CS - 520

Introduction to Artificial Intelligence

Project 1

Group:

1. **Reo Correia** (rc1460)
2. **Manan Davawala** (md1840)
3. **Vijayendra Sai Chennareddy** (vc545)

Problem Statement

Understanding the Problem

The challenges that Bot faces:

- Grid-Based Environment: The ship's layout is represented as a grid, where each cell can be either open or blocked (walls). The bot's movements are constrained to up, down, left, and right, reflecting the grid's structure.
- Hostile Aliens: The presence of hostile aliens poses a significant threat to the bot's mission. Aliens move within the grid, and the bot must avoid collisions with them to ensure the safety of the crew members.
- Crew Member Rescue: The primary objective is to rescue crew members who are randomly placed in open cells. The bot must find the shortest path to reach the crew members while adapting to dynamic alien movements.

Setup

The grid size of **50 X 50** has been used for the ship layout.
For testing the Bot strategies, we have varied the number of aliens from **5 to 50 in increments of 5**.
Each alien value run is averaged over **100 runs**.
Each simulation has an upper limit of **1000 steps**.

Modeling and Solving the Problem

Comparing Navigation Algorithms :

Let's examine why the A* algorithm is superior to BFS, DFS, and Dijkstra's algorithm for Bot''s mission:

1. Breadth-First Search (BFS): BFS explores paths layer by layer, visiting all neighboring nodes before moving to the next level. While BFS guarantees the shortest path, it may not be computationally efficient in this context due to the dynamic nature of alien movements. BFS does not consider heuristics and cannot adapt to changes in the environment.
2. Depth-First Search (DFS): DFS explores paths deeply before backtracking. While it may find a path quickly in certain scenarios, it does not guarantee the shortest path. In a dynamic environment with hostile aliens, DFS is not well-suited as it may lead to inefficient and risky paths.
3. Dijkstra's Algorithm: Dijkstra's algorithm guarantees the shortest path, but it explores nodes uniformly without considering heuristics. It can be computationally intensive and less adaptive to dynamic environments. In the presence of aliens, Dijkstra's algorithm can be less efficient and fail to adapt quickly.
4. A* Algorithm:
    - Pros: The A* algorithm combines the advantages of other algorithms while addressing their limitations.
    - Admissibility: A* guarantees finding the shortest path when using an admissible heuristic.
    - Informed Search: A* uses a heuristic to prioritize the most promising paths, which is crucial for efficient pathfinding in the presence of obstacles and aliens.
    - Dynamic Adaptation: The A* algorithm can adapt to changes in the alien positions as it reevaluates the path at each step.
    - Optimal and Efficient: A* strikes a balance between optimality and efficiency, making it ideal for real-time missions.

Heuristic Choice: Manhattan Distance

The Manhattan distance heuristic, also known as the L1 norm, measures the distance between two points in a grid-based environment. In the context of Bot, the Manhattan distance heuristic provides a good estimate of the distance between the bot's current position and the crew member's location . While this heuristic does not consider blocked cells and aliens, it is computationally efficient and generally results in reasonable pathfinding.

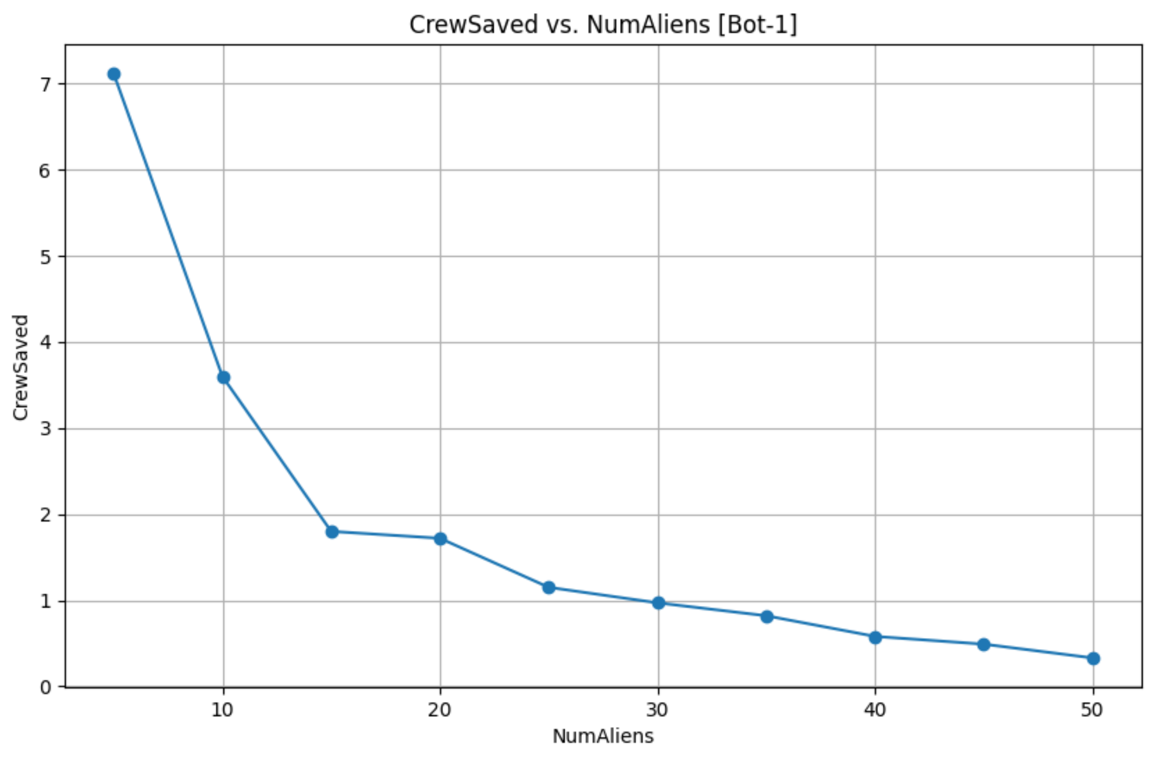
Why Manhattan Distance Is Ideal

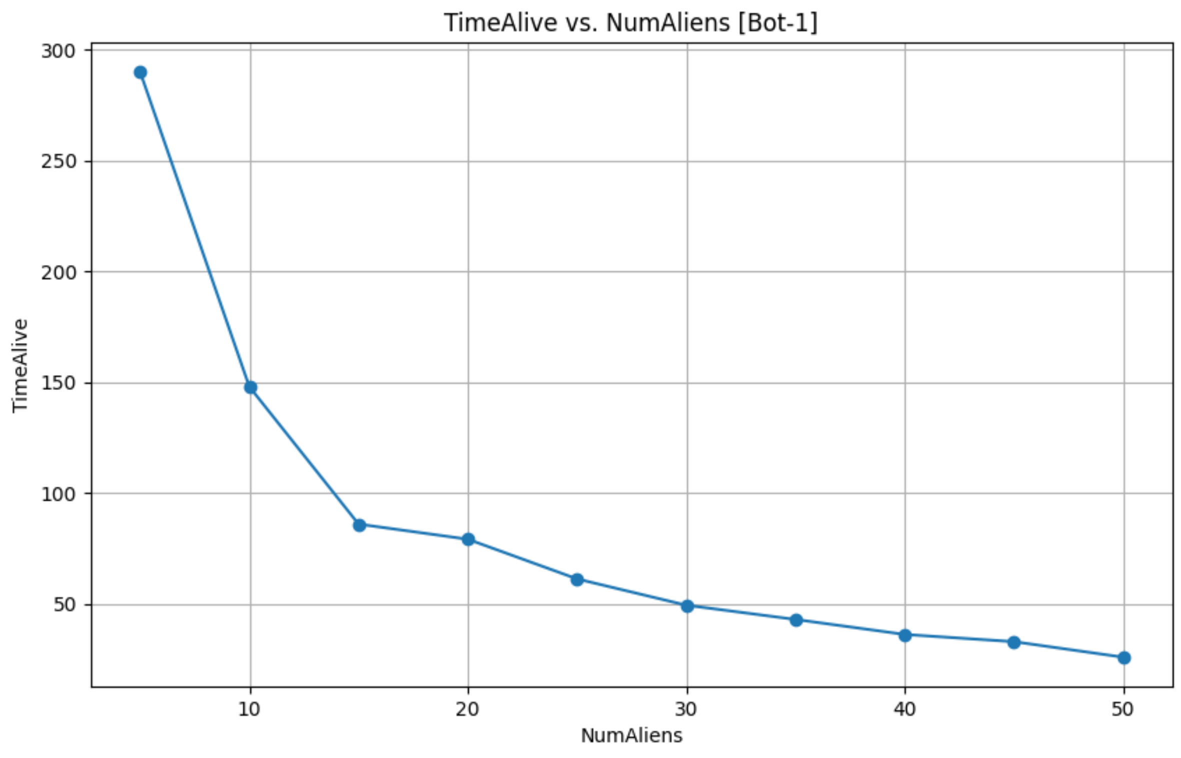
- Grid-Based Environments: The deep space vessel's layout is grid-based, and the bot's movements are limited to up, down, left, and right. In such environments, the Manhattan Distance heuristic is highly intuitive and aligns with the bot's permissible actions.
- Computational Efficiency: In the context of Bot, computational efficiency is crucial. The Manhattan Distance heuristic is computationally lightweight, making it well-suited for real-time applications. It provides a reliable estimate of the distance without complex calculations or excessive resource consumption.
- Adaptability: The bot's mission involves dynamic elements—the movement of aliens. The Manhattan Distance heuristic is adaptable because it recalculates the estimate at each step, allowing the bot to dynamically adjust its path based on the changing alien positions.
- Conservativeness: The heuristic is known for being admissible, which means it never overestimates the true cost to reach the goal. This property is essential for A* search algorithms like the one used by Bot 1, as it guarantees that the optimal path will be found.
- Suitable for Open Spaces: In the context of grid-based environments with relatively open spaces, the Manhattan Distance heuristic is highly effective. It provides a reasonable estimate of the path length, even when obstacles (aliens) are present but are not considered in the heuristic itself.

Analysis of Bot Strategies

1) Bot 1

Bot 1 implements the A* algorithm with the Manhattan distance as its heuristic function to determine the shortest path to the crew's location while avoiding any cells occupied by alien cells. The algorithm computes an optimal path that avoids alien cells. Once this path is computed, Bot 1 proceeds along this path without considering any intervening movements or changes in the positions of the alien cells that may occur along the way.





Failure:

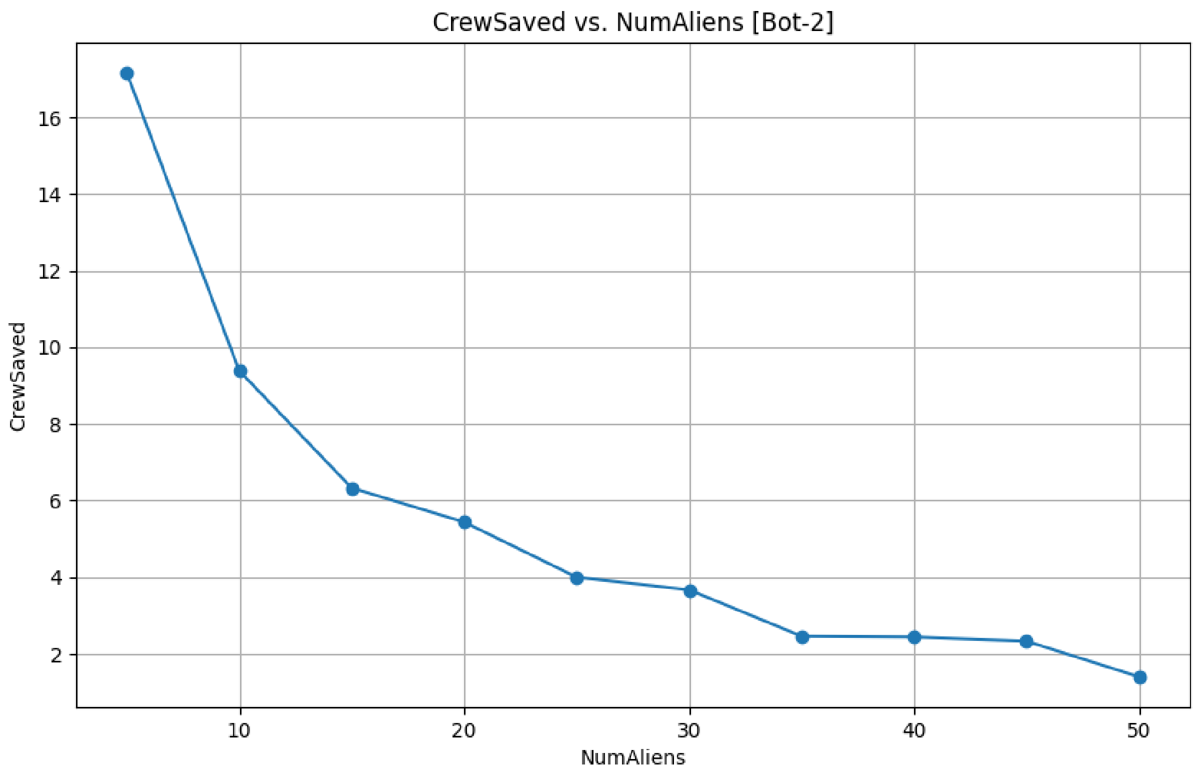
- Bot 1's ability to save crew members or survive for 1000 steps is compromised due to the inherent risk of encountering either an "alien cell" or an adjacent cell to an "alien cell."
- If bot 1 does not find a path to the crew's location. It stays in place indefinitely until it reaches 1000 steps or an alien kills it, essentially marking the end of simulation. Hence we kill the simulation in case no path exists.

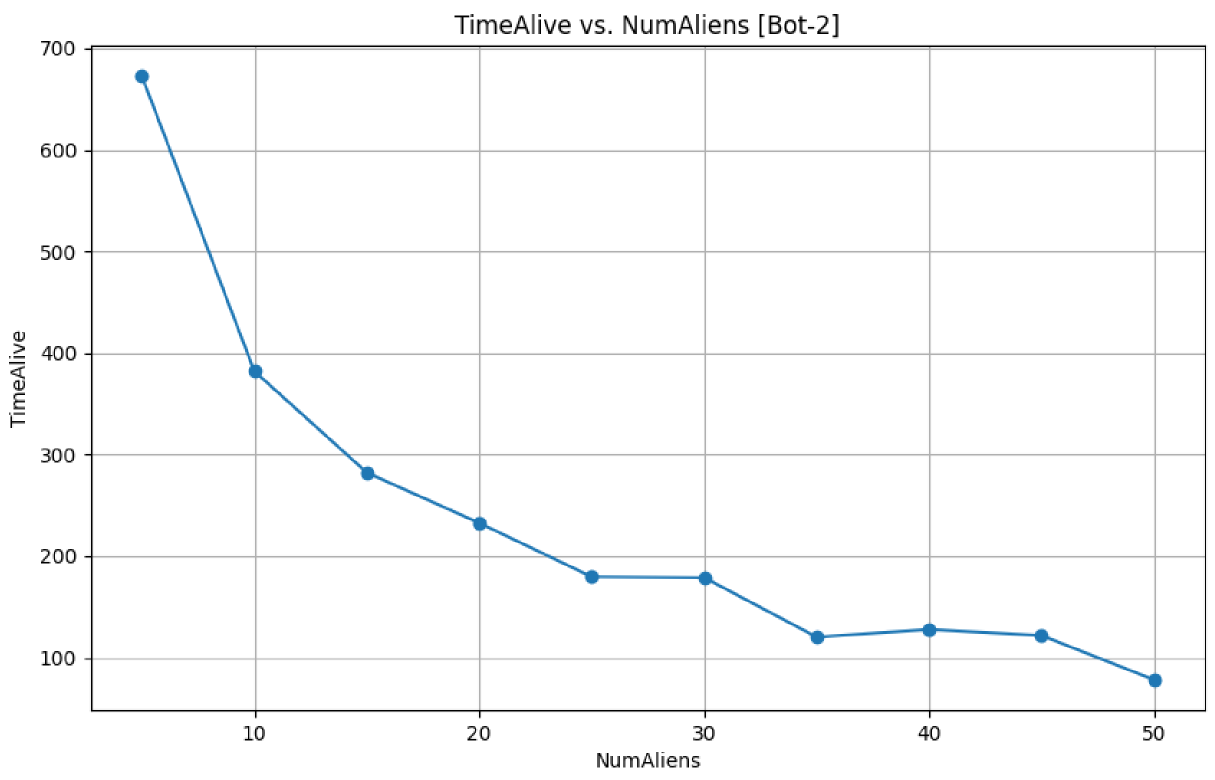
Improvement:

- Dynamically recalculating the shortest path avoiding the alien cell after each and every step. This method has been implemented in Bot 2.

2) <u>Bot 2</u>

Bot 2 implements the A* algorithm with the Manhattan distance as its heuristic function to continuously determine the shortest path to the crew's location while avoiding any cells occupied by aliens. Bot 2 does not simply ignore the movements of the alien. It dynamically recalculates the shortest path after each step. This allows Bot 2 to adapt to the changing positions of the aliens, ensuring a higher degree of survivability.





Failure:

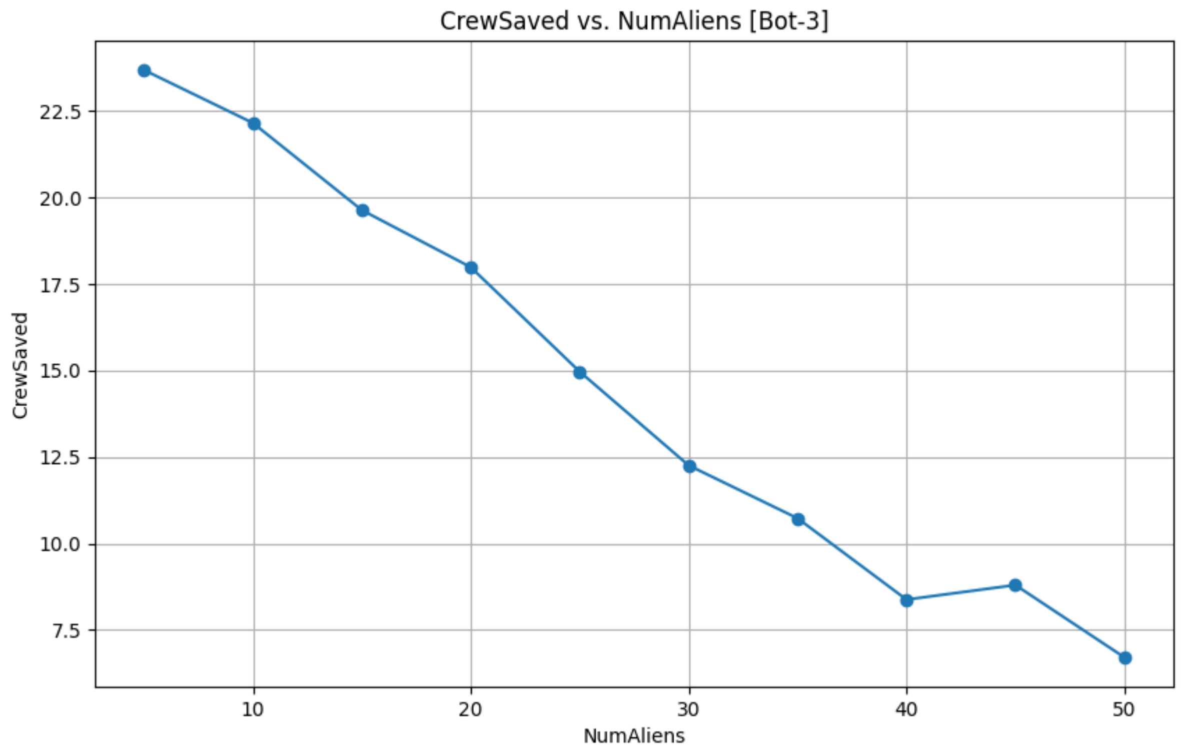
- Bot 2's ability to save crew members or survive for 1000 steps is compromised due to the inherent risk of encountering an alien by entering an adjacent cell to an "alien cell."

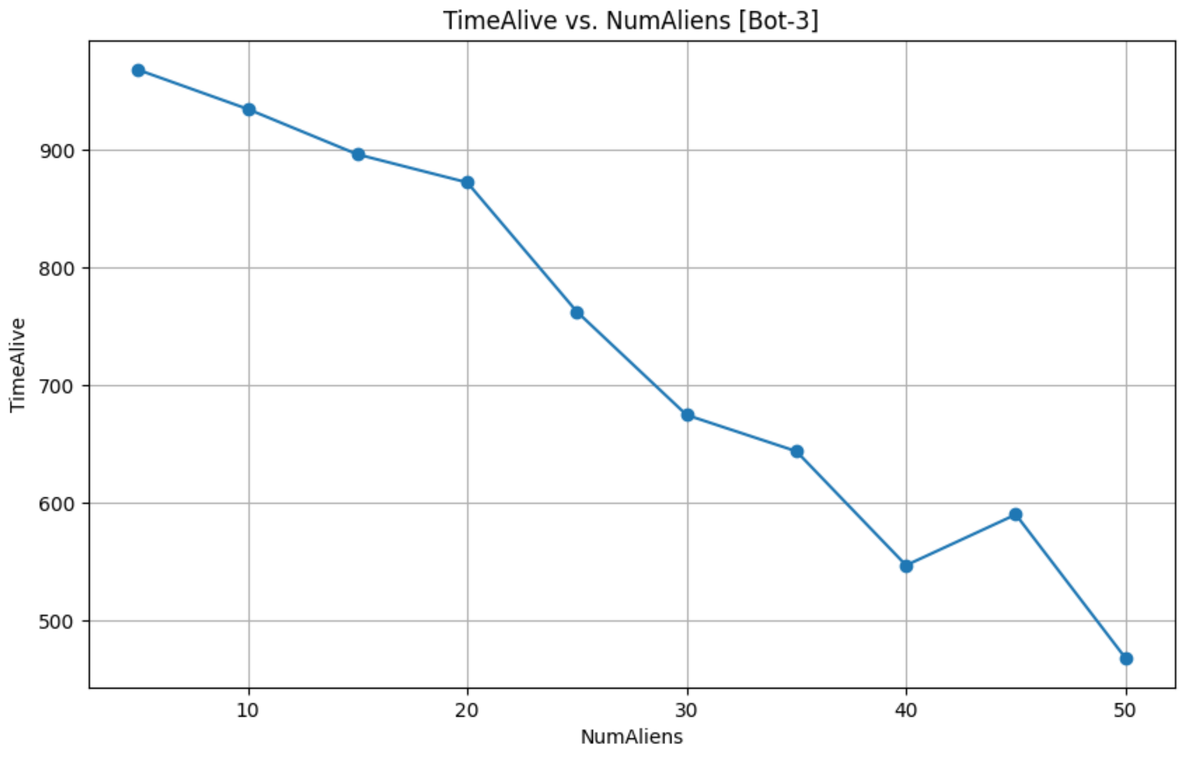
Improvement:

- The bot also actively tries to avoid the alien cells as well as its immediate neighbor.

3) **Bot 3**

Bot 3 implements the A* algorithm with the Manhattan distance as its heuristic function to continuously determine the shortest path to the crew's location while avoiding any cells occupied by aliens and their neighbors. If no such path exists, it recalculates the shortest path to the crew cell while avoiding the alien cells. Because of avoiding the neighbors the bot guarantees its chance of survival at the next step.





Why Bot 3 Fails to Perform Well?

Bot 3 employs two key decisions in its rescue strategy:

Decision 1: Avoiding the current alien positions and any cells adjacent to current alien positions.
Decision 2: Avoiding the current alien positions.

In Bot 3, it first attempts to find the shortest path to the crew member based on Decision 1. If there is no such path, it then plans the shortest path based on Decision 2.

➢ **Improvement Scenario:**

In a scenario where the bot finds a path to reach the crew member while avoiding cells adjacent to all aliens except for one alien, Bot 3 fails to generate any path using Decision 1 and falls back to Decision 2.

➢ **Proposed Improvement:**

- **Link** for the implementation of A* with the proposed heuristic: Bot 3 Improvement

To address this issue, Bot 3 should aim to find the shortest path to the crew member while avoiding the current alien positions and, if possible, avoid cells adjacent to aliens. To achieve this, we will modify our heuristic function to:
heuristic function = manhattan_distance(neighbor, crew) + no_of_aliens_adjacent(neighbor, alien_cells)

The new heuristic considers the number of aliens adjacent to a neighbor cell, aiding the bot in avoiding cells that are most densely populated by aliens. By considering the density of aliens around each cell, the bot can better navigate through areas heavily populated by aliens.

This approach ensures that Bot 3 makes the best possible choice in terms of avoiding aliens, whether following Decision 1 or Decision 2.

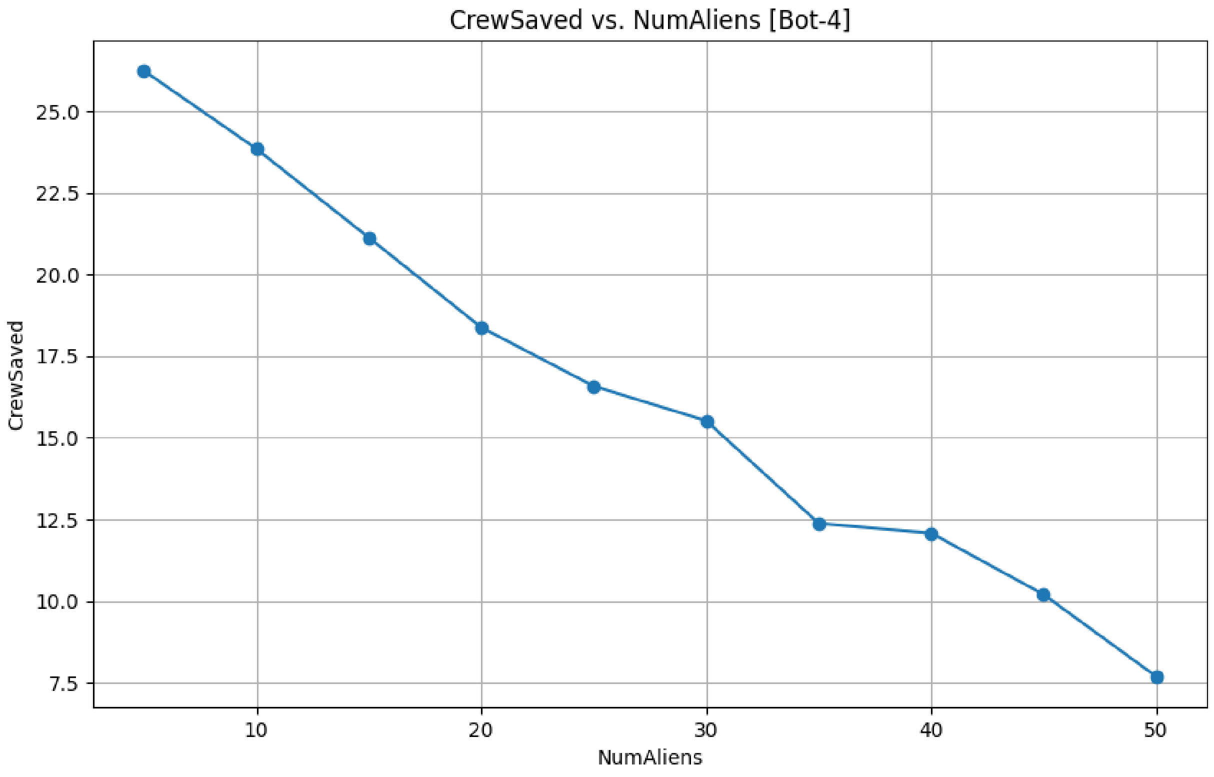
4) <u>Bot 4</u>

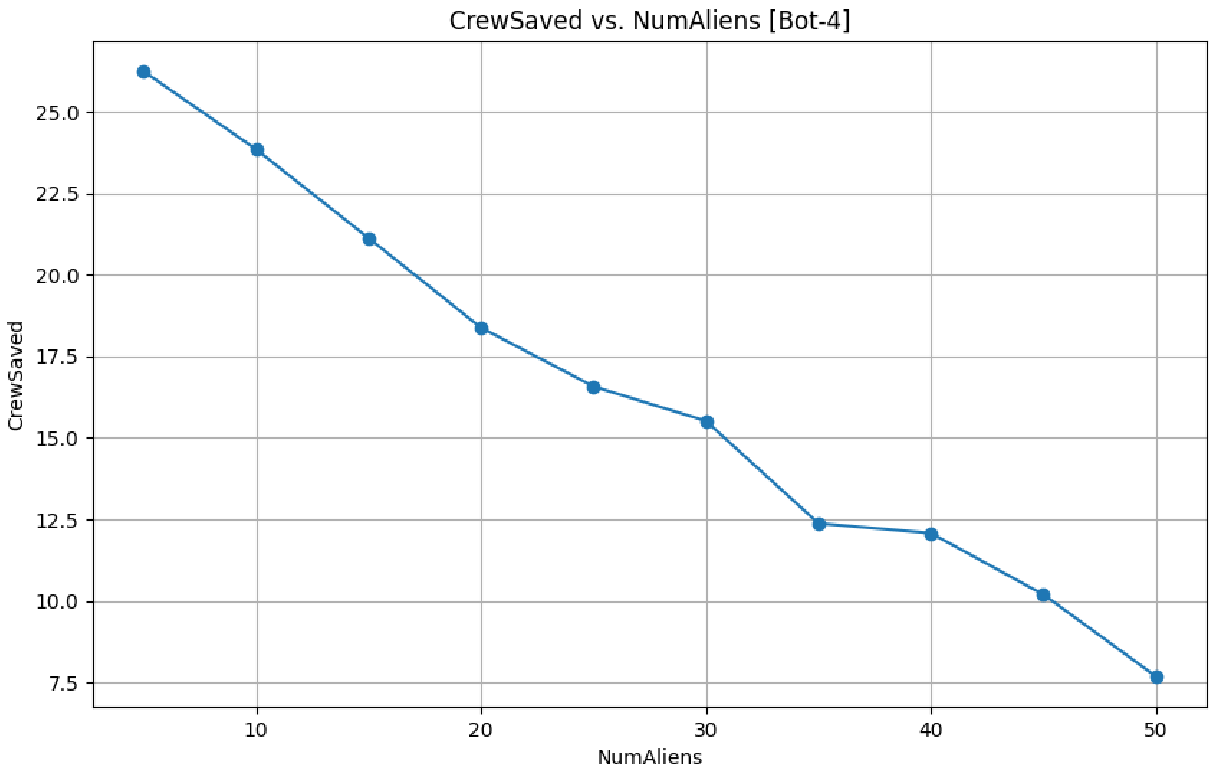
In order to avoid paths with frequent alien visits, we introduced the idea of approximating the regions where the aliens are most likely to travel. In this, we take the current alien positions in each step and generate random alien movements for a few steps. We repeat this step multiple times so as to get a better approximation of path frequency. Every alien step movement is stored in a dictionary. This dictionary thus acts as a frequency count for cells in the ship. The key corresponds to the cell in which the alien has moved through and the value is the number of times an alien moved through that cell. Later on, the heuristic is adjusted for this cells in such a manner that the cells with higher frequency of alien visits have a lower probability of being in the A* path. Hence, by avoiding the cells with maximum likelihood of alien presence, we keep the bot alive for a longer time and increase the chances of saving more crew lives.

There are 3 main factors to be considered for the above approach:

1. Number of steps to be simulated by the alien
2. Number of simulations of the above movement
3. Adjusting/multiplicative factor for the heuristics

While there are no touchstones for a good value for the above factors, it is logically backed that with increase in the number of steps or simulations, the multiplicative factor would decrease. After experimenting with multiple values of the above factors, the ideal hyper-parameter values came out to: 15 steps, 50 simulations, 0.05 multiplicative factor.


CrewSaved vs. NumAliens [Bot-4]
CrewSaved
NumAliens
25.0
22.5
20.0
17.5
15.0
12.5
10.0
7.5
10
20
30
40
50


CrewSaved vs. NumAliens [Bot-4]
CrewSaved
NumAliens
25.0
22.5
20.0
17.5
15.0
12.5
10.0
7.5
10
20
30
40
50

Drawback of Bot 4:
Bot 4's unique approach to heuristics occasionally leads to conflict between its own survival and
its mission to save the crew. This results in the bot opting for paths that are significantly longer, taking it away from the crew's location. Due to the cap of 1000 steps, this kind of trade off leads to less number of crews saved before reaching 1000 steps. Thus although it saves more crew lives in a lesser amount of time than Bot-3, the performance can still be enhanced further.

Proposed Improvements:
This can be solved by fine tuning the heuristic by collecting the data over multiple simulations and analyzing the data to tune the heuristic multiplier.
As well as providing more information to the bot by running more simulations for more steps, which is currently restricted to 10 steps for 30 simulations due the computational resources available.

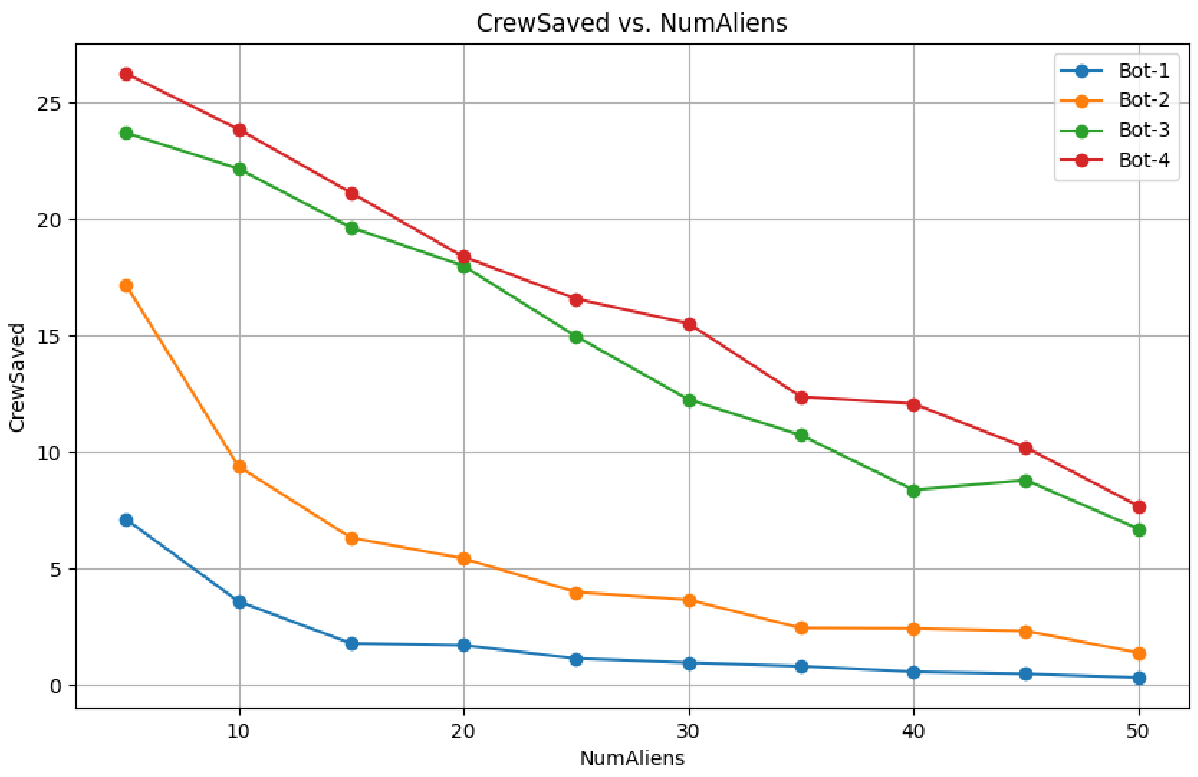
Evaluation Metrics

Below, we show the table comparing the average values of crew saved and the time for which the bot was alive for each Bot strategy. We consider 30 aliens as a touchstone. The percentage increase is shown over each successive bot. As can be seen from the table, there is a dramatic increase for both the parameters from Bot-1 to Bot-2 and from Bot-2 to Bot-3. There is also a marginal increase in performance for Bot-4 from the already optimized Bot-3.

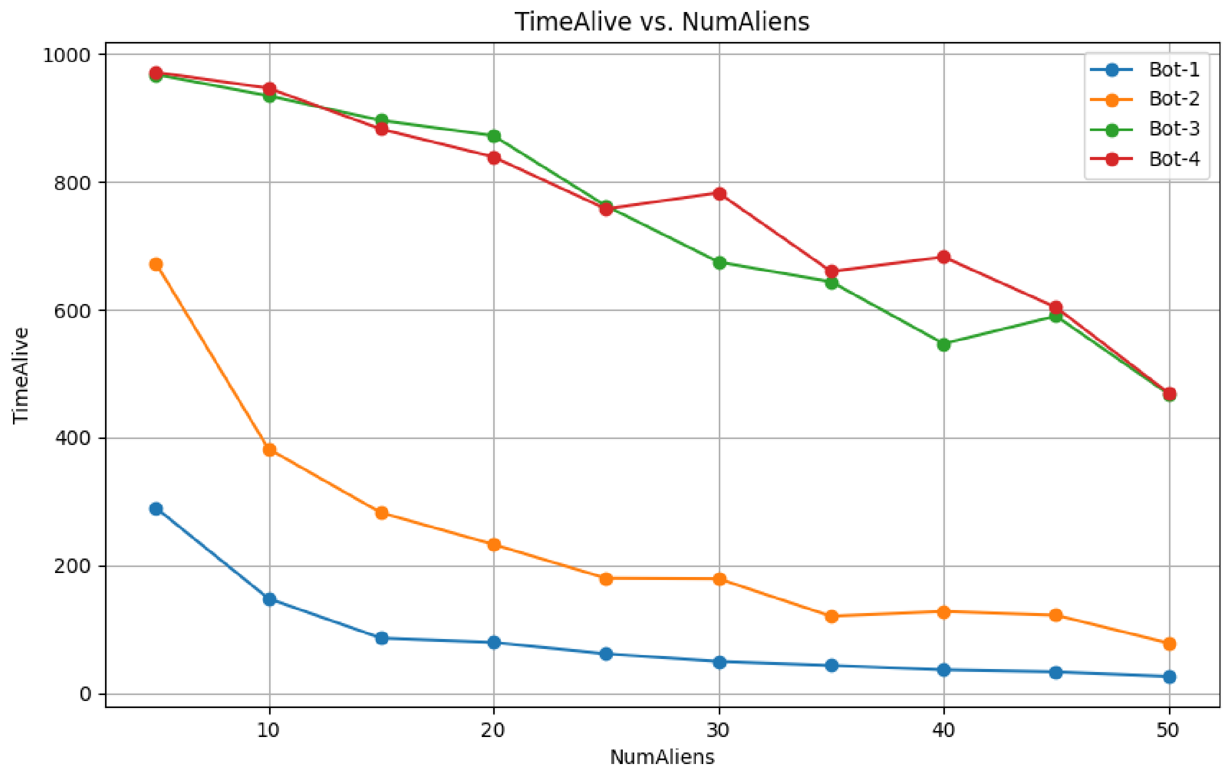
Bot	Average CrewSaved	Percentage increase
Bot-1	0.97	-
Bot-2	3.67	+278%
Bot-3	12.26	+234%
Bot-4	15.52	+26.6%

Bot	Average TimeAlive	Percentage increase
Bot-1	49.42	-
Bot-2	178.77	+261%
Bot-3	674.65	+277%
Bot-4	782.90	+16%

Bot Performance



As visible in the graph, each successive bot is an improvement over the previous bot. The values from the table can be visually re-affirmed in the graph. A massive jump in the number of crew saved can be observed between Bot-2 and Bot-3 by adding the feature of avoiding the cells adjacent to the aliens.



Again, the above graph visually affirms the results posted in the above table. There is improvement in each successive bot strategy. An interesting observation here is to analyze the TimeAlive parameter comparison between Bot-3 and Bot-4. The time for which Bot-4 is alive is either less than that of Bot-3 or marginally more. But, the crew lives saved by Bot-4 is consistently more than those saved by Bot-3. This brings about the inference that the improved heuristics for Bot-4 enables it to save more crew lives in lesser time as compared to Bot-3.

Bot-5 (Ideal Bot):

Monte Carlo Approach

Theoretical considerations might suggest that Monte Carlo Simulation, a probabilistic approach, could outperform traditional deterministic methods like A* Algorithms.

Theoretical Advantages

Theoretically, the Monte Carlo Simulation Algorithm offers several advantages:

- Adaptability: The algorithm can adapt to changing scenarios and unpredictable alien movements, making it suitable for dynamic environments.
- No Heuristic or Complete Knowledge: Monte Carlo does not require a heuristic or complete knowledge of the environment. This can be beneficial in situations where obtaining accurate heuristic information is difficult.
- Probabilistic Decision-Making: It can provide probabilities for different actions, allowing bots to make informed decisions. This offers adaptability to dynamic environments, like those with unpredictable alien movements. Theoretically, this adaptability should give the bot an advantage.

Implementation Strategy:

For each bot action, it will simulate multiple scenarios of alien movements to make decisions based on successful crew rescues.

Edge Case Challenge:
In certain scenarios, the bot encounters an issue where it repeatedly traverses the same explored path back and forth while trying to avoid aliens. This behavior results in the bot spending an excessive amount of time exploring unproductive routes to reach the crew member.

Proposed Improvement:
To mitigate this problem, we introduce a new heuristic function, Heuristic 2, to our existing set of heuristic functions.

- Previously, Heuristic 1 was based on the probability of the number of collisions per simulation.
- Now, with Heuristic 2: manhattan_distance(action, crew_cell) + cell_score[action], we incorporate two additional factors into the bot's decision-making process:
    1. Manhattan Distance: The distance between the bot's current action and the crew member's cell, providing guidance on proximity to the crew.
    2. Cell Score: This metric tracks the number of times a bot has visited a particular cell. It encourages the bot to prioritize cells it has visited less frequently, enhancing its focus on rescuing crew members.

By using Heuristic 2 in tandem with Heuristic 1, the bot is more likely to avoid revisiting cells it has explored extensively, thus optimizing its path exploration and improving its efficiency in rescuing crew members.

Factors Contributing to Poor Performance
However, in practice, the Monte Carlo Simulation Algorithm might not perform as expected, and several factors contribute to its underwhelming performance:

1. High Dimensionality: The high dimensionality of the problem, with numerous open cells and unpredictable alien movements, can make it challenging for Monte Carlo to sample relevant paths effectively. It may spend too much time exploring unproductive paths.
2. Lack of Domain-Specific Information: Monte Carlo Simulation relies heavily on random sampling, and it does not leverage domain-specific information about the environment. In contrast, A* Algorithm uses heuristics that provide valuable guidance in pathfinding, leading to more efficient solutions
3. Evaluation Metrics: Monte Carlo Simulation's probabilistic nature makes it challenging to define clear evaluation metrics for bot performance. In contrast, A* Algorithms offer more quantifiable measures, making it easier to assess performance..
4. Resource Limitations: The limited computational resources available on the bot might limit the number of simulations that can be run within a reasonable time frame, reducing the accuracy and effectiveness of the Monte Carlo method.
5. Convergence Challenges: The Monte Carlo method might require a vast number of simulations to converge to a meaningful solution. In a time-constrained environment, achieving this level of convergence can be difficult.

Conclusion

- While the Monte Carlo Simulation Algorithm in the "Invasion of the Bot-Grabbers" method is adaptable and resilient to complex problems, its practical performance can be disappointing due to limitations in computational resources, granularity of simulations, and the inherent unpredictability of alien behavior.
- If provided with a substantial amount of computational power, the Monte Carlo method can perform ideally like an ideal bot.

Project Contribution:

Manan Davawala: Initial BFS implementation for Bot-1, Implementation of Bot-4
Reo Correia: Implementation of ship layout setup, A* approach for Bot-1 and Bot-2
Vijayendra Sai Chennareddy: Implementation of Bot-3 and Monte Carlo approach for Bot-5

Note: Every member of the group was equally involved in the ideation and debugging stages of the entire project. Everyone contributed immensely to resolving complicated coding and theoretical issues or dilemmas. Furthermore, each member brought their own expertise to the project in a way that's difficult to express as a simple part of their contribution!