

Intro to AI Project 1 - Fire Extinguisher

By Krithish Ayyappan and Lakshya Gour

Overview of Bot 4s...

In order to understand the best strategy for our bot 4 we ended up trying out many different methods. Here we detail all the many variations of bot 4 that we ended up testing out:

Bot Strategies:

Bot 4: A with Manhattan Distance Heuristic*

- **Design:** Bot 4 employed the A* algorithm, using the Manhattan distance heuristic to guide its pathfinding. It calculated the best path to the goal by balancing the distance traveled and the remaining distance to the goal.
- **Fire Avoidance:** The bot avoided cells adjacent to fire to reduce the likelihood of getting trapped.
- **What Worked:** This strategy was effective at finding paths in fire-free areas but struggled when the fire spread more rapidly or occupied large sections of the grid.

Bot 5: A with Threshold for Fire Tolerance*

- **Design:** Bot 5 built upon Bot 4 by introducing a threshold that allowed it to consider paths with a limited number of fire-adjacent cells if no completely safe paths were available.
- **What Worked:** This added flexibility helped in extreme scenarios but didn't significantly outperform Bot 4. The threshold strategy did not show notable improvement. We attempted to try and see how different thresholds would affect the success rate for the bot but did not find significant difference when it came to the threshold value and so we chose a risk threshold of 5, which would dynamically choose the must avoid cells.

Bot 6: Fire Spread Simulation

- **Design:** Bot 6 anticipated fire spread and assigned higher costs to cells likely to catch fire. It simulated future fire states to predict riskier paths.
- **What Worked:** Predicting fire spread helped the bot avoid getting trapped but led to more computational complexity. The bot's decision-making process became slower as it accounted for future risk, which hindered its performance in fast-spreading fire scenarios. Initially we did not know that the bot could be supplied with information such as the flammability metric which is what led to the creation of this bot.

Bot 7: Adaptive A Search*

- **Design:** Bot 7 initially avoided both fire and adjacent cells but adapted to take slightly riskier paths if no completely safe routes were available. It adjusted its strategy dynamically.
- **What Worked:** The dynamic adjustment improved flexibility, but the performance gain was minimal in practice.

Bot 8: Fire-Aware Uniform Cost Search (UCS)

- **Design:** Bot 8 used Uniform Cost Search with a cost function that considered:
 - **Fire Risk:** Penalized cells near fire.
 - **Distance to Goal:** Penalized longer paths.
 - **Escape Potential:** Estimated how easy it would be to escape from future fire risk in a particular cell.
- **What Worked:** This multifaceted cost function gave Bot 8 a strong performance, allowing it to balance immediate and future fire threats more effectively.

Bot 9: Fire-Aware UCS with Euclidean Distance Heuristic

- **Design:** Bot 9 extended Bot 8's strategy by using **Euclidean distance** as the heuristic for calculating the straight-line distance between points. This made it more efficient in scenarios where diagonal movement could help.
- **What Worked:** Bot 9 achieved the **highest average success rate** across all values of $q(0.744316)$, performing particularly well when tested with flammability parameters of $q = 0.2, 0.4$, and 0.85 . The combination of UCS, Euclidean distance, and fire risk awareness worked well.

Bot 10: UCS with Chebyshev Distance Heuristic

- **Design:** Similar to Bot 9, but used Chebyshev distance, which considers diagonal movements, making it potentially more efficient for grids where diagonal paths could reduce travel time.
- **What Worked:** While Bot 10 showed promise in certain configurations, it didn't outperform Bot 9 in most simulations.

Bot 11: A with Euclidean Distance*

- **Design:** Bot 11 combined Bot 7's adaptive search with Euclidean distance, aiming for more precise pathfinding.
- **What Worked:** The diagonal movement consideration improved the bot's efficiency in finding shorter paths, but it still wasn't as fire-aware as Bot 9.

Bot 12: A with Chebyshev Distance*

- **Design:** Bot 12 followed the same approach as Bot 11 but used Chebyshev distance to evaluate diagonal paths.

- **What Worked:** Like Bot 11, it showed promise in certain grid configurations but didn't achieve the same success rate as Bot 9.

Bot 13: Fire-Aware UCS with Fire Spread Prediction and Chebyshev Distance

- **Design:** Bot 13 was a complex version of Bot 8, incorporating future fire spread prediction alongside Chebyshev distance. It ended up simulating another spread of the fire to see how far the fire would reach.
- **What Worked:** While this bot was extremely cautious and strategic, its complexity often resulted in slower decision-making, which doesn't directly affect this project as it doesn't consider time taken to make decisions, however would do so in a more realistic manner.

Detailed Bot 9 (Our Final Bot 4) Explanation

So some things we noticed after collecting data and looking at our simulation was that the burning number of cells, the future burning cells and the distance to the button all matter. If the bot only used current cells, it can't guarantee safety for its next moves and get stuck or it may be better as the bot is actually farther than the fire to the button, thus making a rewarding approach. We needed a combination of these approaches for Bot 9 where it can avoid cells based on its risk, distance, and future risk. Thus, we developed Bot 9 with the parameters of fire penalty, distance penalty, and escape penalty. These 3 parameters define the next path in such a way that it combines the risk of the current cell by using how many burning neighbor cells there are, how far this cell is from the button to choose the shortest path, and an escape cost to see if this cell is really worth it by simulating the next step's burning cells.

The cost function here would be a utility function:

$$\text{Cost}(\text{cell}) = (\text{fire_penalty_constant} \times \text{burning_neighbors}) + (\text{distance_penalty_constant} \times \text{distance_to_goal}) + (\text{escape_penalty_constant} \times \text{fire_potential}).$$

Bot 9 was designed by tuning the fire penalty, distance penalty, and escape penalty to balance immediate risks, future threats, and efficiency in reaching the goal. By simulating different scenarios, we refined these parameters to ensure the bot avoids dangerous cells while still prioritizing the shortest, safest path to the button. This tuning allowed Bot 9 to dynamically adapt its strategy based on real-time conditions, making it more effective in complex environments. Based on Bot 8's original idea, we were able to achieve higher results by changing how we calculate distance to Euclidean Distance Heuristic.

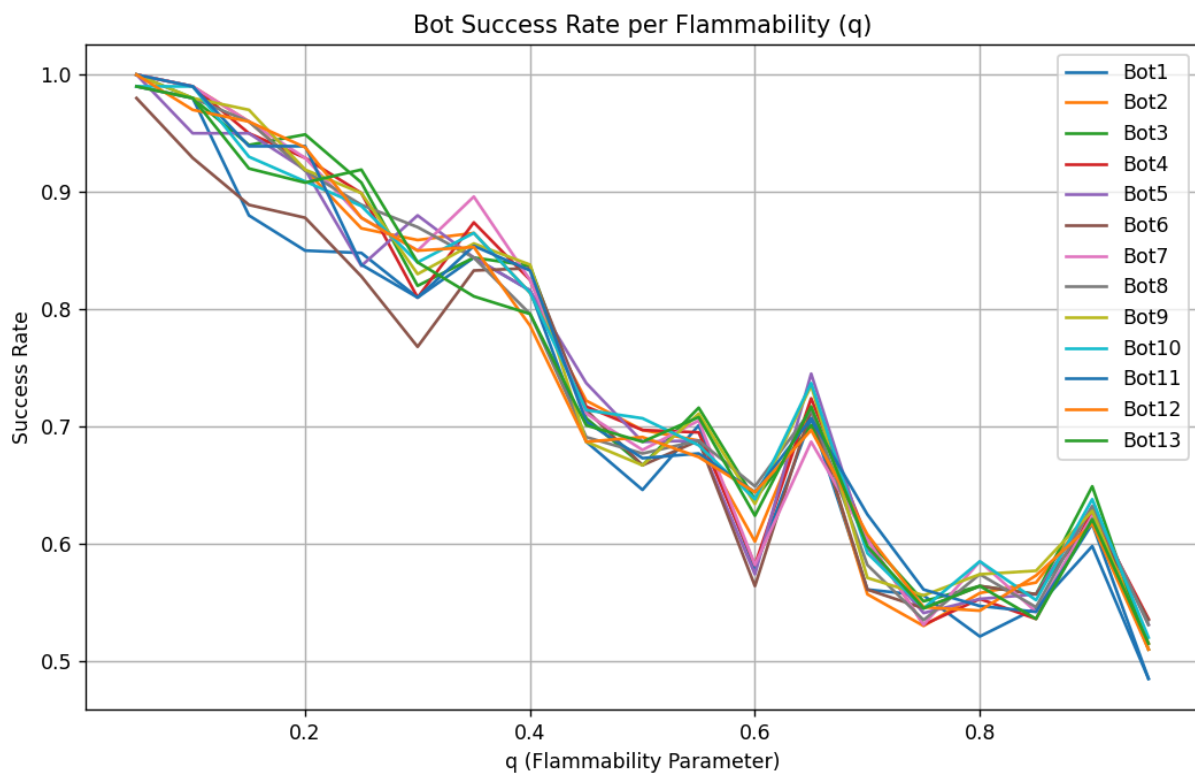
Analysis

In order to analyze which bots performed the best, we ran simulations at different flammability parameters (q) from 0 to .95 at 0.05 steps. For each of the parameter values, we conducted 100 simulations for each of the bots. The simulations consisted of the same starting positions for the fire, the bot, and the button. The only randomness that we did not keep the same between the data was the spread of the fire (under guidance of the TA). We would save the path that the bot took, the starting positions for the button, bot start, fire start, and whether the situation was

possible to succeed in. Taking out the cases where it was not possible to successfully reach the button (for example when the button was engulfed or surrounded by a ring of fire), we then took the average success rates for each of the bots at each q value. Here is one way we decided to approach analyzing the differences in our bots:

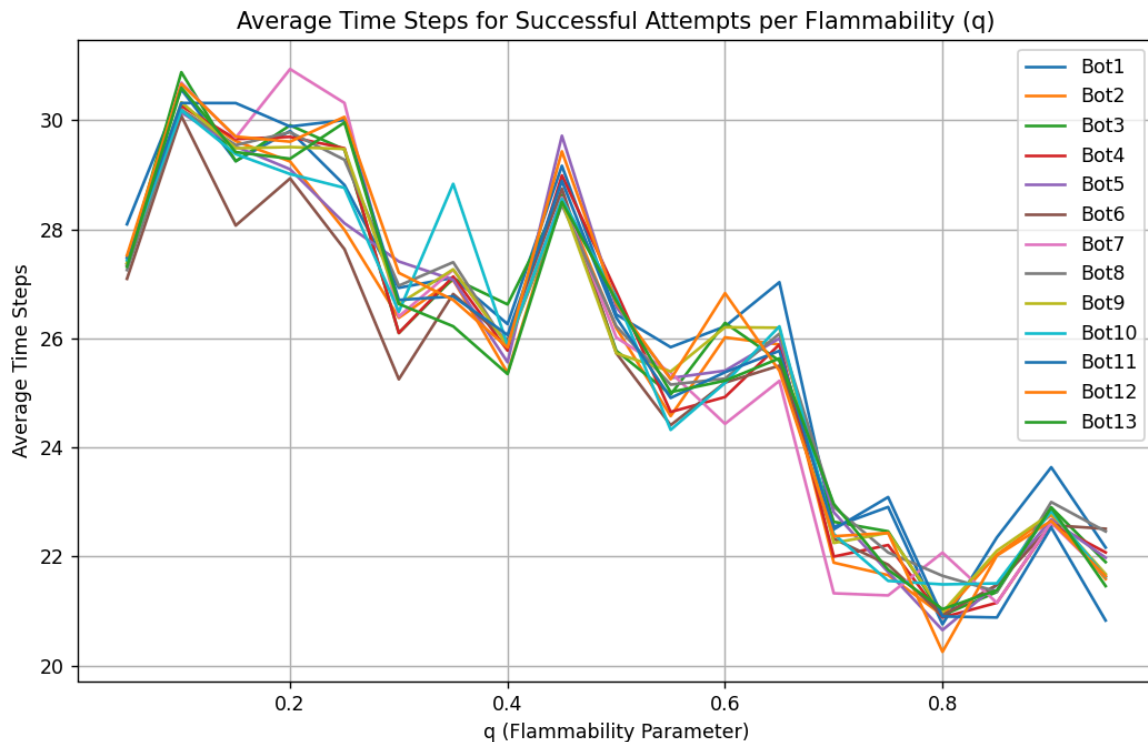
Success Rate of Each Bot at Each q:															
q (Flammability Parameter)	Bot1	Bot2	Bot3	Bot4	Bot5	Bot6	Bot7	Bot8	Bot9	Bot10	Bot11	Bot12	Bot13	Best	Bot
0.05	0.99	1.00	1.00	1.00	1.00	0.98	1.00	0.99	1.00	0.99	1.00	1.00	0.99	Bot2	
0.10	0.98	0.99	0.98	0.99	0.95	0.92	0.99	0.98	0.98	0.99	0.99	0.97	0.98	Bot2	
0.15	0.88	0.96	0.94	0.95	0.95	0.88	0.96	0.96	0.97	0.93	0.93	0.96	0.92	Bot9	
0.20	0.85	0.91	0.94	0.91	0.91	0.86	0.92	0.90	0.91	0.90	0.93	0.91	0.89	Bot3	
0.25	0.84	0.86	0.89	0.89	0.82	0.82	0.86	0.88	0.89	0.87	0.83	0.86	0.91	Bot13	
0.30	0.81	0.85	0.82	0.81	0.88	0.76	0.85	0.87	0.83	0.84	0.81	0.85	0.84	Bot5	
0.35	0.81	0.83	0.81	0.83	0.81	0.80	0.86	0.81	0.83	0.83	0.82	0.81	0.77	Bot7	
0.40	0.80	0.79	0.82	0.80	0.80	0.81	0.80	0.78	0.83	0.79	0.80	0.77	0.78	Bot9	
0.45	0.68	0.70	0.70	0.71	0.73	0.70	0.69	0.67	0.68	0.70	0.69	0.68	0.68	Bot5	
0.50	0.64	0.69	0.66	0.69	0.68	0.66	0.66	0.67	0.66	0.70	0.66	0.67	0.68	Bot10	
0.55	0.68	0.66	0.68	0.66	0.66	0.66	0.67	0.66	0.69	0.65	0.65	0.64	0.68	Bot9	
0.60	0.55	0.56	0.60	0.53	0.54	0.53	0.53	0.61	0.59	0.60	0.58	0.58	0.58	Bot8	
0.65	0.70	0.71	0.68	0.71	0.73	0.70	0.68	0.70	0.72	0.73	0.70	0.69	0.71	Bot5	
0.70	0.55	0.54	0.58	0.58	0.59	0.55	0.58	0.57	0.56	0.58	0.60	0.59	0.58	Bot11	
0.75	0.55	0.53	0.54	0.52	0.53	0.54	0.52	0.53	0.55	0.54	0.55	0.53	0.54	Bot1	
0.80	0.50	0.53	0.53	0.52	0.52	0.53	0.55	0.54	0.54	0.55	0.52	0.51	0.53	Bot7	
0.85	0.54	0.55	0.54	0.52	0.54	0.54	0.52	0.53	0.56	0.53	0.52	0.55	0.52	Bot9	
0.90	0.58	0.59	0.61	0.59	0.60	0.59	0.58	0.60	0.59	0.60	0.58	0.58	0.59	Bot3	
0.95	0.48	0.51	0.50	0.52	0.52	0.53	0.50	0.52	0.50	0.51	0.47	0.50	0.50	Bot6	

[Success Rate Data] [Table above, visualized below]



- **Bot 9** had the highest success rate across simulations, achieving an average success rate of **0.744316**.

- The threshold mechanism (limiting the number of allowed fire cells) did not show any significant improvement in performance.
- Strategies involving fire-risk awareness, future fire spread prediction, and effective cost functions yielded better results.



[Average Time each bot took for each q]

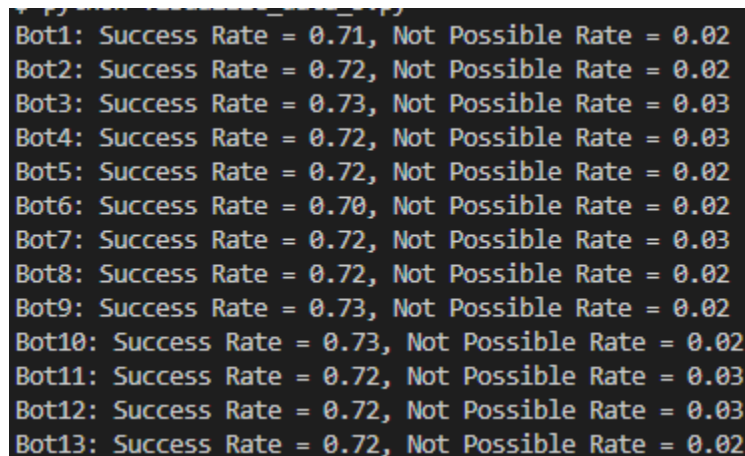
We notice that bots emphasize shorter distance and keep more options available (less limiting cells), they tend to be faster and have less average time steps. Most bots that are not relying on the flammability of the ship such as Bot 1, 2, 3 tend to have a consistent average time step throughout the board. However, bots such as 6, 7, 8, 9, 10, 11, 12, 13 tend to rely on the flammability and cost of a cell, thus taking more cells to achieve a safer path to the goal. As we can see with the brown line (Bot 6), it tends to be the fastest through simulation with a low q parameter. However, this jumps after the 0.5 mark showing a peak in our downward linear model and a shift in these risk-calculating bots. While utility cost function bots such as 5, 8, 9, tend to also jump in this shift at 0.5 and take longer routes, they are not as bad as simulation bots like Bot 6, 7, 12, 13 which tend to make the most moves. Bots 1, 2, 3, after the 0.5 peak in q tended to also switch and take the most moves. Essentially, the more the bot was asked to avoid, the more moves it took in its approach.

Although we tried many strategies, none of them were statistically significantly better than any other, as most of them had very close success rates. After analyzing the performance across various metrics, we decided to choose **Bot 9** for its overall balance between success rate and time efficiency.

While simple A* algorithms were effective in fire-free environments, bots that integrated fire-awareness and dynamic risk evaluation strategies performed significantly better in fire-prone scenarios. The success of Bot 9 suggests that factoring in future fire spread and using more flexible heuristics like Euclidean distance can greatly enhance a bot's ability to make informed decisions and successfully navigate dangerous environments

Why Failure? When is success not possible?

We notice that most times the bot fails more as the flammability parameter tends to increase, this is noticed in the success rate by q graph. As q increases, the success rate across all bots tends to decrease due to the number of cells available or meeting a bot's conditions significantly decreasing. One problem we noticed was what depends on a bot's decision. If the fire was closer, the bot should try to beat the fire and take a faster approach. Essentially, it is a race disregarding the risk in tiles to beat the fire to the button before it gets engulfed. Another issue was the more complicated our approach became, the more difficult time a bot had finding the optimal choice and deciding it was Not Possible.



Bot1:	Success Rate = 0.71,	Not Possible Rate = 0.02
Bot2:	Success Rate = 0.72,	Not Possible Rate = 0.02
Bot3:	Success Rate = 0.73,	Not Possible Rate = 0.03
Bot4:	Success Rate = 0.72,	Not Possible Rate = 0.03
Bot5:	Success Rate = 0.72,	Not Possible Rate = 0.02
Bot6:	Success Rate = 0.70,	Not Possible Rate = 0.02
Bot7:	Success Rate = 0.72,	Not Possible Rate = 0.03
Bot8:	Success Rate = 0.72,	Not Possible Rate = 0.02
Bot9:	Success Rate = 0.73,	Not Possible Rate = 0.02
Bot10:	Success Rate = 0.73,	Not Possible Rate = 0.02
Bot11:	Success Rate = 0.72,	Not Possible Rate = 0.03
Bot12:	Success Rate = 0.72,	Not Possible Rate = 0.03
Bot13:	Success Rate = 0.72,	Not Possible Rate = 0.02

[Table 2: Success Rate and Not Possible Rate]

With the data below and approaches above, we notice that bots that tend to have approaches that block off more cells (looking into all neighbors of burning cells, calculating future fire cells, factoring in distance into burning cells and its neighbors, etc.) tends to increase a bot's Not Possible Rate. This happens because these strategies progressively reduce the number of available cells as the simulation progresses. By constraining more cells over time, the bot effectively limits its possible paths for movement, especially in scenarios where the number of unblocked cells is already small. As a result, the bot is left with fewer viable options to reach its goal, increasing the likelihood of encountering situations where success becomes impossible. In essence, imposing stricter constraints in an already limited environment amplifies the chances

of a "Not Possible" outcome. Thus, larger constraints with an already small domain leads to a higher not possible state.

Improvements and Making the Best Bot:

Through our data analysis and previous insights, we want a bot that does not “remove” too many cells from its potential choices as nothing can be certain due to a chance of a cell being caught on fire based on a parameter and randomness. Also, if the fire is closer to the button, then the bot should prioritize beating the fire and taking riskier decisions despite the growth of the fire. If the bot is closer than the fire, then the bot can easily make smart decisions to reach the button with less risk. Thus, the best way to create a bot that can factor in all these decisions would be with **context-awareness**. A bot that has knowledge of how far the current fire is from the button, a risk factor that updates based on this decision to influence its “risky” decision making (stepping into cells closer to the fire), and somehow planning a safer route while doing so (escape route calculation). This risk-aware strategy allows the bot to be adaptive, carefully navigating high-risk areas when necessary while always maintaining a fallback plan to avoid becoming trapped. By intelligently weighing risk versus reward in its situation based on its given situation, it can outperform more rigid strategies that either avoid risk altogether or block too many paths, leading to fewer successful outcomes.