Tecnológico de Monterrey

Integration of Computer Security in Networks and Software Systems (Gpo 401)

Integrative Activity - Multiagent Systems with CG

Juan Pablo Buenrostro | A01645981

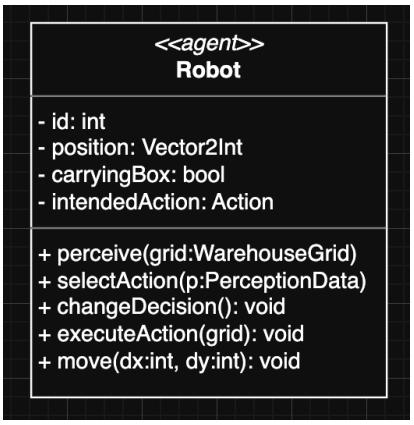
Joaquín Hiroki Campos Kishi | A01639134

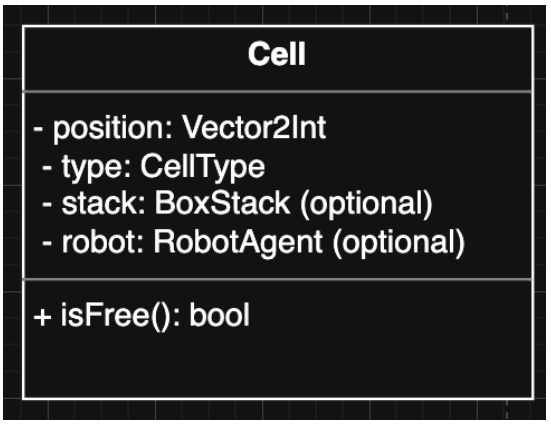
Esteban Camilo Muñóz Rosero | A01644609

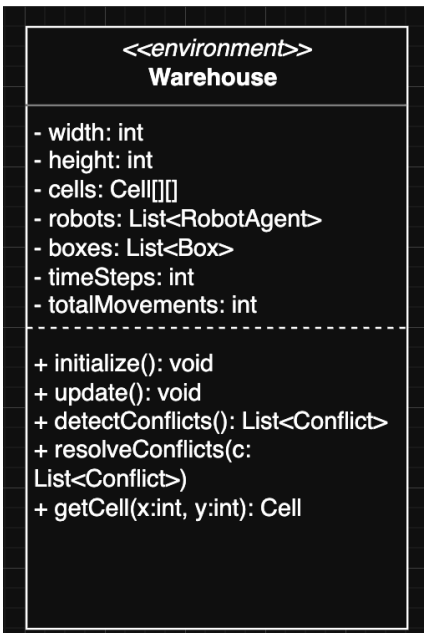
Fernanda Yaltzin Jiménez Ramos | A01645485

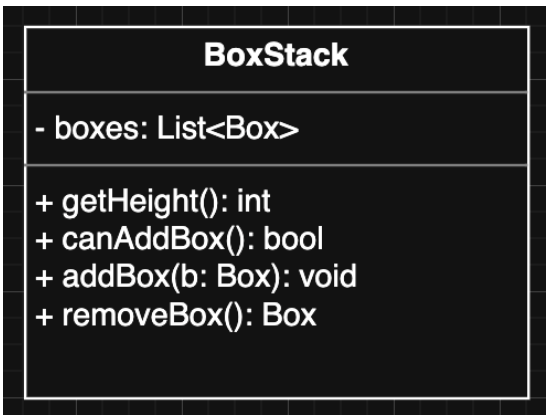
November 24, 2025.

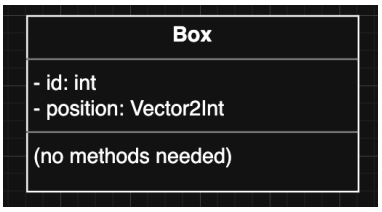
Class Diagrams

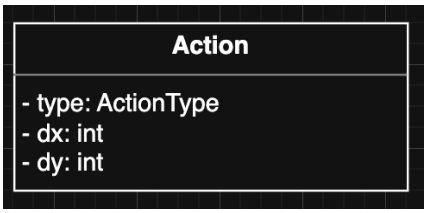




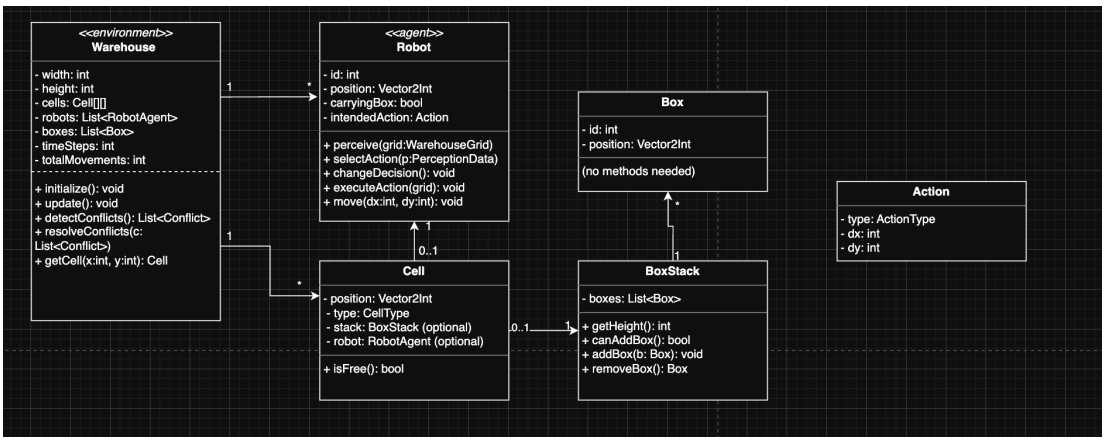




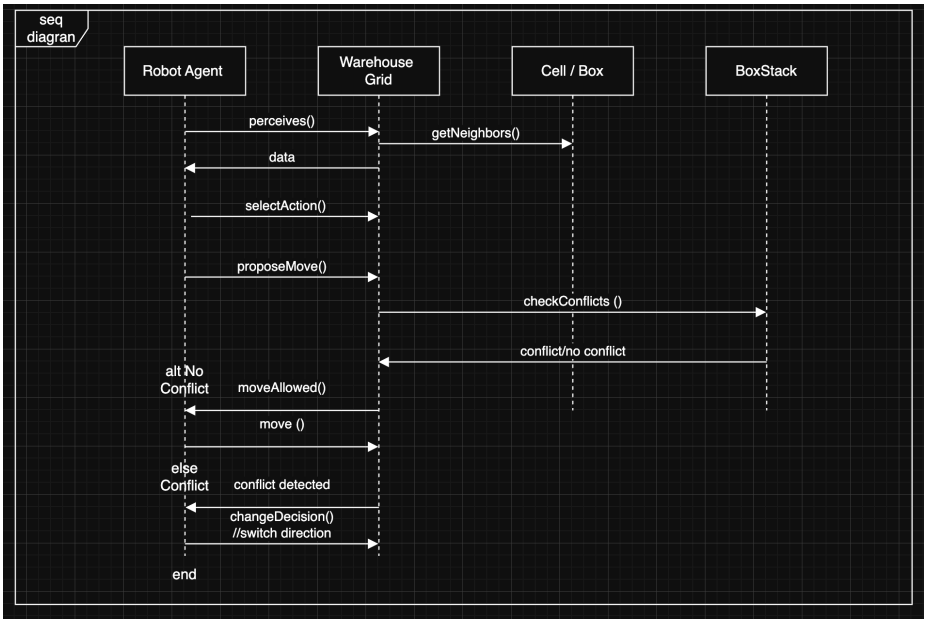




UML



Sequence Diagram



Agent Communication Protocol

Robots operate without a central controller and without explicit message–passing. Each robot decides its actions independently using only local sensing and a simple conflict–resolution rule. Coordination emerges from these local interactions.

Local State Information

- **POSITION**: Current grid coordinates of the robot.
- **ADJACENT_CELLS**: Sensor readings for the four neighboring cells, indicating whether each one is:
    - free
    - a wall
    - occupied by another robot
    - containing a box stack (including its current height).
- **CARRYING_STATUS**: Boolean value given by a pressure sensor that indicates whether the robot is currently carrying a box.
- **INTENDED_ACTION**: Action selected for the current simulation step (move, pick, drop, or wait).

Action Types

- **MOVE(direction)**: Attempt to move one cell up, down, left, or right.
- **PICK_BOX**: Pick up a box from an adjacent cell if a loose box or a non-empty stack is available.
- **DROP_BOX**: Place a carried box on the current stack or create a new stack, respecting the maximum height of 5 boxes.
- **WAIT**: Stay in the current cell for one simulation step.

Conflict Types

- **SAME_CELL_CONFLICT**: Two or more robots attempt to move into the **same target cell** at the same time.
- **SWAP_CONFLICT**: Two robots attempt to **swap positions simultaneously** (robot A moves into B's cell while B moves into A's).

Conflict Resolution Rule

When a conflict is detected, all involved robots apply the **changeDecision()** rule:

- The movement for the current step is canceled (robots remain in their original cells).
- Each robot selects an alternative action, for example:
    - choose a different MOVE direction that is free, or
    - WAIT for one step to give way to others.

This reactive rule prevents collisions, reduces deadlocks in narrow aisles, and keeps the system fully decentralized.

Execution Cycle

For each simulation step, every robot executes the following loop:

1. **PERCEIVE**: Read sensors and update local state (POSITION, ADJACENT_CELLS, CARRYING_STATUS).

2. **SELECT_ACTION**: Choose an INTENDED_ACTION based on its local goal (e.g., search for boxes, move toward stacking area, drop box).

3. **CHECK_CONFLICTS**: The simulation engine aggregates all INTENDED_ACTIONs and identifies SAME_CELL_CONFLICT and SWAP_CONFLICT cases.

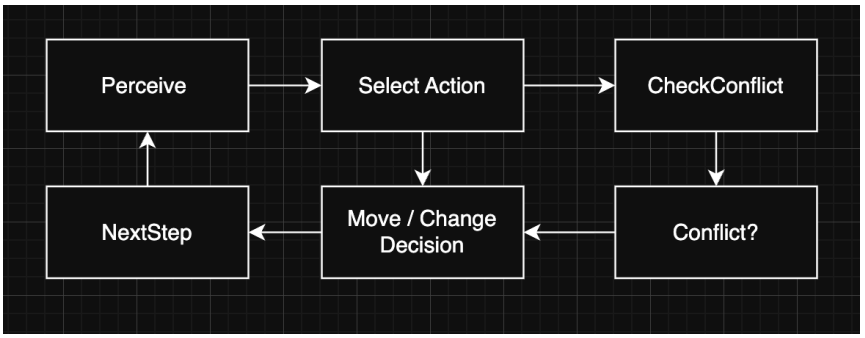
4. **RESOLVE**:

    - If no conflict exists for a robot, it keeps its INTENDED_ACTION.

    - If a conflict exists, the robot calls **changeDecision()** and updates its action (often to WAIT or to a different direction).

5. **EXECUTE**: Robots perform the final action for this step (movement, pick, drop, or wait).

Robots therefore interact only through movement and local sensing. Cooperation does not require explicit communication channels or priorities.



Cooperative Strategy for Solving the Problem

The cooperative strategy combines spatial decomposition of the warehouse with simple local rules:

- The warehouse grid is divided into zones, and each robot is mainly responsible for the boxes located inside its zone.

- Within its zone, a robot:

    1. Searches for loose boxes or partially filled stacks.

    2. Uses **PICK_BOX** to collect boxes.

3. Transports them to a **shared stacking area** where stacks of up to 5 boxes are built in an ordered way.

- Because each robot focuses on its own zone, the number of encounters and potential conflicts is reduced. At the same time, the shared stacking area guarantees that all boxes end up organized in a small, centralized region.

Global cooperation emerges from:

- **Local autonomy**: Each robot plans only with local information (sensors and its zone).
- **Conflict handling**: When two robots try to enter the same cell or swap positions, the conflict resolution rule forces them to cancel the move and apply **changeDecision()**, which typically results in waiting or choosing an alternative path.
- **Task distribution**: Zoning keeps travel distances shorter and avoids congestion in the aisles.

Over time, these mechanisms keep the traffic flow in the warehouse relatively smooth, reduce idle time caused by deadlocks, and allow all robots to contribute to collecting and stacking boxes until the warehouse is fully organized into stacks of at most five boxes.

Simulation Results and Analysis

The simulation was executed in a warehouse of size 20 × 10, with 30 initial boxes randomly distributed and 5 autonomous robots starting from random empty cells. The system was run with a maximum execution limit of 3000 steps. Across multiple runs, the following results were observed:

```
Python

Steps: 3000

Moves: 180–250

Organized: False
```

Although the robots successfully picked up and transported several boxes to the DROP row, the system did not manage to organize all boxes within the given time limit. This outcome is expected given the decentralized nature of the agents and the simplicity of the local decision-making rules.

Reasons for Incomplete Organization

Several factors contribute to the system not reaching full organization:

1. Strict Completion Criterion

The simulation only counts as “organized” when every box has been placed in the DROP row. If a single box remains elsewhere in the warehouse, the condition remains False. This makes the requirement very strict and difficult to meet within the time limit.

2. Zone-Based Searching

Each robot is restricted to searching for boxes only within its assigned horizontal zone. If a box is pushed or moved outside this zone (due to collisions, detours, or random directional changes), the responsible robot will no longer retrieve it, and robots from other zones will ignore it.

3. Random Conflict Resolution

When two robots detect a conflict (same-cell or swap), they apply a random movement via changeDecision(). While this prevents collisions, it also causes; inefficient movements,repeated wandering,boxes being pushed across zones,occasional deadlocks or loops in tight aisles.

These factors collectively reduce the system’s efficiency and prevent full completion within 3000 steps.

Potential Strategy for Improvement

A more sophisticated strategy could significantly reduce the time and number of movements:

1. Dynamic Zone Boundaries

Zones could expand or shrink depending on whether a robot has more or fewer boxes left in its area.

2. Cross-Zone Assistance

Robots could be allowed to pick up boxes from adjacent zones once they finish their own tasks, reducing the number of “orphan boxes”.

3. Priority-Based Conflict Resolution

Instead of random detours, robots could use:

ID–based priority (e.g., lower ID yields), or distance-based priority (robot closer to the target proceeds). This would reduce random wandering and unnecessary collisions.

4. Improved Navigation

Using a simple pathfinding technique (e.g., greedy or BFS) instead of pure Manhattan-direction movement would allow robots to reach boxes and DROP areas more directly.