Fire Extinguisher: Project 1 in Introduction to AI (CS-520)

Harsha Rajendra (hr458), Santhosh Janakiraman (sj1230)
Computer Science Department, Rutgers University
hr458@scarletmail.rutgers.edu, sj1230@scarletmail.rutgers.edu

Abstract

In this project, we are addressing the challenges faced by an autonomous robotic entity aboard the deep space vessel Archaeopteryx. The robot is tasked with responding to fire emergencies while the crew is in hibernation. The vessel's layout is meticulously structured on a 40x40 grid, with cells designated as 'blocked' or 'open' through a systematic opening process, ensuring that approximately 60% of the grid is accessible. The robot's mission is to navigate to a button that activates the fire suppression system, while skillfully avoiding the threat of spreading fire, which can start randomly in an open cell. We have developed four robust strategies for the robot, including a static pathfinding approach that disregards fire spread, a dynamic replanning method that accounts for current fire locations, a cautious strategy that avoids both fire and its immediate vicinity, and a custom-designed robot incorporating innovative decision-making algorithms. We are rigorously evaluating each robot's performance across multiple simulations with varying fire spread probabilities, with the firm goal of analyzing success rates and identifying failure causes. Our objective is to explore different strategies for addressing challenges in various environments, improving our understanding of optimal methods and designing algorithms with high success rates.

The Task

At the outset of the simulation (t=0), the autonomous bot, fire suppression button, and initial fire cell are deliberately positioned at distinct random locations within the grid of open cells on the deep space vessel Archaeopteryx to ensure a realistic operational scenario. The simulation unfolds in discrete time steps (t=1,2,3,...) during which critical actions are meticulously executed:

1. **Decision Making:** The bot analyzes its position and strategically selects a neighboring cell based on a comprehensive navigation strategy.

2. **Movement Execution:** The bot adeptly relocates to the chosen neighboring cell, reflecting its navigation capabilities.

3. **Button Interaction:** Upon entering the cell containing the fire suppression button, the bot activates the mechanism, effectively extinguishing the fire and accomplishing the mission.

4. **Fire Advancement:** In the absence of a successful button press, the fire systematically spreads to adjacent open cells based on predefined rules, creating a dynamic hazard that the bot must skillfully navigate.

5. **Collision Check:** The bot's position relative to the fire is rigorously monitored. Any overlap between the bot and the fire deems the task a failure, underscoring the high-stakes nature of the navigation strategy.

The primary objective is to develop and evaluate robust strategies that guide the bot's navigation decisions as it traverses the ship, effectively avoiding fire and reaching the button to mitigate the threat. This project seeks to enhance our understanding of how various factors influence algorithm performance in dynamic environments and how we can address different challenges by implementing a range of responsiveness and adaptability.

The Ship

The ship's layout is meticulously crafted on a square grid of size 40×40, initially designating all cells as 'blocked.' An interior cell is then randomly chosen to be 'open,' initiating an iterative process of modifying the grid by opening blocked cells with precisely one open neighbor, until no further cells can be opened. Prior to opening dead end cells—defined as open cells with only one open neighbor—a criterion is set to ensure that approximately 60% of the ship is open. This step is essential for striking a delicate balance between accessibility and obstacles, ultimately enhancing the ship's layout for smooth movement and maneuvering. Subsequently, roughly half of the identified dead ends are randomly selected to open one of their blocked neighbors, adding refinement to the improvement of the ship's design.

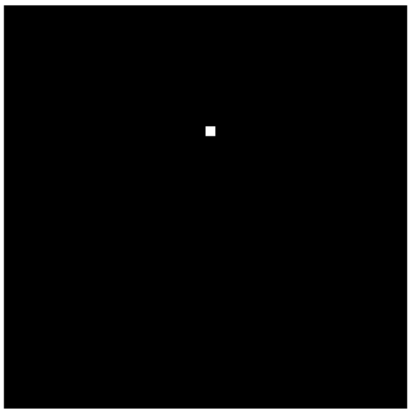
Figure 1: The Initial Grid with Starting Cell. Here (12,20)

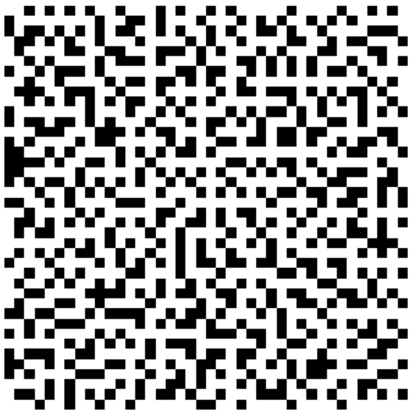
Figure 2: The Grid after Opening Cells and Finding Dead Ends

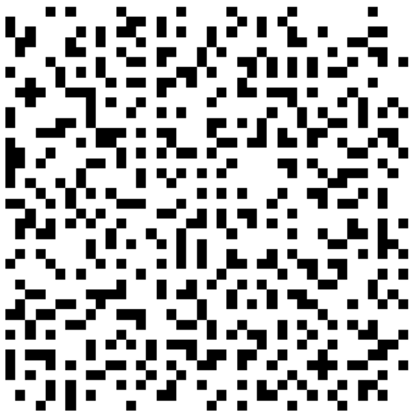
Figure 3: The Final Grid after Expanding Dead Ends

1. **Grid Initialization and Initial Cell Selection:** A 40×40 square grid is established with all cells initially blocked. The interior cell Ex:(12,20) is randomly selected and marked as open, serving as the starting point for subsequent modifications.
2. **Accessibility Iterative Process and Dead End Identification:** Iteratively all currently blocked cells that have exactly one open neighbor are identified, and one of these cells is opened randomly.
3. **Final Layout and Connectivity Enhancement:** The algorithm randomly selects half of the identified dead ends to mark one of their neighboring blocked cells as open, enhancing connectivity and layout.After completing all the iterations and adjustments, the grid has more open areas, allowing for easier movement and navigation.

Placing the Bot, Fire and Button

The bot is positioned within an open cell in the ship's grid, and it can move one cell at a time in any of the four cardinal directions: up, down, left, or right. A fire ignites randomly in an open cell, spreading to adjacent open cells at each time step, with its spread governed by a probabilistic model. The probability of a non-burning cell catching fire is defined by the flammability parameter q (ranging from 0 to 1) and the number of currently burning neighboring cells K, following the formula:

$$\text{Probability} = 1 - (1 - q)^K$$

When q is near 0, fire spread is unlikely, while values close to 1 significantly increase the probability of spread, particularly in cells with multiple burning neighbors. The button, located in an empty cell within the ship, serves as the activation mechanism for the fire suppression system, requiring it to be pressed in order to stop the fire.

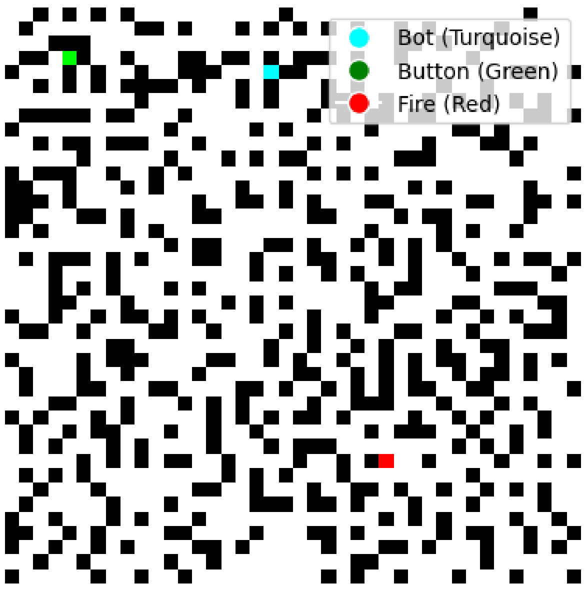

Figure 4: The grid after the deployment of the Bot, Fire, and Button

Bot Designs

Essentially, the bot can choose to perform one of the following actions at any time: move to a nearby unoccupied cell or stay where it is. The difference between the approaches is in how the bot makes its decisions. In this project, we have implemented and evaluated the efficacy of the strategies outlined below.

- **Bot 1:** Bot 1: Uses Breadth-First Search (BFS) to find the shortest path to the button by avoiding the initial fire cell and executing that plan. However, it ignores the spread of the fire, highlighting the limitations of static pathfinding and its failure to account for evolving threats that may arise.
- **Bot 2:** Employs Uniform Cost Search (UCS) to prioritize safety by avoiding fire cells and calculating the cost of moving through safe cells, ensuring that the path taken minimizes overall risk. It stays in place if the next position is on fire, enabling dynamic decision-making and cost-effective pathfinding.
- **Bot 3:** Implements A* search, utilizing a heuristic based on the Manhattan distance to the goal. At every time step, the bot re-plans the shortest path to the button, considering both current fire cells and their adjacent cells as threats, and avoiding them whenever possible. If a safe path isn't available, it plans the shortest route based solely on the current fire cells and executes the next step in that plan, enhancing its navigation strategy while prioritizing safety.
- **Bot 4:** Enhances A* by predicting fire spread and creating a safety corridor. It assesses risk levels based on proximity to fire, enabling informed movement decisions, such as detours when fire is

imminent. The safety corridor consists of safe paths that avoid current and predicted fire cells. By prioritizing these corridors, the bot can stay in safer areas. If a planned move is risky—such as entering a cell likely to catch fire—it can choose to remain in place or reroute within the corridor, reducing the risk of being caught in the fire while efficiently approaching the button.

Running the Code:

Make sure that Python is installed on your machine. Execute the following commands in your terminal or command prompt to install the required packages:

```
cd  /Desktop/AI_Project_Harsha_Santhosh

pip install -r requirements.txt
python project1_HH_SJ.py
```

(The requirements.txt file is included with the code).

Troubleshooting: You can review the documentation for any clarifications or contact us directly if you have any questions.

Data Analysis Metrics:

The following analytics are derived from an extensive data run conducted on all four bots, using the parameters outlined below.

Grid Size=40

Simulation Count=500

Q Values=0.1,0.2,...1.0

$$\textbf{Average Success Rate} = \frac{\sum_{i=1}^{n} \text{Success}_i}{\text{Simulation Count}}$$

Grid Size: 40 → Defines a square grid of dimensions 40x40 cells where the bots operate. This fixed size ensures consistent conditions across simulations, influencing the complexity of navigable paths and interactions with fire spread.

Simulation Count: 200 → Specifies the number of iterations for evaluating each algorithm's performance. This repetition enhances result reliability by averaging out randomness, leading to statistically significant insights into each bot's effectiveness.

Q Values: → Generates a list of flammability values from 0.0 to 1.0 in increments of 0.05. Each value represents a different probability of fire spread, allowing for analysis of how varying flammability levels affect bot decision-making.

Average Success Rate → The average success rate is calculated across multiple simulations for each bot and flammability level. This metric reflects overall effectiveness and adaptability of the bots in navigating the grid, avoiding fire and reaching the fire extinguish button in randomly generated grids.

Simulation:

The simulation involves creating a grid representing a space vessel with "open" and "blocked" cells. The main variables changing are the fire spread probability (q) and the algorithms used for pathfinding: Bot 1 (BFS), Bot 2 (UCS), Bot 3 (A*), and Bot 4 (Custom).

For each simulation:

- A grid is created, and cells are opened until at least 60% are "open."
- A bot navigates from a starting position to a button location while avoiding fire cells.
- Fire spreads based on the probability q after each move. A total of 500 grids are generated for each q value. The expected outputs are overall success rates for each bot and success rates when a successful path is possible, plotted against varying fire spread probabilities (q).

2) Design, Algorithm & Working for Bot 4

Design

The flammability factor remains unknown to Bot 4, as having prior knowledge of a fixed flammability factor is impractical.

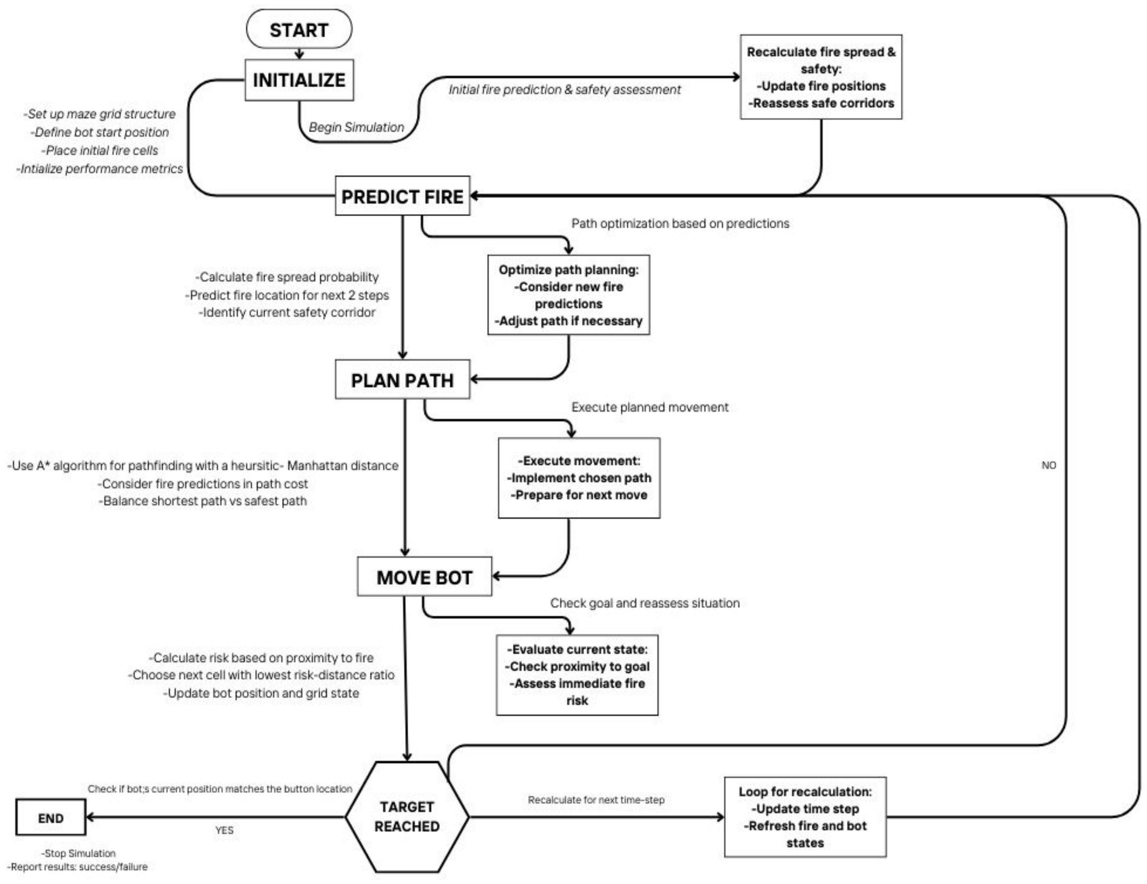
START
INITIALIZE
-Set up maze grid structure
-Define bot start position
-Place initial fire cells
-Intialize performance metrics
Begin Simulation
Initial fire prediction & safety assessment
Recalculate fire spread & safety:
-Update fire positions
-Reassess safe corridors
PREDICT FIRE
Path optimization based on predictions
-Calculate fire spread probability
-Predict fire location for next 2 steps
-Identify current safety corridor
Optimize path planning:
-Consider new fire predictions
-Adjust path if necessary
PLAN PATH
Execute planned movement
-Use A* algorithm for pathfinding with a heursitic- Manhattan distance
-Consider fire predictions in path cost
-Balance shortest path vs safest path
-Execute movement:
-Implement chosen path
-Prepare for next move
NO
MOVE BOT
Check goal and reassess situation
-Calculate risk based on proximity to fire
-Choose next cell with lowest risk-distance ratio
-Update bot position and grid state
-Evaluate current state:
-Check proximity to goal
-Assess immediate fire risk
Check if bot;s current position matches the button location
END
YES
TARGET REACHED
Recalculate for next time-step
Loop for recalculation:
-Update time step
-Refresh fire and bot states
-Stop Simulation
-Report results: success/failure

Figure 5: Design of Bot 4

Algorithm 1 Bot 4 Algorithm

```
Bot4Algorithm(grid, start, goal, fire_cells, q):
while current_position $\neq$ goal do
   predicted_fire ← PredictFireSpread(grid, fire_cells)
   safety_corridor ← CalculateSafetyCorridor(grid, current_position, goal, fire_cells, predicted_fire)
   path ← Bot4Pathfinding(grid, current_position, goal, fire_cells, predicted_fire, safety_corridor)
   if path is None then
      return False
   end if
   next_position ← path[1]
   risk_level ← AssessRisk(next_position, fire_cells, predicted_fire)
   if risk_level is high then
      Adjust next_position based on risk and safety_corridor
   else if risk_level is medium then
      if next_position not in safety_corridor then
         next_position ← current_position
      end if
   end if
   current_position ← next_position
   grid, fire_cells ← SpreadFire(grid, fire_cells, q)
   if current_position in fire_cells then
      return False
   end if
end while
return True

Function: PredictFireSpread(grid, fire_cells)
predicted_fire ← Copy of fire_cells
for each cell in fire_cells do
   Add adjacent cells to predicted_fire
end for
return predicted_fire

Function: CalculateSafetyCorridor(grid, start, goal, fire_cells, predicted_fire)
Use BFS to find safe paths from start to goal
return Set of cells in safe paths

Function: Bot4Pathfinding(grid, start, goal, fire_cells, predicted_fire, safety_corridor)
Use A* algorithm with custom heuristic considering fire and safety_corridor
return Optimal path or None if no path found

Function: AssessRisk(position, fire_cells, predicted_fire)
Calculate risk based on proximity to current and predicted fire
return Risk level (normalized between 0 and 1)
Function: SpreadFire(grid, fire_cells, q)
Spread fire to adjacent cells based on probability q
return Updated grid and fire_cells
```

- **Note:** grid: 2D list of strings, representing the maze layout
- start: tuple (int, int), starting position of the bot
- goal: tuple (int, int), target position (button location)
- fire_cells: set of tuples (int, int), positions of fire cells
- q: flammability, fire spread probability (0 to 1)
- current_position: tuple (int, int), current bot position
- predicted_fire: set of tuples (int, int), predicted fire positions
- safety_corridor: set of tuples (int, int), safe path positions
- path: list of tuples (int, int), planned path for the bot
- risk_level: float, assessed risk (0 to 1)

=0

Working

Bot 4 navigates the grid by predicting fire spread and evaluating risks based on proximity to the nearest fire. It establishes a dynamic safety corridor—an area deemed safe based on fire predictions and its current position. Using the A* algorithm, it finds the best route to the button while balancing distance and safety. The bot adapts its movements based on risk levels, taking calculated risks for rewards or staying put in dangerous situations.

Breakdown

Prediction: `predict_fire_spread(grid, fire_cells, steps=2)`

- Predicts how the fire will spread in the next 2 steps.
- Returns a set of cells that are likely to be on fire soon.
- Helps the bot anticipate dangerous areas.

Effect on Bot 4 movement: By predicting future fire locations, the bot can avoid moving into cells that might become dangerous in the near future, allowing it to plan safer routes.

Risk Assessment: `assess_risk(pos, fire_cells, predicted_fire)`

- Calculates the risk level for a given position.
- Returns a value between 0 (safe) and 1 (high risk).
- Used to decide whether to take risky moves or stay put.

Effect on Bot 4 movement: This function directly influences the bot's decision to move or stay in place. In high-risk situations, the bot might choose to remain stationary rather than move into danger.

Planning Safety:`calculate_safety_corridor(grid, start, goal, fire_cells, predicted_fire)`

- Identifies a safe path from start to goal, avoiding current and predicted fire.
- Returns a set of safe cells that form a corridor.
- Helps the bot find safer routes to the button.

Effect on Bot 4 movement: The safety corridor guides the bot's movement, encouraging it to stay within safer areas. This might lead to slightly longer paths, but with a higher chance of survival.

Pathfinding: `bot4_pathfinding(grid, start, goal, fire_cells, predicted_fire, safety_corridor)`

- Uses A* algorithm with custom heuristics.
- Considers fire locations, predicted fire, and safety corridor.
- Returns the optimal path to the goal, balancing shortest distance and safety.

Effect on Bot 4 movement: This function determines the primary path the bot will attempt to follow. It balances finding the shortest route with avoiding dangerous areas, leading to more intelligent navigation.

Table 1: Bot 4 Movement Strategy

Risk Level	**Action**
High risk (> 0.7)	Allow 30% chance (if random.random() < 0.3) to make a risky move, otherwise stay put
Medium risk (> 0.3)	Moves only if the next position is in the safety corridor.
Low risk (≤ 0.3)	Follows the planned path.

The bot uses the path from `bot4_pathfinding` path, adjusting its moves based on assessed risks—taking bold actions when needed, being cautious in moderate risks, or confidently following the path when safe.

2) *Bot Performance Analysis: Creating Test Environments and Evaluating Results:*

We graph the Average Success Rate against the flammability factor (Q) (0.1,0.2...1.0) for 500 Simulations to analyze how effectively each bot reaches the button across multiple simulations.

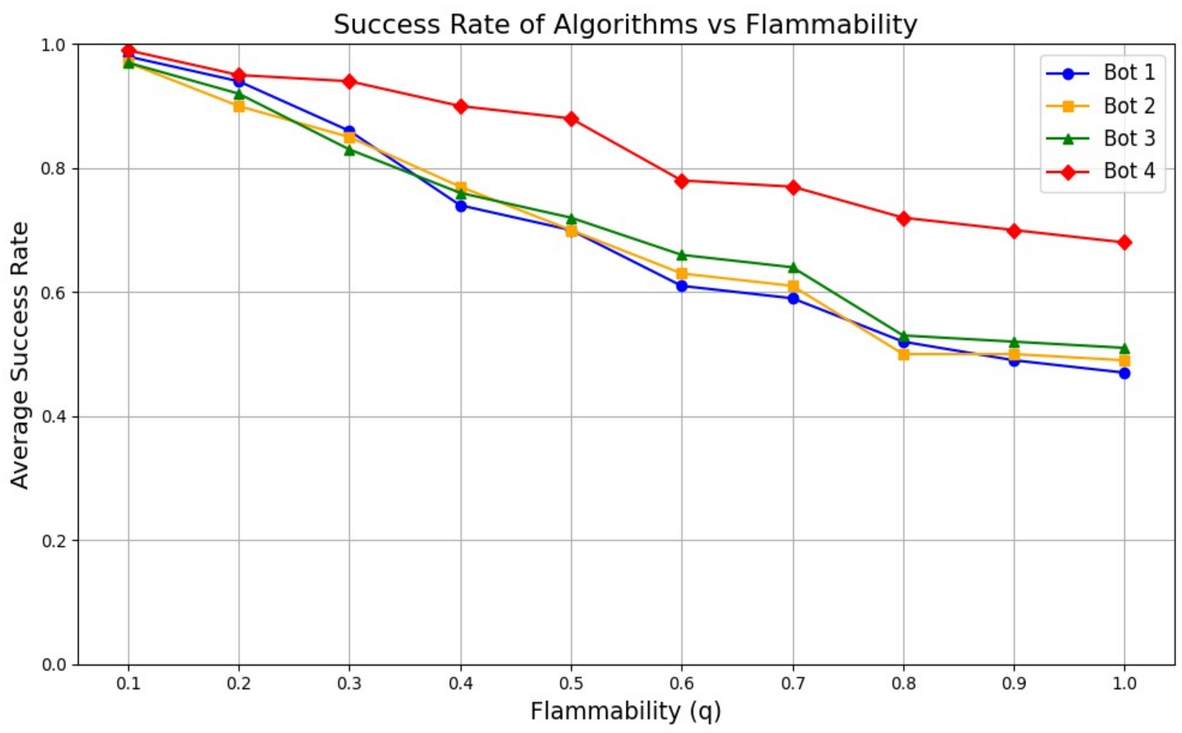

Figure 6: Average Success Rate vs. Flammability Factor (Q)

- In the average success rate graph, all bots show high success rates when the flammability factor(q) is low. However, as the flammability increases, these success rates decline.
- The average success rate for bot 1 declines marginally compared to bots 2, 3, and 4. This is due to bot 1's approach of blindly following the shortest path after avoiding an initial fire cell, without any specific plan to account for the fire's progression.
- The average success rates for bots 2 and 3 are comparable, but for higher values of q, bot 3's success rate increases. This is because bot 3 accounts for both the fire cells and their adjacent cells within the ship as potential threats, allowing it to adjust its next move while still following the shortest path to the nearest fire cells.
- Across several iterations, bot 4 consistently outperformed the other bots, achieving a higher success rate in reaching the button as the flammability (q) increased.
- Bot1 (BFS) quickly identifies viable paths and escapes in low flammability scenarios by exploring neighboring cells level by level; this approach helps it rapidly find safe routes in stable environments with minimal fire spread, allowing it to maintain a high chance of survival —even at q=0.
- The similar success rates of Bot 2 (UCS) and Bot 3 (A*) across all values of q can be attributed to their common focus on finding the shortest path using cost-based exploration. Despite Bot3(A*) employing a heuristic for prioritization, in many scenarios—especially with sufficient open cells—the advantage of this heuristic diminishes, causing both algorithms to explore similar paths.
- Bot 4 excels among all bots due to its refined risk assessment and adaptive navigation. By predicting fire spread and calculating safety corridors, it can dynamically avoid danger zones. This proactive approach leads to higher success rates across various q values, enhancing its survival in unpredictable conditions.

We graph the Average Success Rate against the flammability factor (Q) (0.1,0.2...1.0) **out of the simulations where success was possible**. This allows for a clearer evaluation of their performance, excluding scenarios where success was unlikely due to severe fire spread or challenging grid layouts.

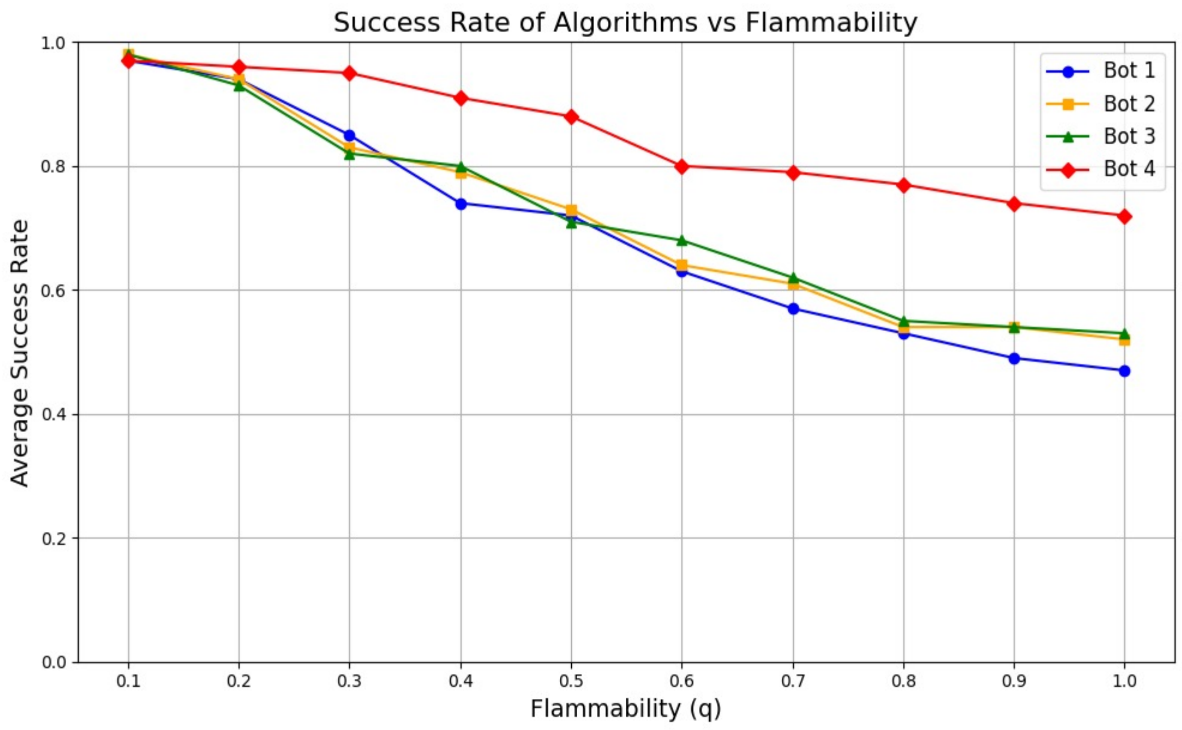

Figure 7: Average Success Rate vs. Flammability Factor (Q) from achievable simulations.

- All four bots show a slight increase in average success rates, especially in high flammability scenarios (q=1), compared to when impossible cases were included.
- Certain failure scenarios result in fire blocking all possible paths in the initial stages. Excluding these scenarios provides a more fair demonstration of the navigation abilities of the bots in more feasible ship layouts.
- Bot3 (A*) can mislead searches due to heuristic inaccuracies (over-estimation), while UCS's cost-focused approach can sometimes perform better. (Ex: q=0.2 & q=0.5)
- In possible scenarios, the bots' ability to navigate around fire becomes more evident, especially benefiting Bot 4, which has predictive fire-avoidance capabilities. Hence, it shows a greater improvement in performance compared to situations where impossible cases were also taken into account.

The comparison between all the bots can be made as follows:

Table 2: Comparison of all 4 Bots

Bot	Heuristic	Look Ahead Capability	Risk Assessment	Proactive Risk Assessment
Bot 1	No	Limited	Nil	No
Bot 2	No	Limited	Nil	No
Bot 3	Yes	Moderate	Moderate	No
Bot 4	Yes	High	High	Yes

3) Why do bots fail or become trapped by fire? Could they have made better decisions to avoid this outcome?

$\rightarrow$ **Bot 1 (BFS):**

• **Why it fails:** Bot 1 explores the shortest path without considering fire dynamics. It makes decisions based purely on distance to the goal, which works in static environments but fails in dynamic ones where fire spreads unpredictably.
• **Could it have made a better decision?:** Since Bot 1 doesn't consider fire spread, it fails by choosing paths that seem short but are unsafe. A better strategy would require fire awareness, which Bot 1's rigid approach lacks, as it only considers the initial fire cell.
• **Failure Example:** The bot may choose a path that becomes unreachable as fire spreads, getting trapped or walking into flames.

$\rightarrow$ **Bot 2 (Uniform-Cost Search):**

• **Why it fails:** Bot 2 seeks the lowest-cost path to the goal, factoring in distance but not anticipating fire spread. It only considers current fire cells and can get stuck on paths that become dangerous.
• **Could it have made a better decision?:** UCS could improve by factoring in dynamic cost changes based on fire proximity. However, it doesn't adjust cell costs based on fire spread, limiting its decision-making.
• **Failure Example:** UCS might choose an efficient path that later becomes blocked by fire, committing to a cost-optimal route without real-time fire dynamics.

$\rightarrow$ **Bot 3 (A* Search):**

• **Why it fails:** A* uses a heuristic considering fire cells and their proximity, outperforming UCS. However, it can still fail due to the unpredictable nature of fire spread. It may not fully anticipate how quickly fire can cut off potential paths.
• **Could it have made a better decision?:** A* could enhance its heuristic or strategy by dynamically updating fire spread predictions and considering future fire spread. The current heuristic may not be aggressive enough in discouraging paths that will become dangerous.
• **Failure Example:** A* may choose an optimal path based on distance and fire proximity but could get trapped if the fire spreads into adjacent cells while en route.

$\rightarrow$ **Bot 4 (Advanced Pathfinding with Fire Prediction):**

• **Why it fails:** Bot 4 is the most advanced, incorporating fire prediction, safe corridors, and risk assessment. However, it can still fail due to unpredictable fire spread or being trapped in risky areas.
• **Could it have made a better decision?:** Bot 4 generally makes the best decisions, but failures often arise when no safe options remain. It may overestimate its ability to avoid fire in risky situations, potentially leading to poor choices.
• **Failure Example:** Despite fire prediction, Bot 4 might get trapped if no safe corridors exist, or if fire spreads unexpectedly. If its fire path predictions are incorrect or it moves too cautiously, it risks being caught by flames.

4) Constructing the Ideal Bot

To build the ideal bot for this scenario, we would focus on creating a system that not only responds to immediate dangers but also proactively anticipates future risks. The bot should use dynamic strategies to adjust to fire spread unpredictability and balance risk-taking with safety. Here are the key components and ideas:

$\rightarrow$ 1. Information the Ideal Bot Would Use:

• *Current Grid Layout:* For a given ship layout - the bot needs to understand which cells are open, blocked, or on fire.
• *Bot's Position and Goal Location:*It must track its current location and the button.
• *Current Fire Spread:* The bot needs to know which cells are currently on fire.
• *Predicted Fire Spread:* The bot should predict future fire spread based on observed patterns, tracking the direction of fire spread. The direction of fire spread can be determined by tracking the fire cells for every step of navigation. Based on the locations of fire cells at current time, the direction in which most cells have burnt, could be considered as probable direction of fire spread for the next step of navigation.
• *Escape Routes:* The bot must continuously assess available paths that remain open and unthreatened by fire for multiple steps ahead.
• *Surrounding Risk Assessment:* It needs to evaluate risks based on proximity to fire and potential future blockages.
• *Fire Spread Rate Estimation:* The bot would need to estimate the flammability factor q based on how fast the fire spreads.

$\rightarrow$ 2. Dynamic Fire Spread Estimation (Probable q):

• *Estimating q On-the-Fly:* Instead of being provided with a fixed flammability value q which is impractical, the bot would calculate a probable value of q by observing how fire spreads across the grid over time. In each iteration, the bot could compute how many new cells catch fire and multiply this by some normalization factor k to get estimated flammability q. k is determined by trial and error for various simulations and identifying which value aligns best with our predictions and actual fire spread.
• *Adjusting Strategy Based on q:* If estimated q is low, the bot might be more aggressive; if high, it would favor safer paths.

$\rightarrow$ 3. Heuristics for Ideal Pathfinding:

• *Dynamic Heuristic Based on Estimated q:* The bot could calculate risk scores for paths based on estimated q.
• *Fire Spread Prediction Over Multiple Turns:* The bot could incorporate multi-turn fire spread predictions into its heuristic. It should consider how far the fire will likely spread over the next few steps and factor that into its decision-making. This could be computed by considering the direction of fire spread and fire clusters.
• *Fire Clusters:* If there is a cluster of cells on fire, we can estimate that X number of adjacent cells to this cluster is most likely to catch fire in the next few moves. The constant X can be determined using the computed probable flammability factor q, and the number of adjacent cells to the fire cluster. It can be computed as:

$$X = q \cdot k \cdot \textit{(no. of cells in the fire cluster)}$$

where $k \rightarrow$ is the normalization factor to avoid disproportionate values that could divert the bot from the path. (X can be used in our heuristic to predict fire spread over several moves.)

$\rightarrow$ 4. Anticipatory Backtracking and Pre-Emptive Actions:

• *Proactive Backtracking:* The bot would evaluate its path for potential dangers and backtrack if necessary.
• *Alternate Path Maintenance:* The bot should maintain multiple candidate routes to avoid getting stuck.

$\rightarrow$ 5. Real-Time Risk Assessment:
• Continuous assessment of surroundings is essential, adjusting plans based on updated fire predictions and risk levels.

Bonus: ***To identify a ship layout that optimizes the likelihood of a bot successfully extinguishing a random fire.***

Algorithm 2 Bonus Graph Creation

```
Initialize a grid with all cells as 'blocked'
for row = 1 to GRID_SIZE do
  if row mod 4 = 1 then
    Mark all cells in row as 'open'
  end if
end for
for column = 1 to GRID_SIZE do
  if column mod 4 = 1 then
    Mark all cells in column as 'open'
  end if
end for
for row = 6 to GRID_SIZE step 6 do
  for column = 6 to GRID_SIZE step 6 do
    Mark three consecutive cells vertically as 'open'
    Mark three consecutive cells horizontally as 'open'
  end for
end for
Add random openings:
for count = 1 to GRID_SIZE * 2 do
  Open a specified number of random cells (num_openings)
end for
Ensure minimum percentage of open cells:
while total open cells >TARGET_OPEN_PERCENTAGE do
  Open 10 random blocked cells
end while
Find and expand dead ends:
Find all dead-end cells with only one open neighbor
for each dead-end in dead-ends do
  Randomly select half of the dead-ends
  Open one blocked neighboring cell
end for=0
```

For bonus, the ship layout is constructed as following:

Grid Initialization: Initialize all cells as 'blocked'- 40x40. Cells are indexed as $grid[x][y]$. This provides a clear starting point for the bot's navigation.

Horizontal Corridors: Every 4th row is marked as 'open' (e.g., $row \mod 4 = 1$). Rows indexed as $1, 5, 9, \ldots$ are opened. This creates defined horizontal pathways for the bot, facilitating easier movement through the grid.

Vertical Corridors: Similarly, every 4th column is marked as 'open' (e.g., $column \mod 4 = 1$). Columns indexed as $1, 5, 9, \ldots$ are opened, increasing connectivity & providing more options for navigation.

L Shaped Blocks: L-shaped blocks are created for every 6th row and column. Cells indexed at $6, 12, 18, \ldots$ are affected. These blocks introduce obstacles that require the bot to navigate strategically.

Random Openings: A specified number of random cells are opened. These random openings contribute to a dynamic layout, enhancing the bot's adaptability in unpredictable scenarios.

Minimum Open Cells: The algorithm ensures at least 60% of cells are open by iteratively checking their status, preventing excessive blockages and maintaining navigable pathways.

Expanding Dead Ends: The algorithm identifies dead-end cells and opens neighboring blocked cells with only one open neighbor, enhancing accessible routes and improving efficiency for the bot.

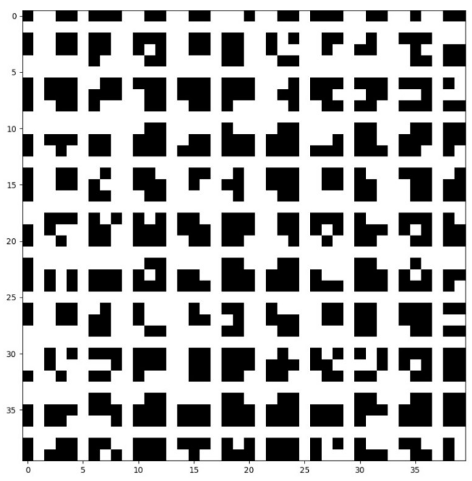

Figure 8: Bonus Grid with L-Shaped Obstacles and Open Corridors

Justification: Why the bonus grid boosts a bot's chances of extinguishing a random fire?

1. Regular open pathways create multiple routes, allowing bots to adapt quickly when fire blocks primary paths to the button.
2. L-shaped blocks improve escape routes for bots by offering non-linear openings, allowing for better movement. This design minimizes the risk of entrapment in straight lines. The fire tends to spread more slowly around corners, allowing bots extra time to escape or explore alternative routes. Additionally, the unpredictable behavior of fire near these corners can lead to better containment, enhancing the bot's ability to traverse the grid.
3. Reduced dead-ends minimize the risk of bots or the button being isolated by fire, increasing overall success rates.
4. Balanced open space allows for efficient bot movement while limiting rapid fire spread, creating a more navigable environment to reach the button.
5. Structured layout with random elements offers predictable corridors for long-distance planning, yet creates unique local configurations that can advantageously position the button or initial fire location.

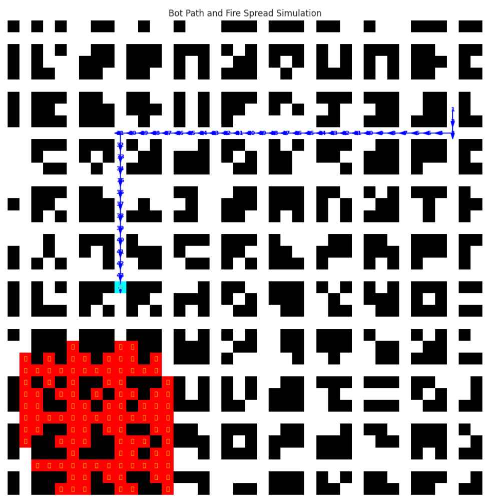

Figure 9: A* Simulation on the Bonus Grid

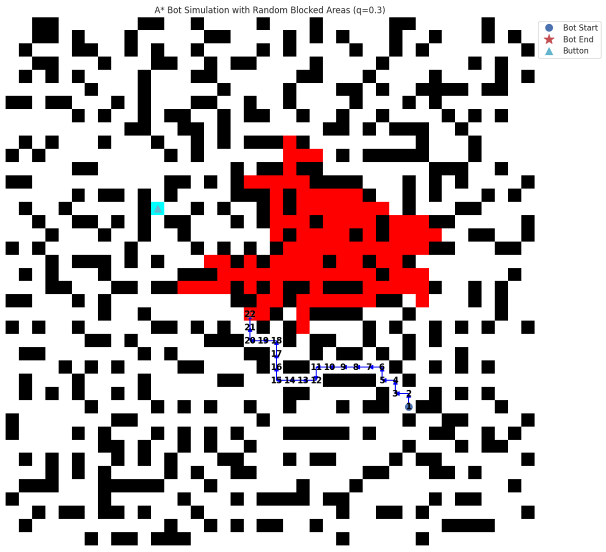

Figure 10: A* Simulation on Project (Normal) Grid

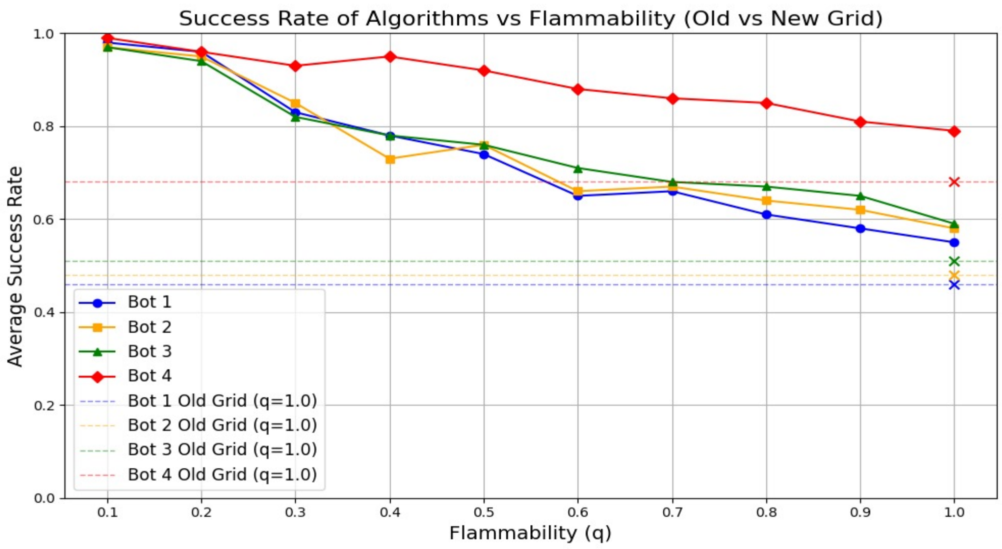

Figure 11: Bot Success Comparison- Project(Old) vs Bonus(New) Grid

Observations: In Figure 9, the simulation of the A* approach at q=0.3, shows that the bot can effectively utilize a long horizontal corridor to reach the button before the fire spreads. In contrast, Figure 10 indicates that the bot is likely to fail with the A* approach, as it must navigate numerous narrow pathways. This increases vulnerability, particularly since the fire often avoids going around blocked cells (Ex:L-shaped block cells), which can further accelerate its spread.

The average success rate plotted against the inflammability factor indicates that all four bots achieved greater success in reaching the button. This suggests that the bonus grid/layout is more effective in maximizing the chances of extinguishing a random fire.

Takeaways for Bot 4:

Our approach as AI engineers and the key takeaways from each aspect bot 4 implementation.

1) How can you account for where the fire is going to be in the future?
In Bot 4, we implemented a fire prediction mechanism through the `predict_fire_spread` function, simulating potential fire spread for a limited number of steps. Model used here employs a simple propagation to adjacent open cells.
Limitation: The prediction is limited to a fixed number of steps (default is 2) to balance future insight and computational efficiency.
Takeaways: Predicting fire spread is essential for proactive decision-making. Simple models offer valuable insights but can be improved. Balancing accuracy and efficiency is key for effective performance.

2) How can you balance avoiding where the fire is going to be vs getting to your goal?
Bot 4 uses a combination of strategies:

a) **Safety Corridor** The `calculate_safety_corridor` function identifies paths that avoid both current and predicted fire locations.
a) **Custom Path** The `bot4_pathfinding` function modifies A* to include: Manhattan distance to the goal, fire avoidance & safety corridor logic.

Takeaways: Heuristic design is crucial for balancing safety and efficiency in navigation. By incorporating predicted states, the bot demonstrates enhanced adaptive behavior in dynamic environments. Additionally, safety corridors effectively guide the bot toward safer paths, ensuring more informed decision-making.

3) When is it better to take a slightly longer path that is farther from the fire vs a shorter path that takes you into riskier territory?
Bot 4 addresses this trade-off through its risk assessment:

- The `assess_risk` function calculates risk based on proximity to fire.
- In high risk situations (>0.7),the bot has a 30% chance to take a risky move and a 70% chance to stay put for safety.

Takeaways: Risk assessment enables more refined decision-making, highlighting that the balance between risk and efficiency is context-dependent. Additionally, incorporating more sophisticated risk-reward analysis could further optimize path choices.

4) How can you identify when you are safe enough that you don't need to worry about avoiding the fire?
Currently, Bot 4 infers safety from:

- Low risk levels from `assess_risk`.
- Presence of multiple safe paths in the safety corridor.
- Distance from the nearest fire or predicted fire cell.

Takeaways: Defining what constitutes "safe enough" is complex and requires careful consideration. Implementing an explicit safety threshold could enhance decision-making, while balancing caution with efficiency continues to be a challenging.

5) How can you make sure that you stay safe?
Bot 4 employs multiple strategies to prioritize safety:

a) Continuous risk assessment using the `assess_risk` function.
b) Safety corridor for path planning.
c) Predictive modeling of fire spread.
d) Decisions made based on risks during simulations.

Takeaways: A multi-faceted approach is essential for ensuring safety, emphasizing that predictive modeling and continuous assessment are crucial components. Additionally, there is an important balance to strike between being overly cautious and taking calculated risks in decision-making processes.

References

[1] Keshav Sharma and Chirag Munshi, *A Comprehensive and Comparative Study of Maze-Solving Techniques by Implementing Graph Theory*, IOSR Journal of Computer Engineering (IOSR-JCE), vol. 17, no. 1, pp. 24–29, Jan–Feb 2015. e-ISSN: 2278-0661, p-ISSN: 2278-8727.

[2] C. Browne and F. Maire, *Evolutionary game design*, IEEE Transactions on Computational Intelligence and AI in Games, vol. 2, no. 1, pp. 1–16, 2010. doi:10.1109/TCIAIG.2010.2041928.

[3] R. Koster, *Theory of Fun for Game Design*, O'Reilly Media, Inc., 2013.

[4] G. K. Sepulveda, F. Besoain, and N. A. Barriga, *Exploring dynamic difficulty adjustment in videogames*, in: 2019 IEEE CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON), 2019, pp. 1–6. doi:10.1109/CHILECON47746.2019.8988068.

[5] E. Valenzuela, H. Schaa, N. A. Barriga, and G. Patow, *Using Search Algorithm Statistics for Predicting Maze and Puzzle Difficulty*, 2024.

[6] A. Hidayatullah, A. Jati, and C. Setianingsih, *Realization of depth first search algorithm on line maze solver robot*, in: 2017 International Conference on Electrical, Electronics, Robotics, and Computing (ICCEREC), pp. 247–251, 2017. doi:10.1109/ICCEREC.2017.8226690.

[7] M. Martín-Nieto, D. Castaño, S. Horta Muñoz, and D. Ruiz, *Solving Mazes: A New Approach Based on Spectral Graph Theory*, 2020.

[8] N. Kumar and S. Kaur, *A Review of Various Maze Solving Algorithms Based on Graph Theory*, vol. 6, pp. 431–434, 2019.

[9] R. Konda, R. Chandan, D. Grimsman, and J. R. Marden, *Optimal Utility Design of Greedy Algorithms in Resource Allocation Games*, in: IEEE Transactions on Automatic Control, vol. 69, no. 10, pp. 6592–6604, Oct. 2024. doi:10.1109/TAC.2024.3375252.

[9] N. Zarayeneh and A. Kalyanaraman, *Delta-Screening: A Fast and Efficient Technique to Update Communities in Dynamic Graphs*, 2021.

[10] Mingyu Xiao, Sen Huang, Yi Zhou, Bolin Ding, *Efficient Reductions and a Fast Algorithm of Maximum Weighted Independent Set*, WWW '21: Proceedings of the Web Conference 2021, pages 3930–3940, 2021.

End of Document