Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Guadalajara

17/11/24

Modelación de sistemas multiagentes con gráficas computacionales

Evidencia 1. Actividad Integradora

Autores:

Fernanda Diaz Gutierrez | A01639572

Miguel Angel Barrientos Ballesteros | A01637150

Carlos Iván Armenta Naranjo | A01643070

Jorge Javier Blazquez Gonzalez | A01637706

Gabriel Alvarez Arzate | A01642991

Evidencia 1: Actividad Integradora

En este documento se presentan especificaciones y detalles sobre la implementación de la simulación de la Evidencia 1. Como breve contexto, nuestra solución al problema fue la siguiente:

Se creó un ambiente virtual el cual representa un almacén de 20 por 20 coordenadas. Dentro de dicho almacén, se crearon 5 diferentes tipos de objetos, 5 de cada uno (25 objetos en total) en coordenadas aleatorias. Además, se crearon 5 robots diferentes, también en coordenadas aleatorias. Se diseñó un modelo multiagente el cual simula un sistema en el que los robots se mueven por el almacén, recogen los objetos del tipo que les corresponde y los acomodan en grupos de 5 en los bordes del almacén (junto a las paredes).

Propiedades de los agentes

En el sistema, existen 5 agentes diferentes, uno por cada robot. Dichos agentes son prácticamente iguales, con diferencias mínimas como ID, entre otras cosas.

Cada robot está implementado para funcionar con un razonamiento deductivo. El robot conoce la ontología diseñada para el sistema, el cual le ayuda a comunicarse principalmente con el ambiente.

Desde un inicio, el agente es capaz de ver las coordenadas del almacén que están ocupadas por objetos, así como aquellas ocupadas por otros robots. Cada agente, individualmente, decide elegir como objetivo el objeto más cercano a él. En cada step de la simulación, el robot decide a qué casilla moverse después, para acercarse a su objetivo, evitando casillas ocupadas por objetos o robots. Cuando el robot está junto a su objetivo, ve el tipo de objeto que es y decide si tomarlo o no tomarlo, dependiendo si es el tipo de objeto que le corresponde recoger. De hacerlo, decide llevar el objeto a un borde del almacén para acomodarlo.

De esta forma, cada robot se hace su camino por el almacén, recogiendo objetos y acomodándolos, evitando colisiones e ignorando objetos que no son suyos.

Las propiedades principales de los agentes (robots) implementados son las siguientes:

- **Reactividad:** Los agentes mantienen una interacción continua con el ambiente: ven los cambios en coordenadas ocupadas, recogen objetos y ocupan coordenadas nuevas (al moverse o acomodar objetos). Esto les permite decidir sus acciones acorde al estado actual del ambiente.
- **Proactividad:** Los agentes tienen la capacidad de planear a futuro. Estos fijan la meta de recoger el objeto más cercano a ellos, de forma que tienen la intención de moverse hasta estar cerca de él. Esta meta cambia a medida que recogen y acomodan nuevos objetos, o cuando llegan a un objeto que no es suyo.
- **Habilidad Social:** El sistema tiene un ambiente multiagente. Los robots no pueden cumplir sus objetivos sin tomar en cuenta a los demás, ya que deben

considerarlos para evitar colisiones y no acomodar objetos en los grupos de otros. Esto también implica cierta coordinación en los movimientos.

Propiedades del ambiente

Como se mencionó anteriormente, el ambiente en este sistema representa un almacén de 20 por 20 coordenadas. La información del ambiente se basa en coordenadas ocupadas por robots, coordenadas ocupadas por objetos, y coordenadas en donde hay grupo de objetos acomodados.

Las propiedades principales del ambiente son las siguientes:

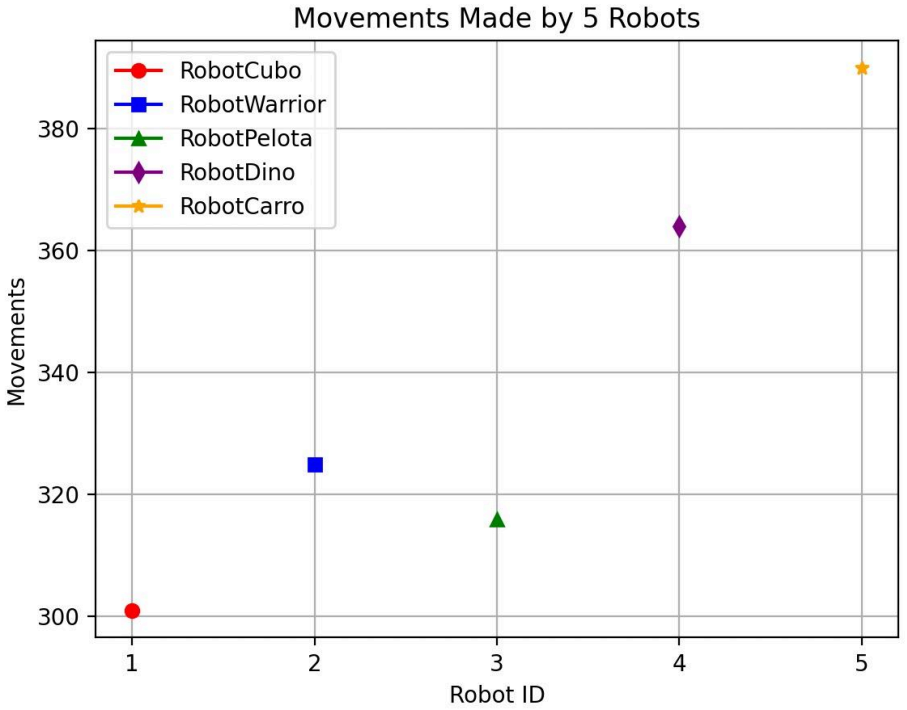
- **Accesible:** El ambiente es completamente accesible en todo momento. Los agentes pueden acceder a toda información relevante sobre el estado del este, y este se modifica si ellos hacen algo (recoger/dejar objetos, moverse, etc.).
- **No determinista:** Si se considera que los objetos y robots empiezan en coordenadas aleatorias, puede decirse que el ambiente no es del todo determinista. Además, los robots en ocasiones deciden moverse por un objeto aleatorio, lo que provoca que los estados del ambiente no siempre puedan ser predecidos en todo momento.
- **Episódico:** Los agentes deciden siempre únicamente basados en la información del ambiente en el episodio actual. No consideran posibilidades futuras o información que aún no tienen.
- **Estático:** El ambiente no tiene ningún proceso corriendo por sí solo. Todo cambio en este es causado por las acciones de los robots, lo que implica que el ambiente sea completamente estático.
- **Discreto:** Similar al ajedrez, hay una cantidad fija de posibles acciones, lo que vuelve al ambiente discreto.

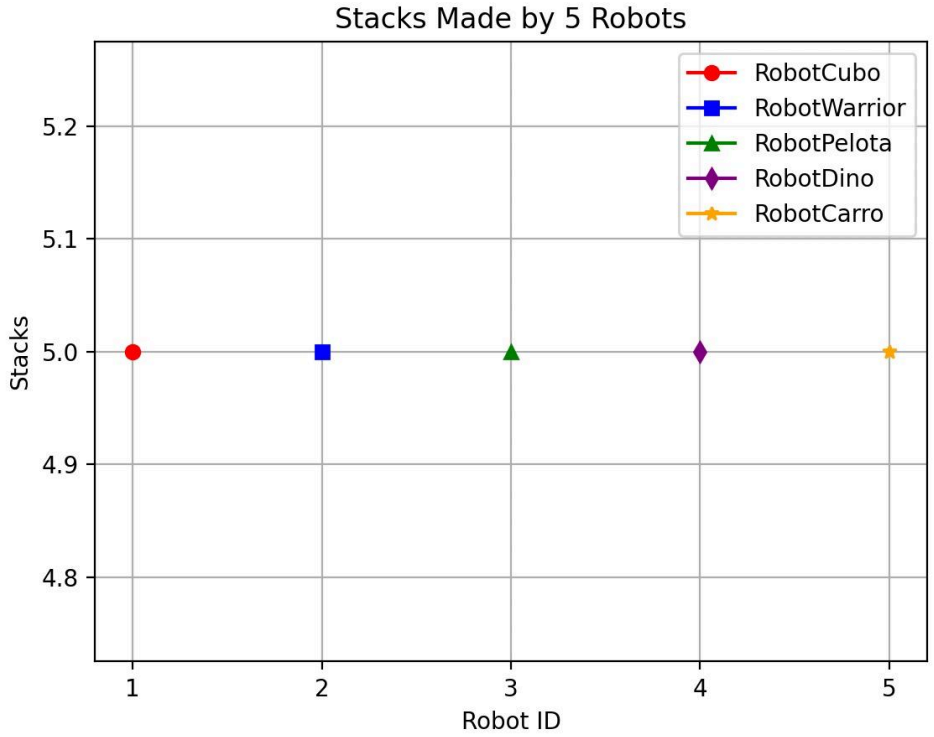
Métrica de utilidad o éxito.

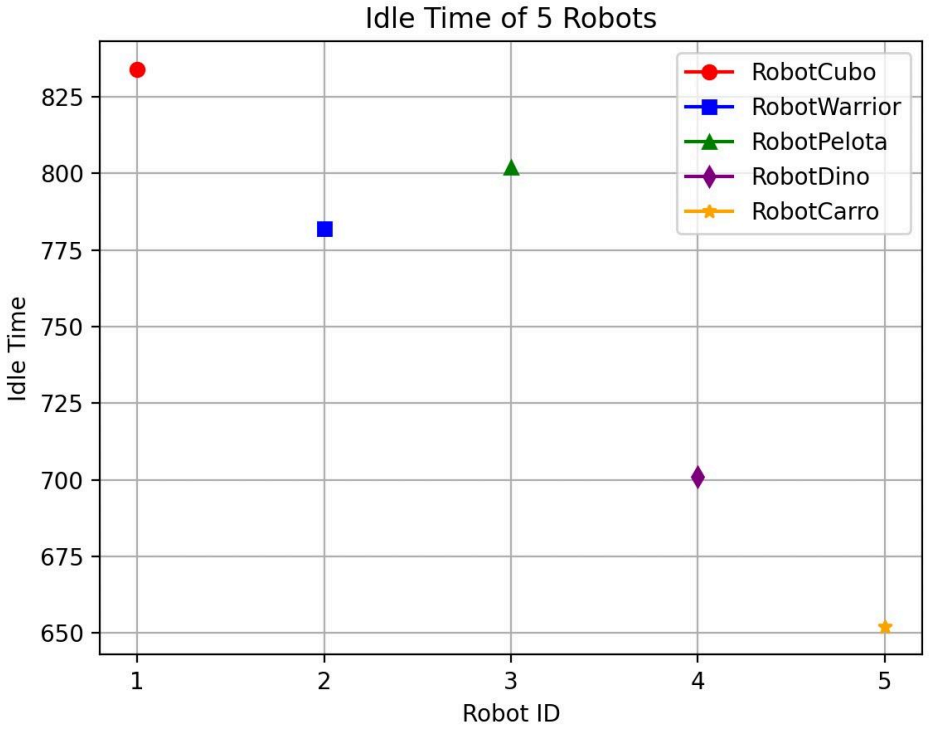
El objetivo final de los robots es limpiar completamente el almacén. Durante la simulación, se van recopilando datos estadísticos sobre el comportamiento de cada robot, lo cual determina su utilidad. Estos datos se basan en objetos que recogen, objetos que acomodan, movimientos que hacen y cantidad de steps en que se quedan esperando o pensando.

- El éxito final de un agente se mide en que hayan acomodado todos los objetos que le corresponden.
- La utilidad aumenta cuando un robot recoge o acomoda un step, y disminuye cuando se queda quieto.

A continuación un ejemplo de los resultados estadísticos de una corrida de la simulación (Gráficas):






Idle Time of 5 Robots
RobotCubo
RobotWarrior
RobotPelota
RobotDino
RobotCarro
Idle Time
825
800
775
750
725
700
675
650
1
2
3
4
5
Robot ID

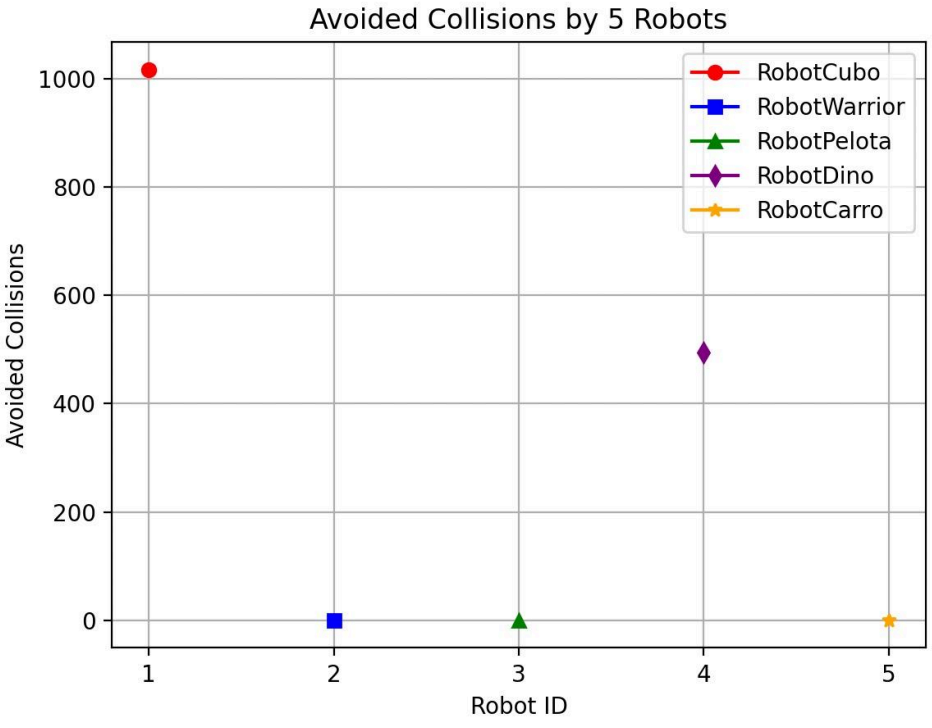

Avoided Collisions by 5 Robots
RobotCubo
RobotWarrior
RobotPelota
RobotDino
RobotCarro
Avoided Collisions
1000
800
600
400
200
0
1
2
3
4
5
Robot ID

Diagrama de clase de los agentes

RobotCubo

State Description:

- **robotID (int):** Identificador único para el robot (0 en este caso)
- **state (int):** Estado actual del robot (0 = chasing, 1 = carrying, 2 = idle)
- **position (tuple):** Coordenadas actuales (x,y) del robot
- **stackAmount (int):** Número de objetos actualmente "stackeados" por el robot. Range None - 5
- **currentStack (tuple):** Tupla de la coordenada (x,y) del stack con el que el robot está interactuando (o None).
- path (list): El camino planeado por el robot, coordenadas (x,y).
- **lastGoAround (str):** Trackea si el robot se movió Horizontalmente (H) ó Verticalmente (V) para evitar una colisión.
- **targetObjectType (int):** El tipo de objeto target para el robot (en este caso 0).
- **board (list):** Lista del tablero, para la percepción del robot.
- **done (bool):** Trackea si el robot ha completado sus tasks.
- **recorded_objects_handled (int):** Número de objetos que se stackearon.
- **recorded_movements_made (int):** Número de movimientos realizados por el robot.
- **recorded_steps_idle (int):** Total de steps en los que el robot estuvo detenido.
- **recorded_avoided_collisions (int):** Número de veces que el robot evitó una colisión.
- **recorded_utility (int):** Número de pick-ups exitosos del robot.

Actions:

- **setTarget():** Settea el target del robot (objeto o stack). Elimina el target de la lista de objetos disponibles cuando ya se setteó.
- **pickUp():** Recoge un objeto si coincide con el tipo del robot.
- **stack():** Apila el objeto en un stack. Reinicia currentStack cuando el stack completa 5 objetos.
- **moveTo():** Mueve al robot un tile hacia su target.
- **idle():** Mantiene al robot como idle y aumenta recorded_steps_idle.
- **setNextHorTile():** Determina el siguiente tile horizontal según el path planeado.
- **setNextVerticalTile():** Determina el próximo vertical tile.
- **goAroundHorizontal():** Maneja desvíos horizontales en caso de bloqueo.
- **goAroundVertical():** Maneja desvíos verticales en caso de bloqueo.

Methods:

- **setup():** Inicializa las variables y el robot state al comenzar la simulación.
- **step():** Ejecuta el ciclo de razonamiento y acción (percepción → razonamiento → ejecución).
- **see(e):** Percibe el entorno actual (boardTiles, por ejemplo).
- **next():** Evalúa las reglas y elige la siguiente acción.
- **action(act):** Ejecuta la acción seleccionada.
- **findPath():** Calcula la ruta hacia el objetivo actual.

Rules:

- **rule_canSetTarget(act):** Determina si el robot puede seleccionar un nuevo objetivo.
- **rule_canPick(act):** Evalúa si el robot puede recoger un objeto.
- **rule_canStack(act):** Valida si el robot puede apilar un objeto.
- **rule_canMove(act):** Verifica si el robot puede moverse al próximo paso.
- **rule_nextHor(act):** Determina si el robot debería moverse horizontalmente.
- **rule_nextVer(act):** Determina si el robot debería moverse horizontalmente.
- **rule_goAroundHorizontal(act):** Gestiona desvíos horizontales.
- **rule_goAroundVertical(act):** Gestiona desvíos verticales.
- **rule_idle(act):** Decide si el robot debería quedarse idle.

RobotWarrior

State Description:

- **robotID (int):** Identificador único para el robot (1 en este caso)
- **state (int):** Estado actual del robot (0 = chasing, 1 = carrying, 2 = idle)
- **position (tuple):** Coordenadas actuales (x,y) del robot
- **stackAmount (int):** Número de objetos actualmente "stackeados" por el robot. Range None - 5
- **currentStack (tuple):** Tupla de la coordenada (x,y) del stack con el que el robot está interactuando (o None).
- path (list): El camino planeado por el robot, coordenadas (x,y).
- **lastGoAround (str):** Trackea si el robot se movió Horizontalmente (H) ó Verticalmente (V) para evitar una colisión.
- **targetObjectType (int):** El tipo de objeto target para el robot (en este caso 1).
- **board (list):** Lista del tablero, para la percepción del robot.
- **done (bool):** Trackea si el robot ha completado sus tasks.
- **recorded_objects_handled (int):** Número de objetos que se stackearon.
- **recorded_movements_made (int):** Número de movimientos realizados por el robot.
- **recorded_steps_idle (int):** Total de steps en los que el robot estuvo detenido.
- **recorded_avoided_collisions (int):** Número de veces que el robot evitó una colisión.
- **recorded_utility (int):** Número de pick-ups exitosos del robot.

Actions:

- **setTarget():** Settea el target del robot (objeto o stack). Elimina el target de la lista de objetos disponibles cuando ya se setteó.
- **pickUp():** Recoge un objeto si coincide con el tipo del robot.
- **stack():** Apila el objeto en un stack. Reinicia currentStack cuando el stack completa 5 objetos.
- **moveTo():** Mueve al robot un tile hacia su target.
- **idle():** Mantiene al robot como idle y aumenta recorded_steps_idle.
- **setNextHorTile():** Determina el siguiente tile horizontal según el path planeado.
- **setNextVerticalTile():** Determina el próximo vertical tile.
- **goAroundHorizontal():** Maneja desvíos horizontales en caso de bloqueo.
- **goAroundVertical():** Maneja desvíos verticales en caso de bloqueo.

Methods:

- **setup():** Inicializa las variables y el robot state al comenzar la simulación.
- **step():** Ejecuta el ciclo de razonamiento y acción (percepción → razonamiento → ejecución).
- **see(e):** Percibe el entorno actual (boardTiles, por ejemplo).
- **next():** Evalúa las reglas y elige la siguiente acción.
- **action(act):** Ejecuta la acción seleccionada.
- **findPath():** Calcula la ruta hacia el objetivo actual.

Rules:

- **rule_canSetTarget(act):** Determina si el robot puede seleccionar un nuevo objetivo.
- **rule_canPick(act):** Evalúa si el robot puede recoger un objeto.
- **rule_canStack(act):** Valida si el robot puede apilar un objeto.
- **rule_canMove(act):** Verifica si el robot puede moverse al próximo paso.
- **rule_nextHor(act):** Determina si el robot debería moverse horizontalmente.
- **rule_nextVer(act):** Determina si el robot debería moverse horizontalmente.
- **rule_goAroundHorizontal(act):** Gestiona desvíos horizontales.
- **rule_goAroundVertical(act):** Gestiona desvíos verticales.
- **rule_idle(act):** Decide si el robot debería quedarse idle.

RobotPelota

State Description:

- **robotID (int):** Identificador único para el robot (2 en este caso)
- **state (int):** Estado actual del robot (0 = chasing, 1 = carrying, 2 = idle)
- **position (tuple):** Coordenadas actuales (x,y) del robot
- **stackAmount (int):** Número de objetos actualmente "stackeados" por el robot. Range None - 5
- **currentStack (tuple):** Tupla de la coordenada (x,y) del stack con el que el robot está interactuando (o None).
- path (list): El camino planeado por el robot, coordenadas (x,y).
- **lastGoAround (str):** Trackea si el robot se movió Horizontalmente (H) ó Verticalmente (V) para evitar una colisión.
- **targetObjectType (int):** El tipo de objeto target para el robot (en este caso 2).
- **board (list):** Lista del tablero, para la percepción del robot.
- **done (bool):** Trackea si el robot ha completado sus tasks.
- **recorded_objects_handled (int):** Número de objetos que se stackearon.
- **recorded_movements_made (int):** Número de movimientos realizados por el robot.
- **recorded_steps_idle (int):** Total de steps en los que el robot estuvo detenido.
- **recorded_avoided_collisions (int):** Número de veces que el robot evitó una colisión.
- **recorded_utility (int):** Número de pick-ups exitosos del robot.

Actions:

- **setTarget():** Settea el target del robot (objeto o stack). Elimina el target de la lista de objetos disponibles cuando ya se setteó.
- **pickUp():** Recoge un objeto si coincide con el tipo del robot.
- **stack():** Apila el objeto en un stack. Reinicia currentStack cuando el stack completa 5 objetos.
- **moveTo():** Mueve al robot un tile hacia su target.
- **idle():** Mantiene al robot como idle y aumenta recorded_steps_idle.
- **setNextHorTile():** Determina el siguiente tile horizontal según el path planeado.
- **setNextVerticalTile():** Determina el próximo vertical tile.
- **goAroundHorizontal():** Maneja desvíos horizontales en caso de bloqueo.
- **goAroundVertical():** Maneja desvíos verticales en caso de bloqueo.

Methods:

- **setup():** Inicializa las variables y el robot state al comenzar la simulación.
- **step():** Ejecuta el ciclo de razonamiento y acción (percepción → razonamiento → ejecución).
- **see(e):** Percibe el entorno actual (boardTiles, por ejemplo).
- **next():** Evalúa las reglas y elige la siguiente acción.
- **action(act):** Ejecuta la acción seleccionada.
- **findPath():** Calcula la ruta hacia el objetivo actual.

Rules:

- **rule_canSetTarget(act):** Determina si el robot puede seleccionar un nuevo objetivo.
- **rule_canPick(act):** Evalúa si el robot puede recoger un objeto.
- **rule_canStack(act):** Valida si el robot puede apilar un objeto.
- **rule_canMove(act):** Verifica si el robot puede moverse al próximo paso.
- **rule_nextHor(act):** Determina si el robot debería moverse horizontalmente.
- **rule_nextVer(act):** Determina si el robot debería moverse horizontalmente.
- **rule_goAroundHorizontal(act):** Gestiona desvíos horizontales.
- **rule_goAroundVertical(act):** Gestiona desvíos verticales.
- **rule_idle(act):** Decide si el robot debería quedarse idle.

RobotDino

State Description:

- **robotID (int):** Identificador único para el robot (3 en este caso)
- **state (int):** Estado actual del robot (0 = chasing, 1 = carrying, 2 = idle)
- **position (tuple):** Coordenadas actuales (x,y) del robot
- **stackAmount (int):** Número de objetos actualmente "stackeados" por el robot. Range None - 5
- **currentStack (tuple):** Tupla de la coordenada (x,y) del stack con el que el robot está interactuando (o None).
- path (list): El camino planeado por el robot, coordenadas (x,y).
- **lastGoAround (str):** Trackea si el robot se movió Horizontalmente (H) ó Verticalmente (V) para evitar una colisión.
- **targetObjectType (int):** El tipo de objeto target para el robot (en este caso 3).
- **board (list):** Lista del tablero, para la percepción del robot.
- **done (bool):** Trackea si el robot ha completado sus tasks.
- **recorded_objects_handled (int):** Número de objetos que se stackearon.
- **recorded_movements_made (int):** Número de movimientos realizados por el robot.
- **recorded_steps_idle (int):** Total de steps en los que el robot estuvo detenido.
- **recorded_avoided_collisions (int):** Número de veces que el robot evitó una colisión.
- **recorded_utility (int):** Número de pick-ups exitosos del robot.

Actions:

- **setTarget():** Settea el target del robot (objeto o stack). Elimina el target de la lista de objetos disponibles cuando ya se setteó.
- **pickUp():** Recoge un objeto si coincide con el tipo del robot.
- **stack():** Apila el objeto en un stack. Reinicia currentStack cuando el stack completa 5 objetos.
- **moveTo():** Mueve al robot un tile hacia su target.
- **idle():** Mantiene al robot como idle y aumenta recorded_steps_idle.
- **setNextHorTile():** Determina el siguiente tile horizontal según el path planeado.
- **setNextVerticalTile():** Determina el próximo vertical tile.
- **goAroundHorizontal():** Maneja desvíos horizontales en caso de bloqueo.
- **goAroundVertical():** Maneja desvíos verticales en caso de bloqueo.

Methods:

- **setup():** Inicializa las variables y el robot state al comenzar la simulación.
- **step():** Ejecuta el ciclo de razonamiento y acción (percepción → razonamiento → ejecución).
- **see(e):** Percibe el entorno actual (boardTiles, por ejemplo).
- **next():** Evalúa las reglas y elige la siguiente acción.
- **action(act):** Ejecuta la acción seleccionada.
- **findPath():** Calcula la ruta hacia el objetivo actual.

Rules:

- **rule_canSetTarget(act):** Determina si el robot puede seleccionar un nuevo objetivo.
- **rule_canPick(act):** Evalúa si el robot puede recoger un objeto.
- **rule_canStack(act):** Valida si el robot puede apilar un objeto.
- **rule_canMove(act):** Verifica si el robot puede moverse al próximo paso.
- **rule_nextHor(act):** Determina si el robot debería moverse horizontalmente.
- **rule_nextVer(act):** Determina si el robot debería moverse horizontalmente.
- **rule_goAroundHorizontal(act):** Gestiona desvíos horizontales.
- **rule_goAroundVertical(act):** Gestiona desvíos verticales.
- **rule_idle(act):** Decide si el robot debería quedarse idle.

RobotCarro

State Description:

• **robotID (int):** Identificador único para el robot (4 en este caso)
• **state (int):** Estado actual del robot (0 = chasing, 1 = carrying, 2 = idle)
• **position (tuple):** Coordenadas actuales (x,y) del robot
• **stackAmount (int):** Número de objetos actualmente "stackeados" por el robot. Range None - 5
• **currentStack (tuple):** Tupla de la coordenada (x,y) del stack con el que el robot está interactuando (o None).
• path (list): El camino planeado por el robot, coordenadas (x,y).
• **lastGoAround (str):** Trackea si el robot se movió Horizontalmente (H) ó Verticalmente (V) para evitar una colisión.
• **targetObjectType (int):** El tipo de objeto target para el robot (en este caso 4).
• **board (list):** Lista del tablero, para la percepción del robot.
• **done (bool):** Trackea si el robot ha completado sus tasks.
• **recorded_objects_handled (int):** Número de objetos que se stackearon.
• **recorded_movements_made (int):** Número de movimientos realizados por el robot.
• **recorded_steps_idle (int):** Total de steps en los que el robot estuvo detenido.
• **recorded_avoided_collisions (int):** Número de veces que el robot evitó una colisión.
• **recorded_utility (int):** Número de pick-ups exitosos del robot.

Actions:

• **setTarget():** Settea el target del robot (objeto o stack). Elimina el target de la lista de objetos disponibles cuando ya se setteó.
• **pickUp():** Recoge un objeto si coincide con el tipo del robot.
• **stack():** Apila el objeto en un stack. Reinicia currentStack cuando el stack completa 5 objetos.
• **moveTo():** Mueve al robot un tile hacia su target.
• **idle():** Mantiene al robot como idle y aumenta recorded_steps_idle.
• **setNextHorTile():** Determina el siguiente tile horizontal según el path planeado.
• **setNextVerticalTile():** Determina el próximo vertical tile.
• **goAroundHorizontal():** Maneja desvíos horizontales en caso de bloqueo.
• **goAroundVertical():** Maneja desvíos verticales en caso de bloqueo.

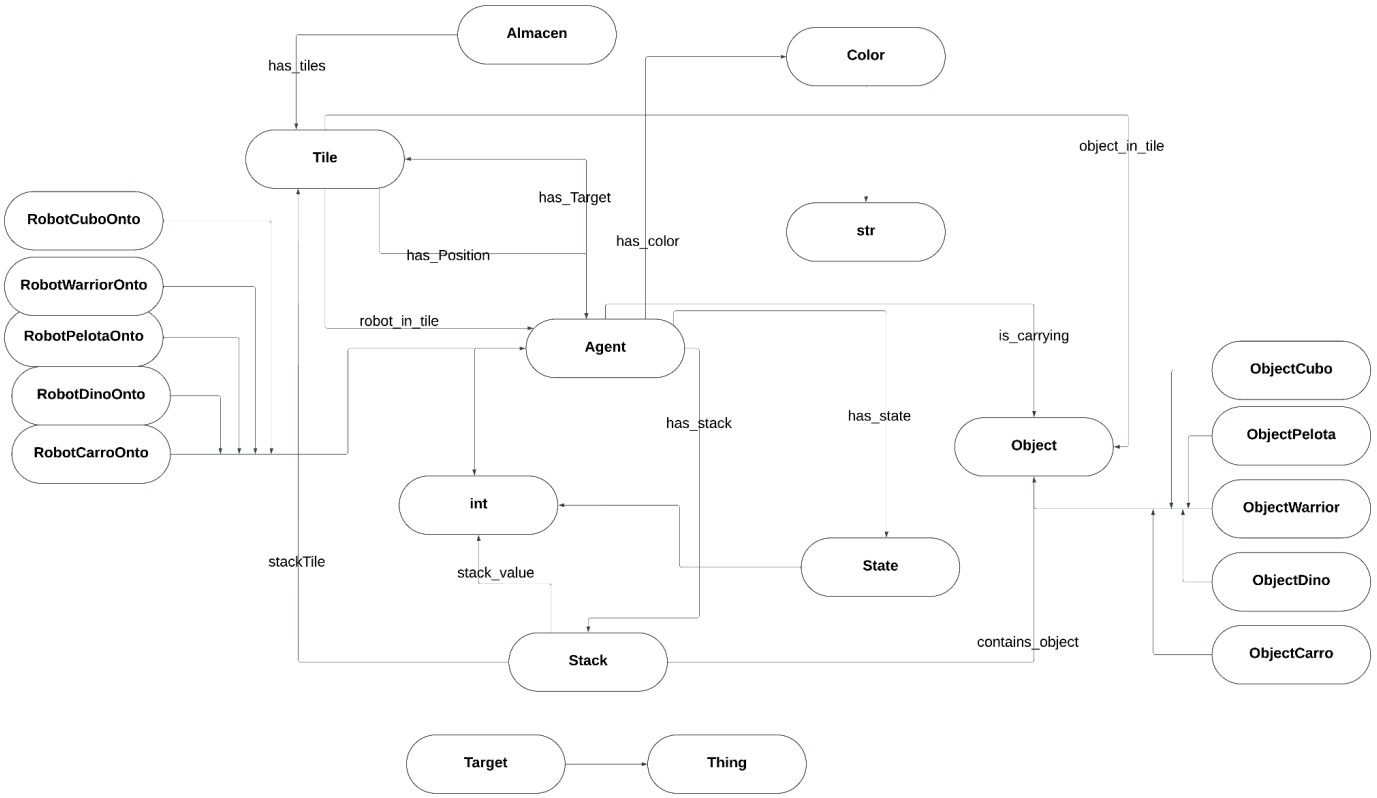
Methods:

• **setup():** Inicializa las variables y el robot state al comenzar la simulación.
• **step():** Ejecuta el ciclo de razonamiento y acción (percepción → razonamiento → ejecución).
• **see(e):** Percibe el entorno actual (boardTiles, por ejemplo).
• **next():** Evalúa las reglas y elige la siguiente acción.
• **action(act):** Ejecuta la acción seleccionada.
• **findPath():** Calcula la ruta hacia el objetivo actual.

Rules:

• **rule_canSetTarget(act):** Determina si el robot puede seleccionar un nuevo objetivo.
• **rule_canPick(act):** Evalúa si el robot puede recoger un objeto.
• **rule_canStack(act):** Valida si el robot puede apilar un objeto.
• **rule_canMove(act):** Verifica si el robot puede moverse al próximo paso.
• **rule_nextHor(act):** Determina si el robot debería moverse horizontalmente.
• **rule_nextVer(act):** Determina si el robot debería moverse horizontalmente.
• **rule_goAroundHorizontal(act):** Gestiona desvíos horizontales.
• **rule_goAroundVertical(act):** Gestiona desvíos verticales.
• **rule_idle(act):** Decide si el robot debería quedarse idle.

Diagrama de ontología



Conclusión

El sistema multiagentes fue completado con éxito. Se implementó un ambiente con las características necesarias para proveer a los agentes con información relevante y actualizada en tiempo real. Los agentes fueron implementados utilizando propiedades adecuadas, razonamiento deductivo, y con la utilización de una ontología que permite la comunicación y entendimiento completo de la información. Los agentes tuvieron las acciones necesarias para pensar, planear y reaccionar, así como para moverse, recoger objetos, evitar colisiones, acomodar objetos e identificar los tipos de objetos.

En cuanto a la simulación completa, los agentes suelen terminar sus tareas con éxito en la gran mayoría de los casos, excepto por casos muy extremos en donde el número de steps máximo no es suficiente. Los robots interactúan entre ellos y con el ambiente de manera adecuada, hasta terminar sus tareas, dejando el almacén limpio, sin importar las posiciones iniciales.

Consideramos que los resultados fueron los esperados, aunque tenemos claras algunas posibles mejoras para aumentar la eficiencia de los robots. La primera oportunidad de mejora es que cuando un robot se atora entre más de cuatro objetos, el robot espera a que algunos de esos objetos sea recogido para continuar su camino (solamente rodean máximo tres objetos juntos). Esto provoca que dicho robot se mantenga quieto por varios steps hasta que la situación cambie. Otra

mejora, es que cuando un robot está a punto de moverse a una casilla ocupada por otro robot, este se espera a que se desocupe, en lugar de rodearlo. Esto provoca que si el robot que estorba se queda quieto, el otro lo esperará indefinidamente. Estos son casos muy extremos que no suelen darse en la corrida promedio, pero sería importante mejorar a los robots en una situación real para evitar cualquier conflicto.