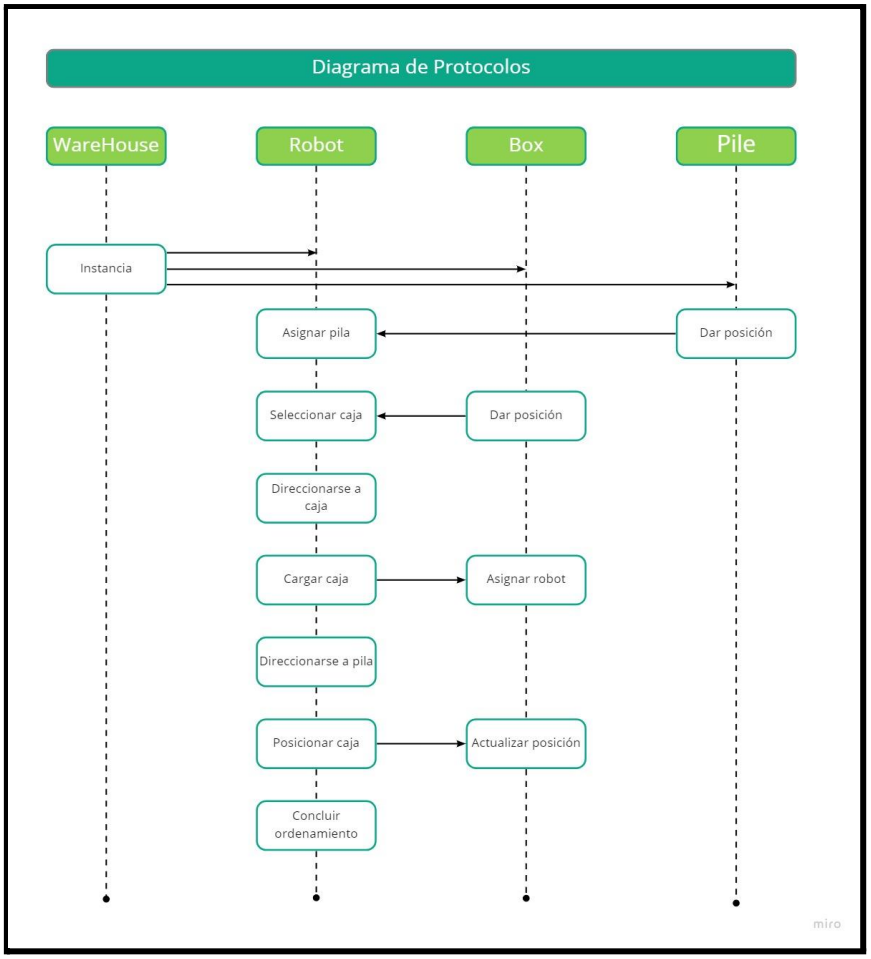
- Permitir que los agentes Robot lleven cajas a la pila más cercana y no solo a una sola asignada al inicio, para que entre todos los agentes colaboren para llenar la pila.
- Crear un ciclo dentro del backend.py que permita añadir todo tipo de agente y los introduzca dentro de la estructura de JSONs. De esta manera el código no estaría tan largo y sería algo más práctico al momento de introducir más agentes.
- Mostrar como las cajas forman una pila dentro de la animación para observar cuántas cajas hay en la pila.
- Permitir que el robot dentro de la animación observe a la caja/pila cuando se dirija a estas.
- Aplicar diferentes transformaciones a los GameObjects para mostrar una mejor interacción de los agentes.

3 Diagramas

3.1 Diagrama de clases



3.2 Diagrama de protocolos



4 Repositorio

URL repositorio: https://github.com/7Pablo/Actividad_integradora_A01734010

5 Conclusión

El objetivo principal de la actividad fue el de aplicar los conceptos aprendidos durante las clases en una actividad en concreto, y a su vez alcanzar el dominio de las subcompetencias de demostración del funcionamiento de los sistemas computacionales y generación de modelos computacionales. Durante la implementación de la solución, tuve la oportunidad de fortalecer mis conocimientos en temas de modelación e interacción de agentes, y modelación y animación de gráficas en tres dimensiones, por lo que fue fundamental para poder culminar las diferentes actividades como el reto. Considero que fue una

actividad retadora pero a su vez de gran valor que aporto a mis conocimientos, para que en un futuro pueda aplicar estos nuevamente.