Tecnológico de Monterrey

Evidencia 1 - Actividad Integradora

Grupo 101

Sadrac Aramburo Enciso - A01643639

Diego Alejandro Espejo López - A01638341

Diego Cisneros - A01643454

Daniel Esparza Arizpe - A01637076

Luis Fernando Li Chen - A00836174

27 de Agosto 2024

Table of Contents

1 Introducción

1.1 Propósito

El propósito de este documento es mostrar el trabajo realizado en la actividad integradora donde diseñamos, implementamos y evaluamos un sistema multiagente compuesto de robots autónomos que organizan un almacén desordenado. El proyecto se divide en tres partes: Simulación de la organización con agentes, modelado y despliegue en gráficos 3D, y la aplicación de visión computacional para la identificación de objetos.

1.2 Análisis del Problema

Existe un Almacén en forma de grid, dentro de este existen cajas en posiciones aleatorias, las cajas están en desorden, tendrán que ser organizadas por 5 robots que navegan el almacén, cada robot tiene la capacidad de recoger, mover y apilar objetos, y puede detectar el estado de una celda del grid mediante sensores. El desafío radica en coordinar a los robots para maximizar la eficiencia en términos de tiempo y movimientos, evitando colisiones y utilizando una estrategia óptima.

1.3 Descripción del Problema

El dueño anterior dejó el almacén en desorden, y la tarea es enseñar a robots a organizar los objetos en pilas de cinco. Los robots deben operar de manera autónoma utilizando razonamiento deductivo y práctico, detectando colisiones y ajustando su comportamiento en consecuencia, además tendrán la capacidad de reconocer los objetos utilizando visión computacional. Se nos entrega el proyecto para crear un software utilizando agentes múltiples y gráficas computacionales, es tarea de nuestro equipo enseñar a los robots a recorrer el almacén y ordenar las cajas dentro de él.

2 Descripción General

2.1 Actores (Agentes)

2.1.1 Agente “Base”: Se creó un agente Base que aparece aleatoriamente dentro del Almacén y es la división de N cajas entre 5, la posición del agente Base será donde el agente Robot apile al agente caja.

2.1.2 Agente “Robot”: El agente robot sólo puede moverse hacia enfrente y puede dar vueltas hacia el Norte, Sur, Este y Oeste, lo que significa que este agente no puede girar en diagonal, tiene como objetivo buscar una al agente caja en sus alrededores, si la encuentra, se pone en su posición, la agarra y busca al agente Base. Si no tiene cajas en sus alrededores, se mueve de manera aleatoria y se repite el proceso.

2.1.3 Agente “Caja”: El agente caja está repartido por el Almacén de manera aleatoria y busca que un Robot llegue a su posición y lo lleve a una Base.

2.2 Ambiente

2.2.1 El ambiente en este modelo es un almacén organizado en una cuadrícula bidimensional (Grid), donde cada celda de la cuadrícula puede contener un agente (un robot, una caja o una base). El ambiente es responsable de definir las reglas de movimiento, la detección de colisiones, y las interacciones entre los diferentes agentes.

2.3 Propiedades de Agentes y Ambiente

2.3.1 Los robots deben demostrar proactividad al planificar la recolección de cajas y su apilamiento en las bases, anticipándose a posibles bloqueos en el almacén y ajustando su comportamiento para maximizar la eficiencia.

2.3.2 Los robots muestran reactividad al detectar la presencia de cajas o bases cercanas y al evitar colisiones con otros robots.

2.3.3 La socialización entre los robots se manifiesta en la forma en que evitan colisiones y coordinan sus movimientos en un entorno compartido. Por ejemplo, un robot podría ceder el paso a otro robot que lleva una caja para evitar colisiones.

2.3.4 El ambiente del almacén es parcialmente accesible. Los robots tienen acceso a información sobre las celdas adyacentes mediante sensores, pero no pueden percibir todo el almacén en su totalidad al mismo tiempo.

2.3.5 El ambiente es no determinista debido a la presencia de múltiples robots que actúan de manera autónoma. Los movimientos y decisiones de otros robots pueden alterar el resultado de las acciones de un robot individual, introduciendo incertidumbre.

2.3.6 El ambiente del almacén es dinámico, ya que los robots se mueven y cambian la disposición de las cajas y bases mientras realizan sus tareas, lo que afecta a las decisiones y acciones de otros robots.

2.3.7 El ambiente es discreto. El almacén está dividido en una cuadrícula con un número finito de celdas, y los robots sólo pueden moverse en direcciones específicas (arriba, abajo, izquierda, derecha) a celdas adyacentes.

2.4 Restricciones

2.4.1 Si el agente Robot encuentra una caja y la lleva a una Base que ya está ocupada por otra caja tiene que ir a buscar otra Base para llevar la caja.

2.4.2 Si dos Robots están a punto de chocar, se da prioridad al robot que tenga una caja, si ambos o ninguno tienen caja, se elige un robot de manera aleatoria para que se mueva.

2.5 Métricas de Utilidad y Éxito

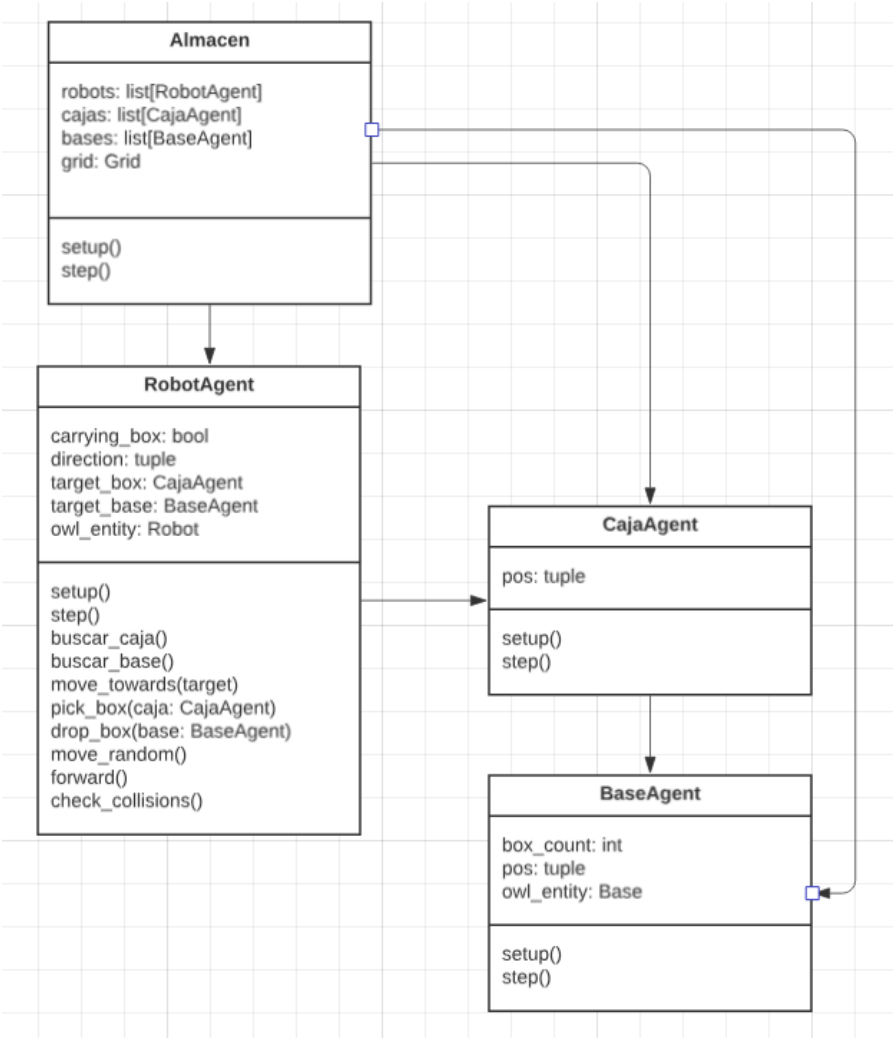
2.5.1 La utilidad es una forma de medir el desempeño de un agente con respecto a la tarea delegada que están resolviendo. Para los agentes robot, la utilidad se basa en cajas recogidas, la utilidad se mide por número de cajas recogidas entre el número de cajas en el almacén, el número de cajas entregadas a bases entre el total de cajas transportadas y el tiempo promedio por tarea.

2.5.2 El éxito de un agente se refiere al grado en que contribuye al cumplimiento del objetivo global del sistema, que en este caso es organizar el almacén en pilas de máximo cinco objetos. Las métricas de éxito podrían incluir si se mide cuántas de las cajas recogidas y entregadas por un robot han contribuido a completar

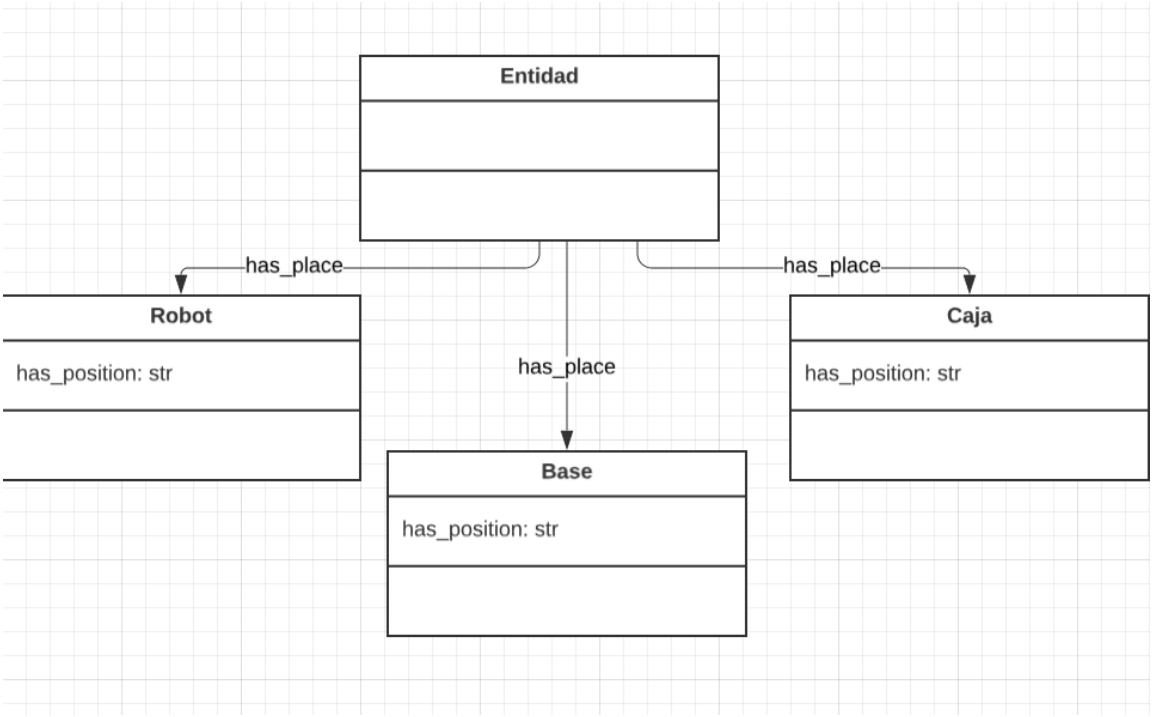
pilas de cinco cajas, evalúa cómo las acciones del robot han contribuido a reducir el tiempo total necesario para completar la organización del almacén y mide cuán eficientemente un robot ha utilizado el espacio en el almacén, minimizando el movimiento y maximizando el uso de las bases.

3 Diagramas

3.1 Diagrama de clases de Agentes



3.2 Diagrama de clases de Ontologías



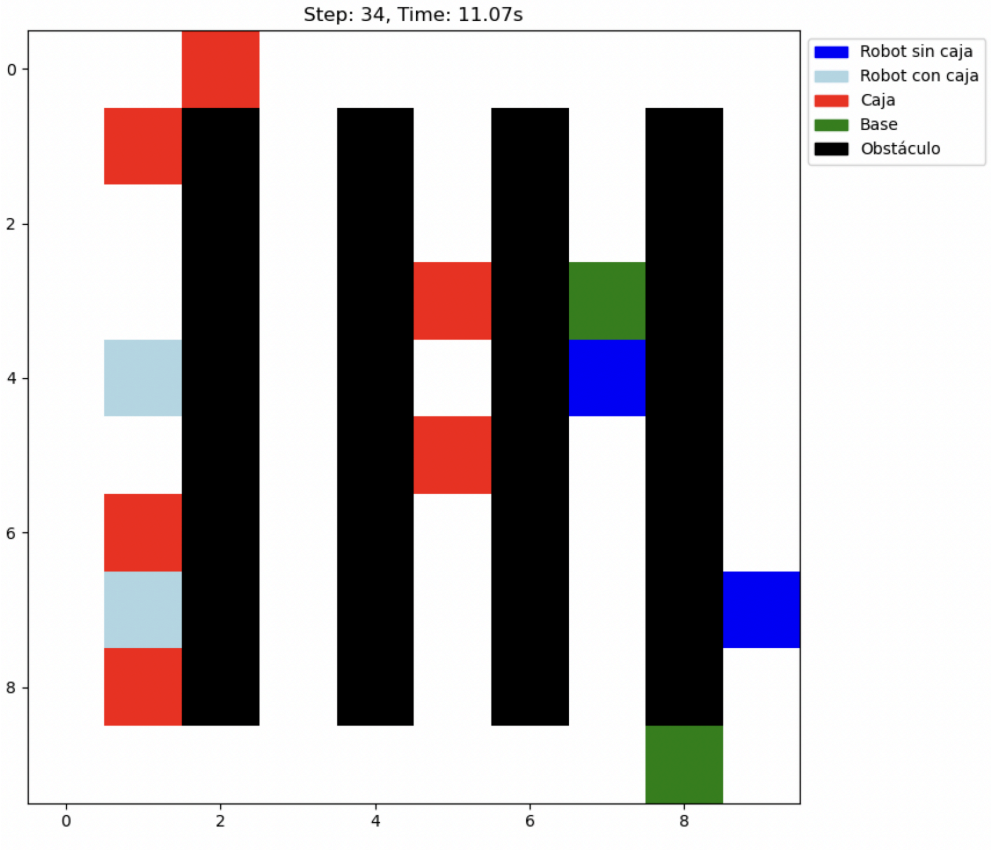
4 Recopilación de Información

4.1 Tiempo Necesario y Número de Movimientos

El tiempo necesario para terminar la ejecución del programa con obstáculos no puede ser determinado con facilidad, ya que los movimientos de los robots, al no tener una caja, son aleatorios, lo que significa que pueden tardar mucho en encontrar una caja, o en su contrario, poco tiempo.

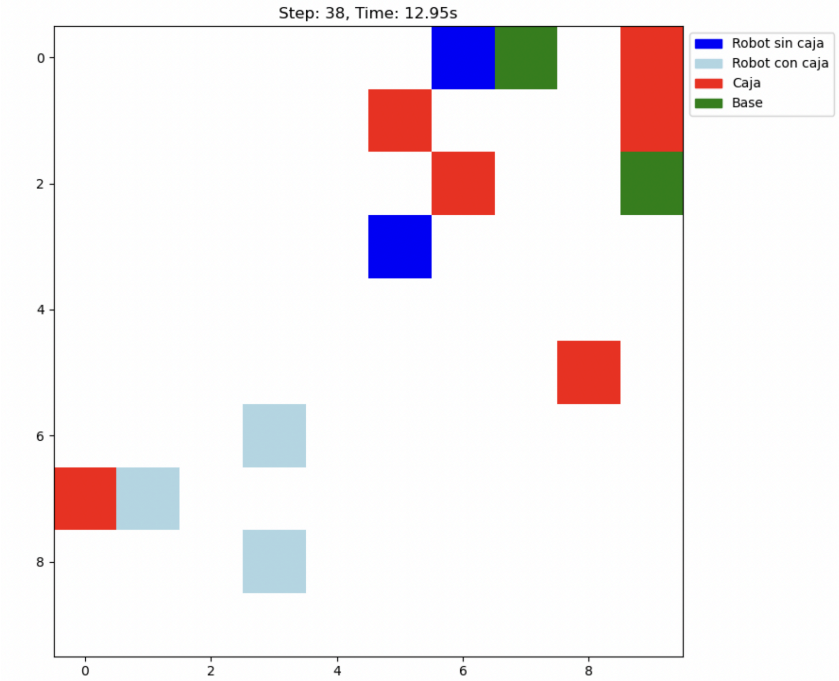
El número máximo permitido de movimientos en nuestro programa con obstáculos es de 1000 steps, no se puede confiar en un promedio para el número de movimientos debido a la aleatoriedad del programa.

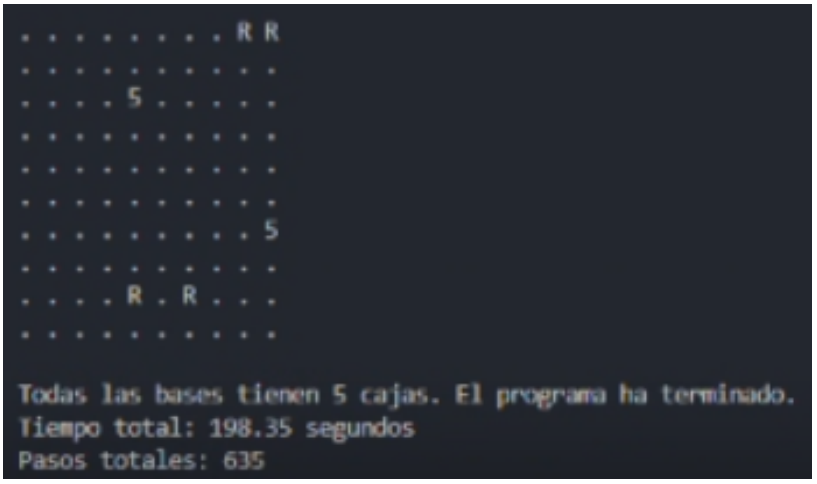
Las siguientes imágenes muestran la ejecución de los programas con obstáculos (“X”) y sin obstáculos respectivamente, así como en formato gráfico con matplotlib:




```
. . . . . . . . . .
. . X . X . X . X .
. . X . X . X . X .
. . X . X . X . X .
. 5 X . X . X . X .
. . X . X . X . X .
. . X . X . X R X .
. . X 5 X . X . X .
. R X . X . X . X .
. . . . R . . . . R

Todas las bases tienen 5 cajas. El programa ha terminado.
Tiempo total: 203.48 segundos
Pasos totales: 650
```



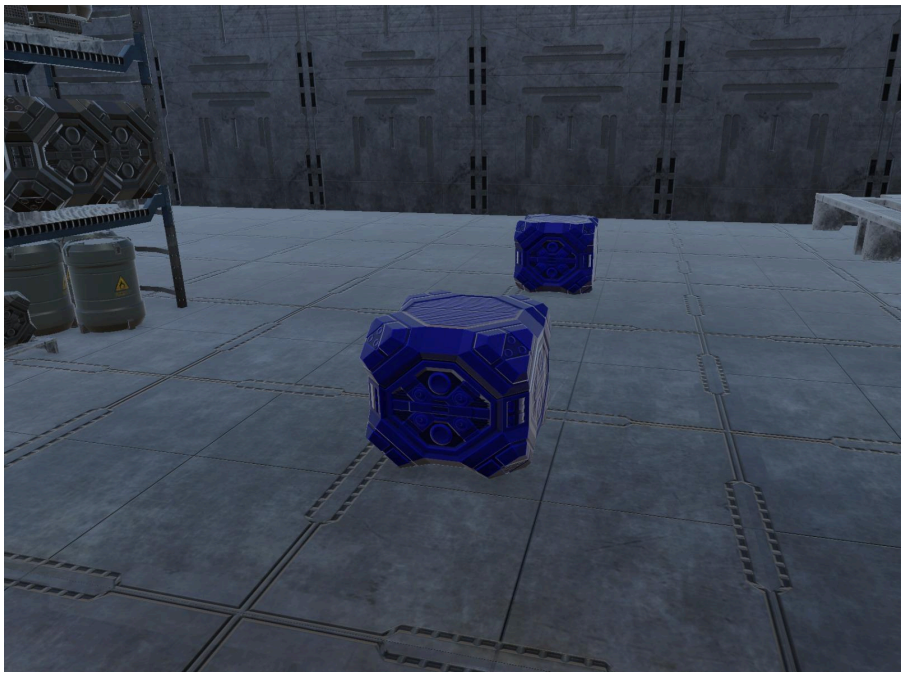


5 Gráficos Computacionales

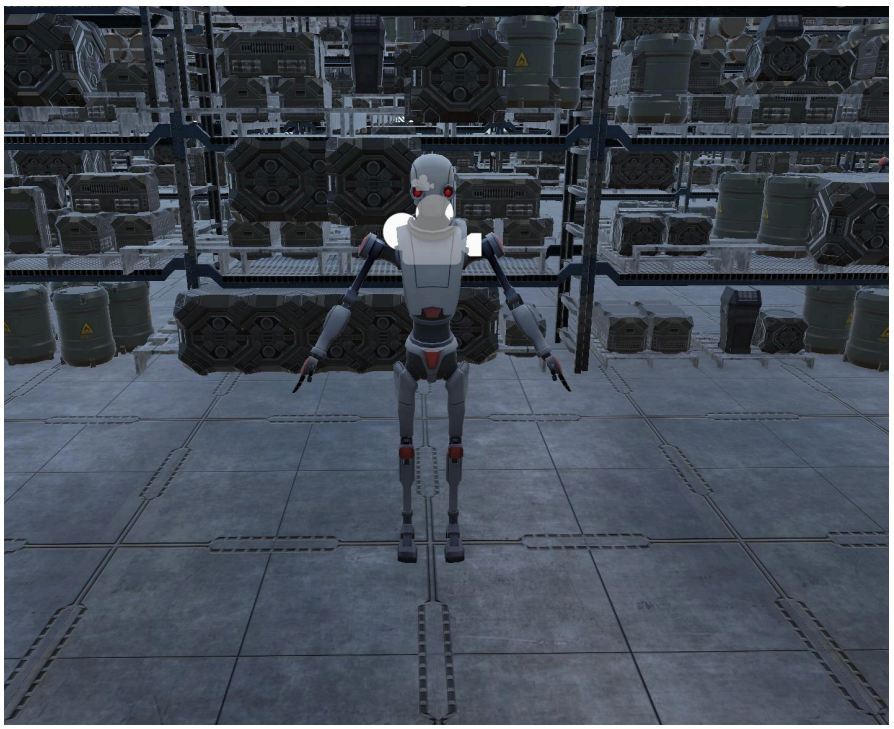
5.1 Materiales

5.1.1 Estantes (Obstáculos):

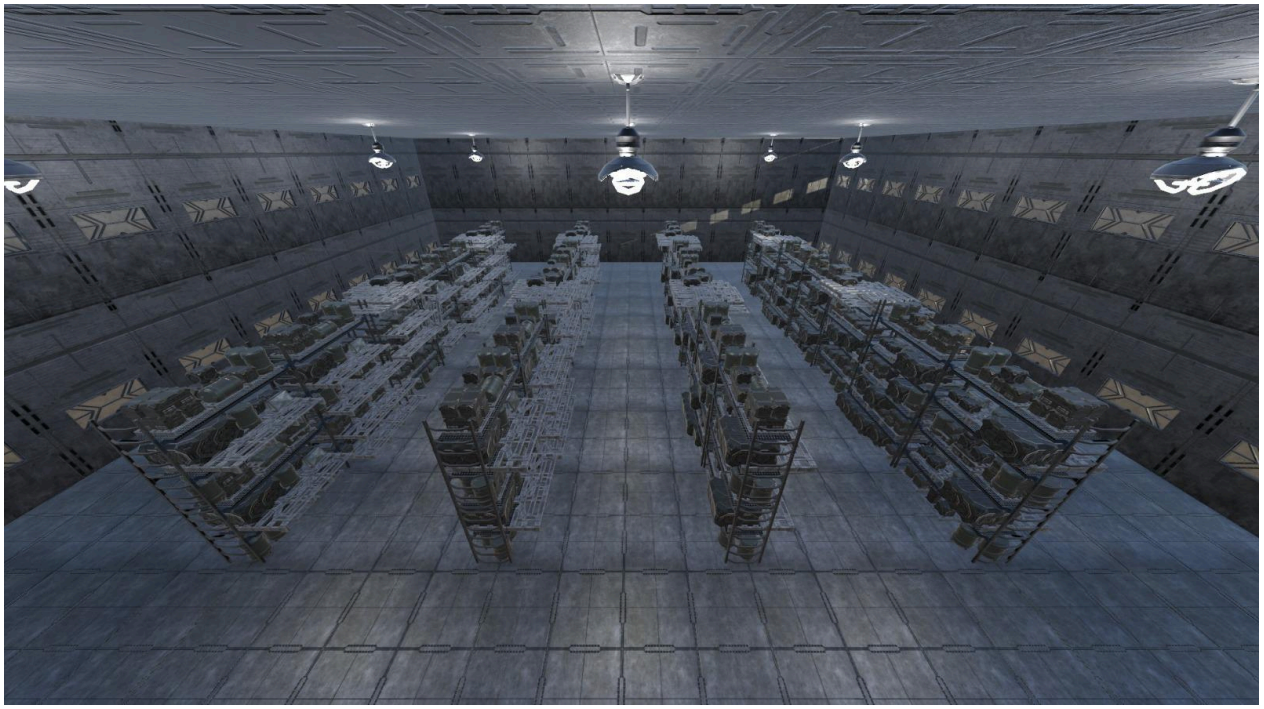
5.1.2 Objetos Varios:



5.1.3 Robot:

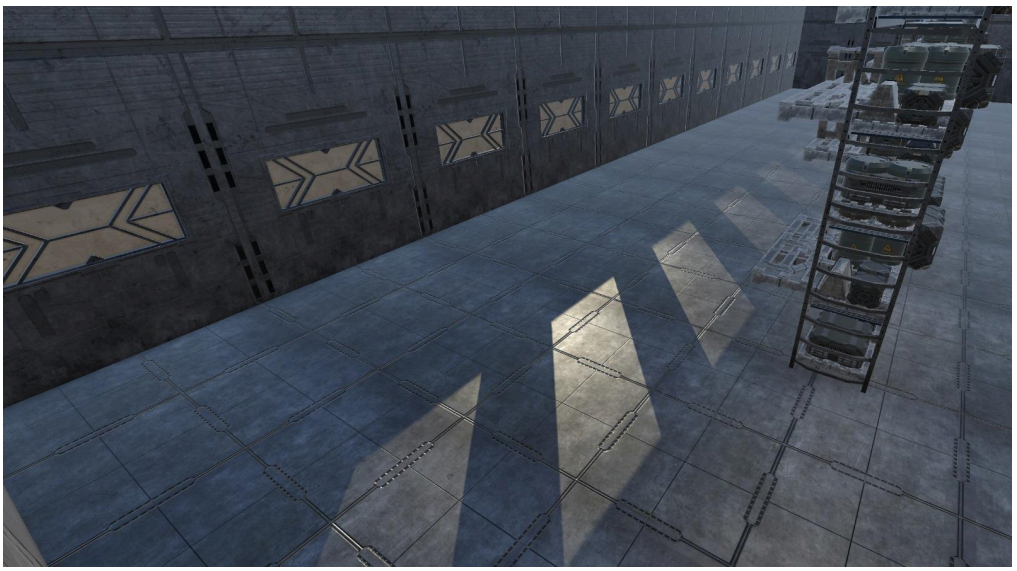


5.1.4 Almacén:

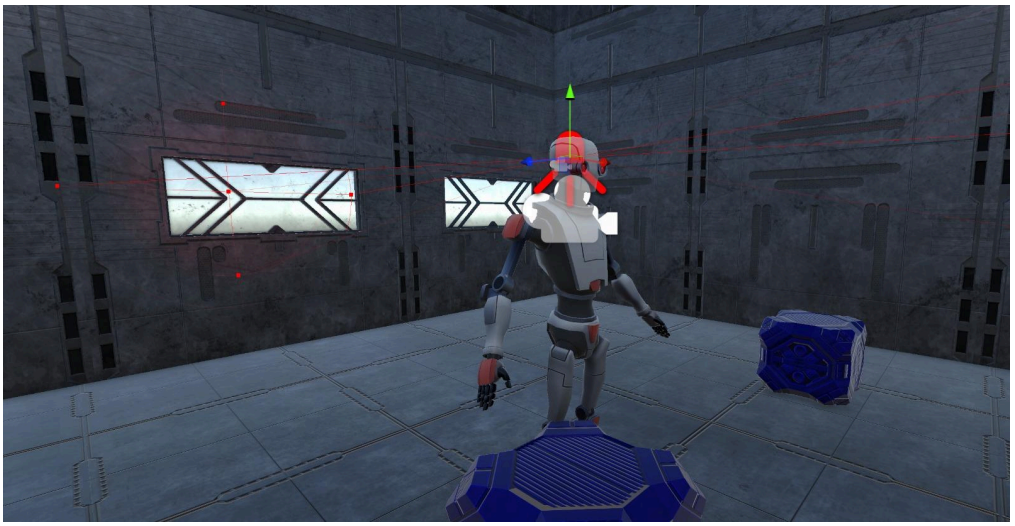


5.2 Iluminación

5.2.1 Iluminación Direccional:



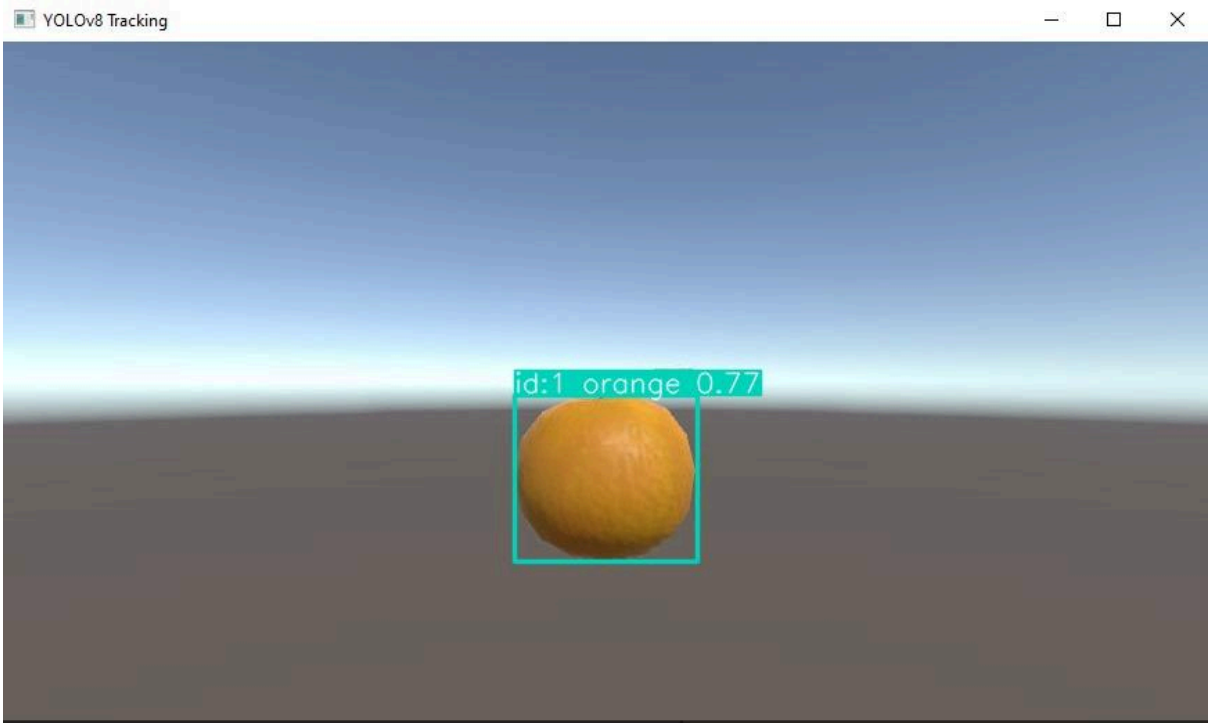
5.2.2 Iluminación Puntual:



6 Visión Computacional

6.1 Modelo de Visión

Para el modelo de visión escogimos YOLO, ya que nuestra prioridad es una detección de objetos rápida y eficiente, con una buena precisión y simplicidad, especialmente en aplicaciones en tiempo real. Para tareas más especializadas, como la segmentación detallada (SAM) o la generación de contenido (modelos de OpenAI), los otros modelos serían más adecuados.



6.2 Interacción con Objetos

El propósito del modelo de visión es la detección de objetos, en nuestro programa tendremos objetos como naranjas dispersas en nuestro grid (Almacén) se van a detectar la presencia de las naranjas y la interacción producirá texto en pantalla.

7 Funcionalidades

7.1 WebSocket

En este proyecto, se optó por utilizar WebSockets en lugar de sockets tradicionales o una REST API para la comunicación entre el servidor de Python y Unity. Más que nada, la principal razón para elegir WebSockets es que estos ofrecen una conexión full-duplex persistente, lo que permite la transmisión bidireccional continua de datos sin la necesidad de establecer una nueva conexión para cada mensaje.

Esto resulta ideal para aplicaciones en tiempo real, como la simulación de agentes en nuestro ambiente, donde los datos deben ser enviados y recibidos de manera constante y en tiempo real. A diferencia de una REST API, que opera bajo un modelo de solicitud-respuesta donde cada interacción requiere una nueva conexión HTTP, WebSockets permiten un flujo de datos continuo, reduciendo la latencia y mejorando la eficiencia en la comunicación (así como permitiendo recibir al mismo tiempo, datos de Unity para la visión computacional). La implementación de WebSockets con Unity se logró utilizando la librería websockets en Python, y en Unity, el paquete WebSocketSharp.

```python
async def handler(websocket, path):
        # Avanzar un paso en el modelo
        model.step()

        # Calcular la duración del programa
        duration = time.time() - start_time

        # Preparar los datos para enviar
        data = {
            'robots': [],
            'cajas': [],
            'bases': [],
            'duration': duration,
            'steps': model.t,
        }

        for robot in model.robots:
            x, y = model.grid.positions[robot]
            data['robots'].append({'x': (x*6)+3, 'y': -(y*6)+3, 'carrying_box': robot.carrying_b

            # Si el robot lleva una caja, enviar la posición de la caja
            if robot.carrying_box and robot.caja_carrying is not None:
                data['cajas'].append({'x': (x*6)+3, 'y': -(y*6)+3})  # La caja sigue la posición

        for caja in model.cajas:
            if caja in model.grid.positions:
```

PROBLEMS OUTPUT TERMINAL PORTS COMMENTS

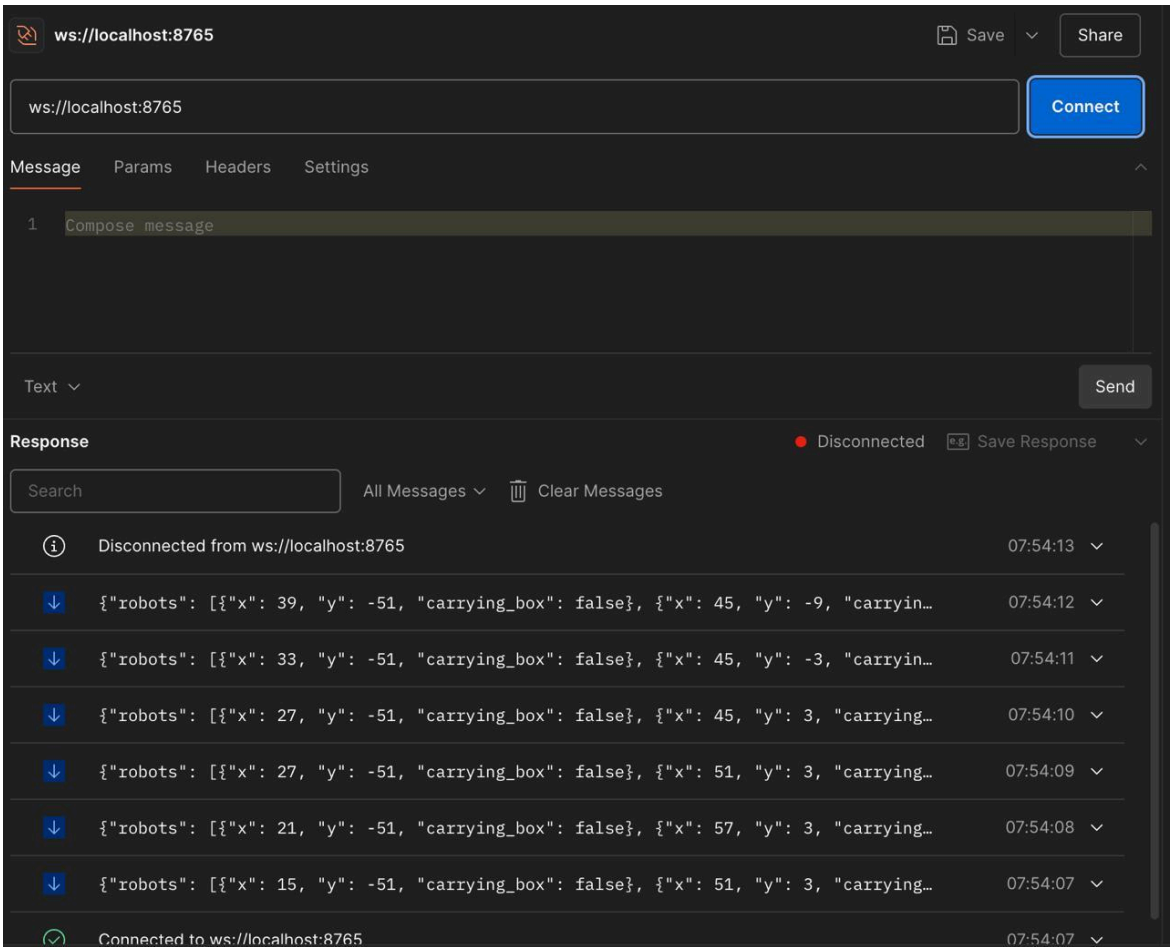
```
RobotAgent (Obj 4): recogió una caja en (8, 5)
RobotAgent (Obj 5): recogió una caja en (8, 0)
RobotAgent (Obj 1): recogió una caja en (2, 6)
RobotAgent (Obj 2): recogió una caja en (3, 4)
RobotAgent (Obj 2): dejó una caja en la base en (1, 4)
RobotAgent (Obj 3): recogió una caja en (6, 7)
RobotAgent (Obj 3): dejó una caja en la base en (7, 8)
RobotAgent (Obj 3): recogió una caja en (6, 9)
RobotAgent (Obj 3): dejó una caja en la base en (7, 8)
RobotAgent (Obj 3): recogió una caja en (7, 9)
RobotAgent (Obj 5): dejó una caja en la base en (1, 4)
RobotAgent (Obj 3): dejó una caja en la base en (7, 8)
RobotAgent (Obj 3): recogió una caja en (9, 7)
```

7.2 Data JSON

El servidor WebSocket transmite el estado del modelo AgentPy a través de un formato JSON estructurado que incluye detalles sobre los robots, las cajas y las bases en la simulación. Cada robot se identifica con un “id” y se proporciona su posición (x,y), además de un indicador booleano “carrying_box” que señala si está llevando una caja, y el “box_id” de la caja que está transportando. Las cajas y las bases también se representan con identificadores únicos y posiciones (x,y), con las bases incluyendo un conteo de las cajas actualmente almacenadas. Y se manda el tiempo transcurrido desde que el modelo se corrió en “duration”, la cantidad de pasos que ha tomado el modelo hasta ese punto “steps”, y si ya se acabó el modelo “complete”.

Si el modelo se termina, la conexión por websocket se cierra y se manda un último JSON con la duración, los steps, y si se completó o no.

Este enfoque JSON permite una comunicación eficiente y en tiempo real entre el servidor y Unity, facilitando la visualización dinámica y precisa del estado de la simulación. La estructura del JSON asegura que se mantenga un registro claro de la posición y el estado de cada agente y objeto en el entorno, lo que es crucial para actualizar la simulación en Unity y reflejar los cambios en la posición de las cajas, así como el progreso general del modelo. Esto ayuda a lograr una integración fluida y efectiva con el entorno 3D en Unity.



Comunicación entre el Python script y el servidor por medio de JSON:


```
{
  "robots": [
      {
          "id": 1,
          "x": 57,
          "y": 3,
          "carrying_box": false
      },
      {
          "id": 2,
          "x": 39,
          "y": -39,
          "carrying_box": true,
          "box_id": 6
      },
      {
          "id": 3,
          "x": 27,
          "y": -39,
          "carrying_box": false
      },
      {
          "id": 4,
          "x": 51,
          "y": 3,
          "carrying_box": false
      },
      {
          "id": 5,
          "x": 3,
          "y": -33,
          "carrying_box": false
      }
  ],
  "cajas": [
      {
          "id": 6,
          "x": 39,
          "y": -39
      },
      {
          "id": 7,
          "x": 27,
```

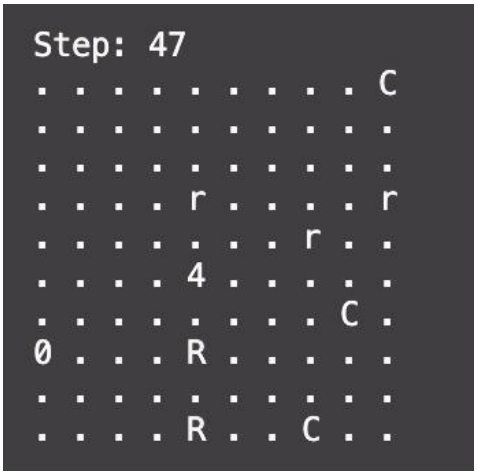
```
            "y": -51
        },
        {
            "id": 8,
            "x": 39,
            "y": -27
        },
        {
            "id": 9,
            "x": 57,
            "y": -27
        },
        {
            "id": 10,
            "x": 27,
            "y": -27
        },
        {
            "id": 11,
            "x": 21,
            "y": -15
        },
        {
            "id": 12,
            "x": 39,
            "y": 3
        },
        {
            "id": 13,
            "x": 9,
            "y": -15
        },
        {
            "x": 3,
            "y": -39
        },
        {
            "x": 51,
            "y": 3
        }
    ],
    "bases": [
        {
            "id": 16,
            "x": 51,
```

```
            "y": 3,
            "box_count": 1
        },
        {
            "id": 17,
            "x": 3,
            "y": -39,
            "box_count": 1
        }
    ],
    "duration": 4.897151947021484,
    "steps": 4,
    "complete": false
}
```

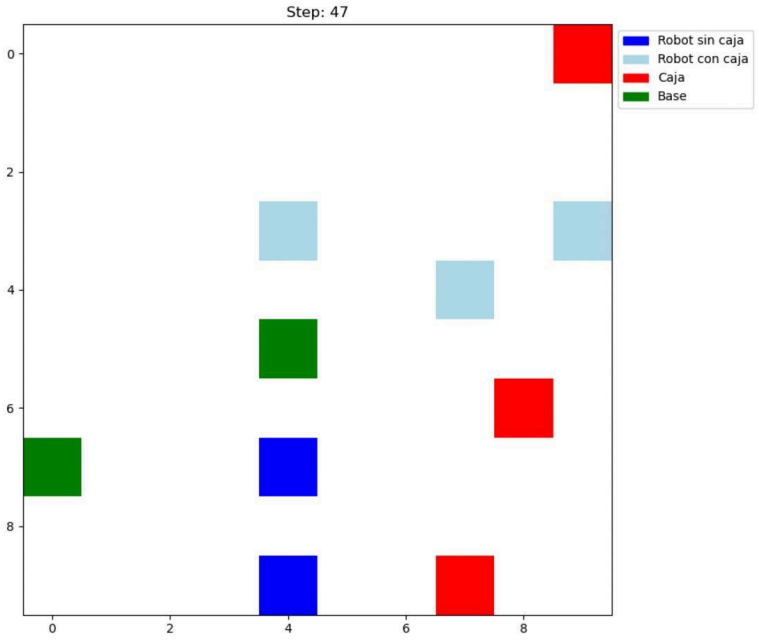
```
{
    "type": "final",
    "duration": 636.91408586502075,
    "steps": 634,
    "complete": true
}
```


7.3 Ejecución

7.3.1 En terminal (Ambiente sin obstáculos): r = robot con caja, R = robot sin caja, C = caja, Núm = Base y cantidad de cajas en ella.

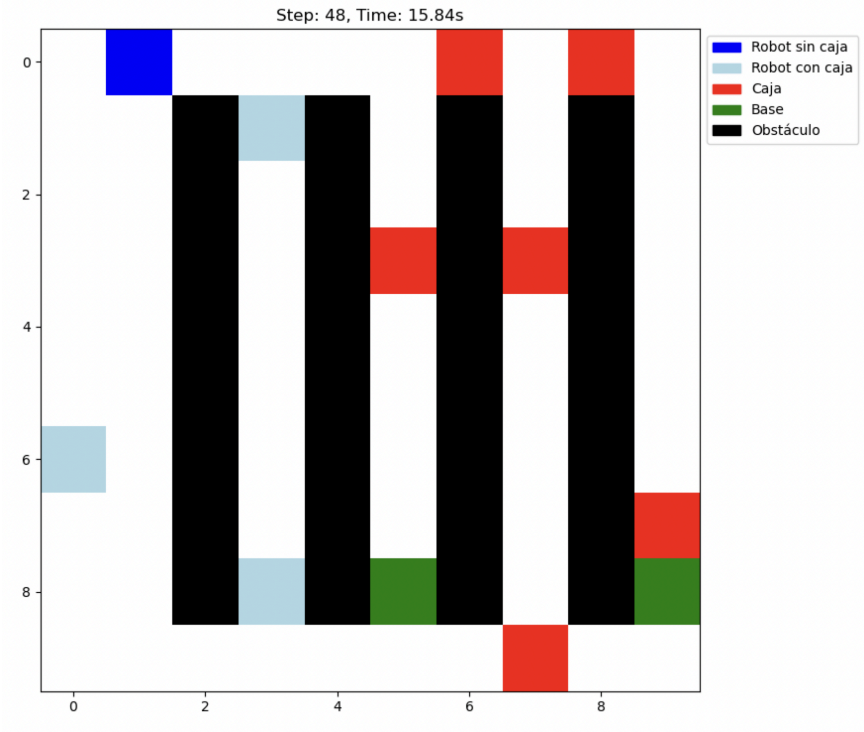


7.3.2 Visual



7.3.3 En terminal (Ambiente con obstáculos): r = robot con caja, R = robot sin caja, C = caja, Núm = Base y cantidad de cajas en ella, X = Obstáculos (Estantes).

7.3.4 Visual



8 Conclusión. Estrategia de Disminución de Tiempo y Movimientos

8.1 Comunicación y Coordinación de Robots

Implementar un sistema básico de comunicación entre robots para que puedan coordinarse mejor. Por ejemplo, cuando un robot encuentra una caja, podría notificar a otro robot especializado en transporte.

8.2 Predicción y Planificación

Antes de moverse, cada robot podría predecir, por medio de heurísticas, la posición futura de los otros robots en función de su velocidad y dirección. Esto permitiría evitar colisiones sin necesidad de detenerse, lo que reduce la cantidad de movimientos y el tiempo total.

8.3 Reducción de Movimientos Aleatorios

Limitar el movimiento aleatorio cuando no hay cajas ni bases cercanas detectadas, y en su lugar, mover al robot hacia una zona del almacén menos explorada. Esto podría lograrse dividiendo el almacén en sectores y asignando sectores a cada robot.

8.4 Conclusión

Implementar estas estrategias requeriría modificar tanto la lógica de los robots como la forma en que interactúan con el entorno, pero debería resultar en una reducción en el tiempo total de ejecución y de los movimientos necesarios para completar la tarea.

9 GitHub

https://github.com/Spiegelin/Sistemas-Multiagentes