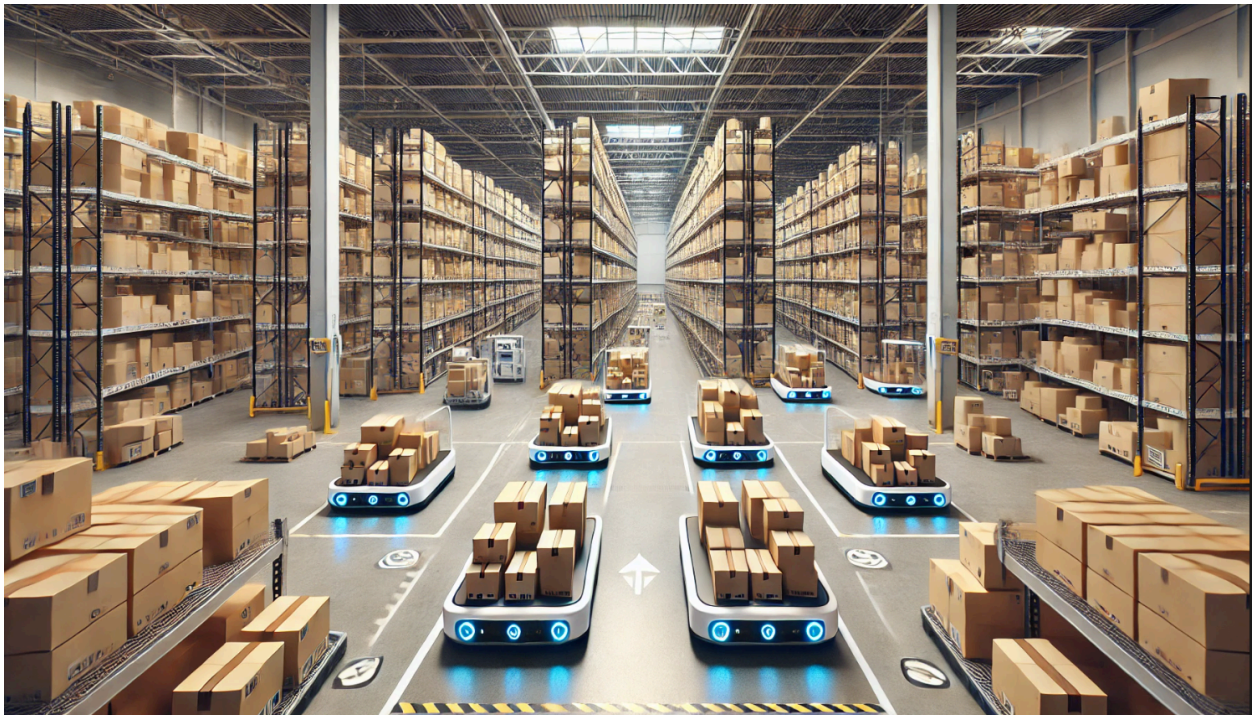
DoseHACK '24

Problem Statement



The Warehouse Autobots Dilemma

Dosepacker Inc. runs one of the largest automated warehouses, where robots (called **autobots**) move products from one place to another. These autobots work on a grid, following tracks to get from **Point A** (where they pick up items) to **Point B** (where they deliver items). However, recently the system has started facing problems like **traffic jams and collisions**, slowing down the entire operation.

To solve this issue, Dosepacker Inc. is asking for your help! Your task is to create a smart system that will control these autobots, helping them move around without bumping into each other or obstacles.

The goal is to build a solution where autobots can move safely and efficiently within the warehouse.

The Mission:

As a participant, your challenge is to:

- Guide multiple autobots across a **matrix grid** from their **starting point** (A) to their **destination** (B), avoiding obstacles and each other.
- Autobots can only understand simple movement commands: `Forward`, `Reverse`, `Left`, `Right`, and `Wait`.
- You must make sure that all autobots reach their destination without any collisions and without blocking each other.

Key Challenges:

1. **Collision Avoidance**:
    - With multiple autobots moving simultaneously on the grid, no two bots can occupy the same cell at any time. You must ensure smooth navigation while avoiding deadlocks and collisions.
2. **Dynamic Movement**:
    - Autobots execute only **one command at a time**. If a bot needs to pause, the `Wait()` command will pause the bot for **1 unit of time**. To pause longer, multiple `Wait()` commands are needed.
3. **Matrix Layout**:

- The warehouse is represented by a grid (matrix) of varying sizes, containing passable cells and blocked cells (obstacles).
- The grid layout changes with each level of difficulty, presenting more obstacles and complex paths.

4. **Real-Time Coordination**:
   - Autobots must work in parallel, coordinating their movements dynamically to avoid blocking each other. Efficient solutions will minimize delays while ensuring safety.

Input Format and Grid Layout:

The warehouse grid is represented as a **2D matrix** where:

- **Empty cells** are passable by autobots (denoted as `.`).
- **Blocked cells** represent obstacles (denoted as `X`).
- Each autobot starts from a **specified point (A)** and must reach a designated **destination (B)**.

Here's an example of how a 5x5 grid might look:


```
A . . X B
. X . . .
. . X . .
. . . . .
A X . . B
```

`A`: Starting points for the autobots.
`B`: Destination points for the autobots.
`X`: Obstacles (cannot be passed).
`.`: Free cells where autobots can move.

Command Set for Bot Movement:

Each autobot starts facing **upward** and can execute the following commands:

- **Forward()**: Moves the bot one position ahead in the direction it's facing.
- **Reverse()**: Moves the bot one position backward.
- **Left()**: Turns the bot 90 degrees to the left (without moving).
- **Right()**: Turns the bot 90 degrees to the right (without moving).
- **Wait()**: Pauses the bot for 1 unit of time. Multiple `Wait()` commands are needed for longer pauses.

For example, the movement for two bots might look like this:

```
Unset
car1: (forward, forward, wait, forward, left, forward, forward)car2: (reverse, right, forward, forward, forward, forward, left, forward)
```

- **Forward()** and **Reverse()** move the bot **1 position**.
- **Left()** and **Right()** turn the bot in place. All bots start **facing upward** by default.

Simulating Movement and Commands:

If you choose **not to use a GUI** for simulation, you must print:

- The **initial position** of each bot on the grid.
- A **step-by-step command list** for each bot, showing every movement.
- The **final position** of each bot at the end of the simulation.

For example, after executing the command set:

```
Unset
car1: (forward, forward, wait, forward, left, forward, forward)
```

You should display the bot's movements in this format:

- **Initial position** of car1.

- Each movement step (e.g., "car1 moves forward").
- **Final position** of car1 once all commands have been executed.

Performance Metrics:

For each simulation, you must calculate:

1. **Total Time**:
    - The total time taken for all autobots to complete their journeys from **Point A** to **Point B**.
    - Each command, including `Wait()`, consumes **1 unit of time**.
2. **Total Commands**:
    - The total number of commands issued to each bot (forward, reverse, wait, etc.).
    - Aim to minimize the number of unnecessary commands and waits.

Note: These metrics have to be submitted at the end of the hackathon.

Additional Requirements:

1. **Collision-Free Execution**:
    - No two bots should occupy the same cell at the same time, ensuring collision-free movement.
2. **Handling Waits**:
    - Bots should intelligently use the `Wait()` command to avoid collisions and deadlocks, especially in tight areas.
3. **Parallel Movement**:
    - Multiple bots can move at the same time, but they must avoid crossing paths or blocking each other.

Evaluation Criteria:

1. **Collision-Free Movement (30%)**:
    - Your solution must ensure that autobots navigate the grid without crashing into obstacles or each other.

2. **Simulation & Real-Time Visualization (20%)**:
    - If using a **GUI**, it should display the autobots' movements on the grid in real-time.
    - If not using a GUI, you must print the movement and command sequence of each bot.
3. **Efficiency of Solution (30%)**:
    - Solutions will be judged based on how efficiently autobots reach their destination, considering the total time taken and the number of commands issued.
4. **Intelligent Decision-Making (20%)**:
    - Solutions that implement **AI techniques** like **reinforcement learning** will earn bonus points for enabling bots to make dynamic, intelligent decisions in real-time.

Submission Guidelines

To ensure a smooth and organized submission process, each team must follow the guidelines below. Your final deliverables will showcase both your solution's functionality and your approach. These submissions are essential for evaluation and must be completed **30 minutes before the final deadline (Except 1:Should start immediately)**.

1. GitHub Repository

Join the GitHub Organization: All team members must join the GitHub organization at DoseHACK-24.

Create a Team: Form your team within the organization and add only your team members.

Create a Repository: Each team must create a repository named after their team and assign access to their team members.

Regular Code Updates: Code should be pushed to the repository **at regular intervals of 5 hours** to ensure consistent progress tracking.

The repository must include:

- All source code.
- A **README** explaining how to run the code, project structure, and any dependencies.
- Logs of commands issued to the autobots for each movement.
-

Make sure that your code is well-documented and organized to allow easy testing and evaluation by the judges.

2. 5-Minute Video Presentation

- Teams must also create a **5-minute video** that explains their code, strategy, and shows a **live demonstration** of their solution running against the **final test case** provided 3 hours before the deadline.
- The video should cover:
    - An overview of the approach and decision-making process (AI techniques used, such as reinforcement learning, if applicable).
    - How the bots avoid collisions and deadlocks.
    - A demonstration of the system working on the final test case grid, showing the solution in action.
- The video must be uploaded to **YouTube** as an **Unlisted Video** (so only those with the link can view it).

3. Google Form Submission

- **30 minutes before the deadline**, a Google Form will be circulated to all teams.
- Each team must submit the following information in the Google Form:
    1. The **link to your public GitHub repository**.
    2. The **link to your unlisted YouTube video** demonstrating the solution.
    3. The **total number of commands** issued for each bot in the final test case.
    4. The **overall run time** of your solution for the final test case.

Teams should ensure that all submissions are completed and submitted via the Google Form by the 30-minute mark. Late submissions or missing information may affect your evaluation.

The Future of Dosepacker Inc. is in Your Hands:

The future of **Dosepacker Inc.'s Smart Warehouse** depends on your ability to create a robust, intelligent system for autobot navigation. Your solution could revolutionize automated logistics and optimize how autonomous systems function in complex environments.

Will you and your team be the one to solve the autobot navigation challenge and bring Dosepacker’s warehouse back to peak efficiency?