CS 520

Introduction To Artificial Intelligence

Project 1 Report

Ship Fire Simulation and Bot Strategies

Group Members:

- ***Abhishek Jani (aj1121)***
- ***Shrey Patel (sp2675)***
- ***Mustafa Adil (ma2398)***

Contents

1. Introduction

This project implements a simulation of a spaceship environment where different bot strategies are evaluated for their effectiveness in navigating through a potentially burning ship to reach a button. The simulation includes various components such as ship layout generation, fire spread mechanics, and different bot navigation algorithms. The primary focus is to evaluate different bot strategies using various search algorithms to find an optimal or near-optimal path. The fire spread probability varies, creating challenges for the bots, which bots must navigate based on predefined or adaptive strategies.

Objectives

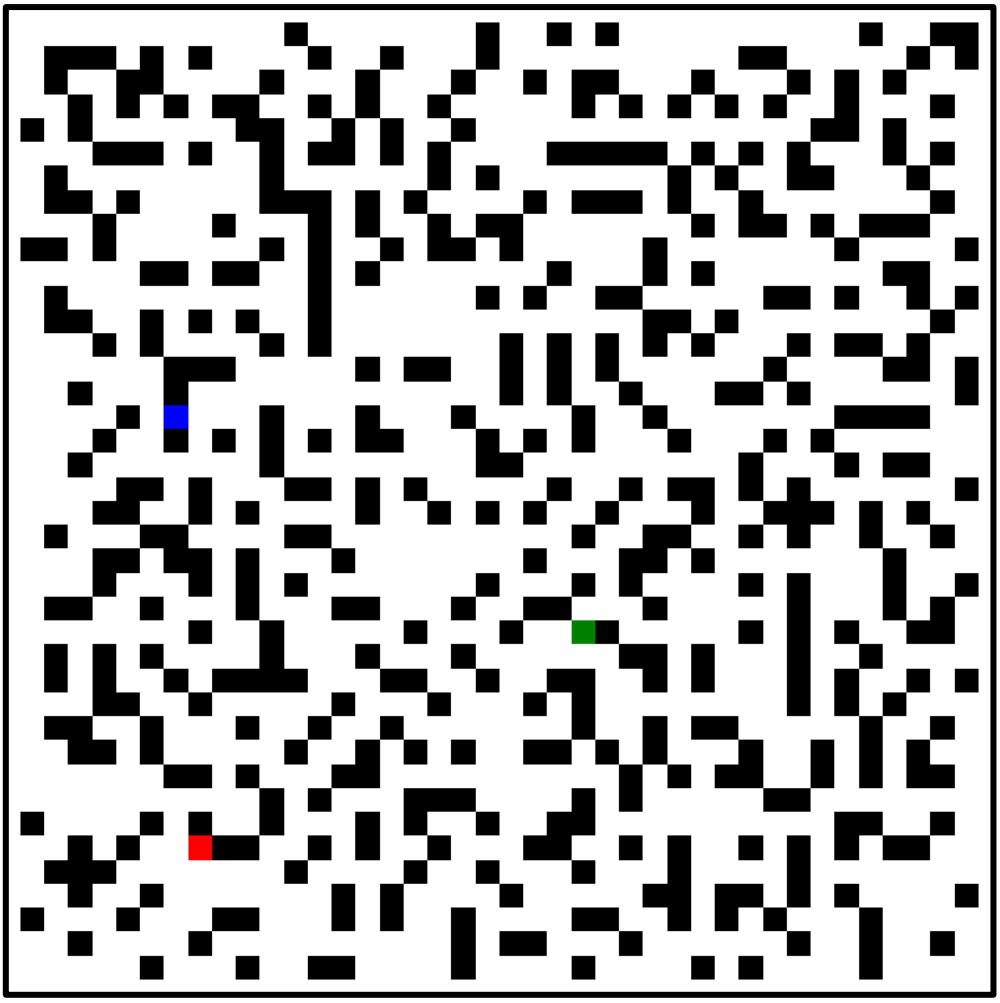
1. Our objective was to create a simulation of the ship, which will be represented by a grid of 40x40 cells.
2. Then we need to build the logic of essential components like fire, button, bots
3. Then we create 4 individual bots with different strategies or algorithms
4. After this, we run the bots for different fire spread from 0 to 0.9 with increments of 0.1
5. Finally, we analyse the success rate to each bot for different fire spread and plot the final graph

2. Project Components

2.1 Ship Environment

The ship environment is represented by a 2D grid where each cell can be in one of five states: BLOCKED, OPEN, FIRE, BOT, or BUTTON. The environment is initialised with a mix of blocked and open cells, creating a maze-like structure. The bot, button, and initial fire locations are randomly placed in open cells. The ship is represented as a grid of size N×N times where:

- The Blocked cells represent walls or obstacles.
- Open cells represent the path that bots can take to reach the button.
- Fire cells represent danger zones where the bot will fail if it moves into them.
- Bots start at a random open position.
- Button represents the goal the bot must reach.



Generated simulated layout of the ship

2.2 Fire Spread Mechanism

Fire spreads probabilistically based on a flammability factor 'q'. In each time step, fire can spread to neighbouring open cells with probability q, ‘q’ can vary from 0 to 1. This creates a dynamic and challenging environment for the bots to navigate.

2.3 Bot Strategies

Four different bot strategies were implemented:

1. Bot 1: Plans the shortest path once at the start, ignoring fire.
2. Bot 2: Re-plans the shortest path at each step, avoiding fire cells.
3. Bot 3: Re-plans the shortest path at each step, avoiding fire and adjacent cells.
4. Bot 4: Uses an A* algorithm with a heuristic that considers both distance and fire risk.

3. Bot Implementation

Bot 1: Static Optimal Path

Bot 1 computes its path from its initial position to the button with the help of BFS algorithm. Bot 1 is simple as it does not take into account the fire spread and ignores it.

- Algorithm: BFS
- Strategy: Plans path once and follows it without considering dynamic fire.
- Advantages: Simple and computationally efficient.
- Disadvantages: Fails if fire spreads across its path.

Bot 2: Recomputing Path with Fire Awareness

Bot 2 is a slightly advanced version of bot 1, such that it takes into account the fire spread and replans its path to avoid fire cells.

- Algorithm: BFS (recomputed at every step)
- Strategy: Replans at each step to avoid fire cells.
- Advantages: Adapts to fire spread, increasing survivability.
- Disadvantages: Computationally intensive due to frequent recalculations.

Bot 3: Avoiding Fire and Neighbouring Cells

Bot 3 improves on the working of bot 2, it actively avoids the cells adjacent to the fire, which reduces the risk of fire spreading into nearby spaces. It also recalculates the possible path at every step while taking into factor the risks associated with the fire.

- Algorithm: BFS with fire and neighbouring fire awareness.
- Strategy: Avoids both the fire cells and the cells neighbouring fire.
- Advantages: Increases survivability further by steering clear of risky zones.
- Disadvantages: Computationally more expensive and may lead to no valid paths if too many cells are marked as dangerous.

Bot 4: A* with Fire Risk Heuristic

Bot 4 is the most advanced bot because it uses the A* algorithm, which incorporates both the Manhattan distance to the goal and the fire penalty for proximity to fire cells. This bot finds the optimal path and minimizes the fire risk by adjusting its heuristic values.

- Algorithm: A* with heuristic (distance and fire risk).
- Strategy: Finds path that balances speed and safety by avoiding fire-prone areas.
- Advantages: Most effective in high-risk environments.

- Disadvantages: Computationally expensive and may struggle in environments with many fire cells.

4. Bot 4 Design and Algorithm

Bot 4 utilizes an A* pathfinding algorithm with a custom heuristic that factors in both distance to the goal and fire risk. This approach allows the bot to make more informed decisions about its path, balancing the need to reach the button quickly with the importance of avoiding dangerous areas.

4.1 A Algorithm*

The A* algorithm is an informed search algorithm that finds the least-cost path from a start node to a goal node. It uses best-first search and finds a least-cost path from a given initial node to goal node.

4.2 Custom Heuristic

The heuristic function used by Bot 4 combines two factors:

1. Manhattan distance to the goal
2. Fire risk penalty

Here is the pseudocode for this :

Purpose: Find the optimal path from the bot to the button while avoiding fire and minimizing risk.

Main Function: Bot4Strategy
Input: Grid layout, Bot position, Button position, Fire positions
Output: Optimal path or None if no safe path exists

1. Initialize:
   - Open Set: Priority queue with (f_score, bot_position)
   - Came From: Dictionary to reconstruct path
   - G Score: Dictionary of current best known path cost to each position
   - F Score: Dictionary of estimated total cost (path cost + heuristic) to goal

2. While Open Set is not empty:
   a. Current = Position with lowest F Score in Open Set
   b. If Current is Button position:

- Reconstruct and return the path

c. For each Neighbor of Current:
If Neighbor is not a blocked cell:
- Path Cost = G Score[Current] + 1
- Fire Risk = CalculateFireRisk(Neighbor)
- Total Cost = Path Cost + Fire Risk

If Total Cost is better than known G Score for Neighbor:
- Update Came From, G Score, and F Score for Neighbor
- Add Neighbor to Open Set

3. If loop completes without finding Button, return None (no safe path exists)

Function: CalculateFireRisk(Position)
Purpose: Evaluate the danger of a given position based on proximity to fire
1. Count burning neighbors (adjacent cells on fire)
2. Return burning neighbors * 0.5 (adjust risk factor as needed)

Function: Heuristic(Position, Goal)
Purpose: Estimate remaining distance to goal
1. Return Manhattan distance between Position and Goal
(Sum of horizontal and vertical distances)

The Manhattan distance provides a basic estimate of the distance to the goal, while the fire risk penalty adds additional cost to paths that are close to fire cells.

4.3 Fire Risk Calculation

The fire risk is calculated based on the number of neighbouring cells that are on fire. Each burning neighbour adds a risk penalty of 0.5 to the cell's cost. This encourages the bot to stay away from fire when possible, while still allowing it to take calculated risks if necessary.

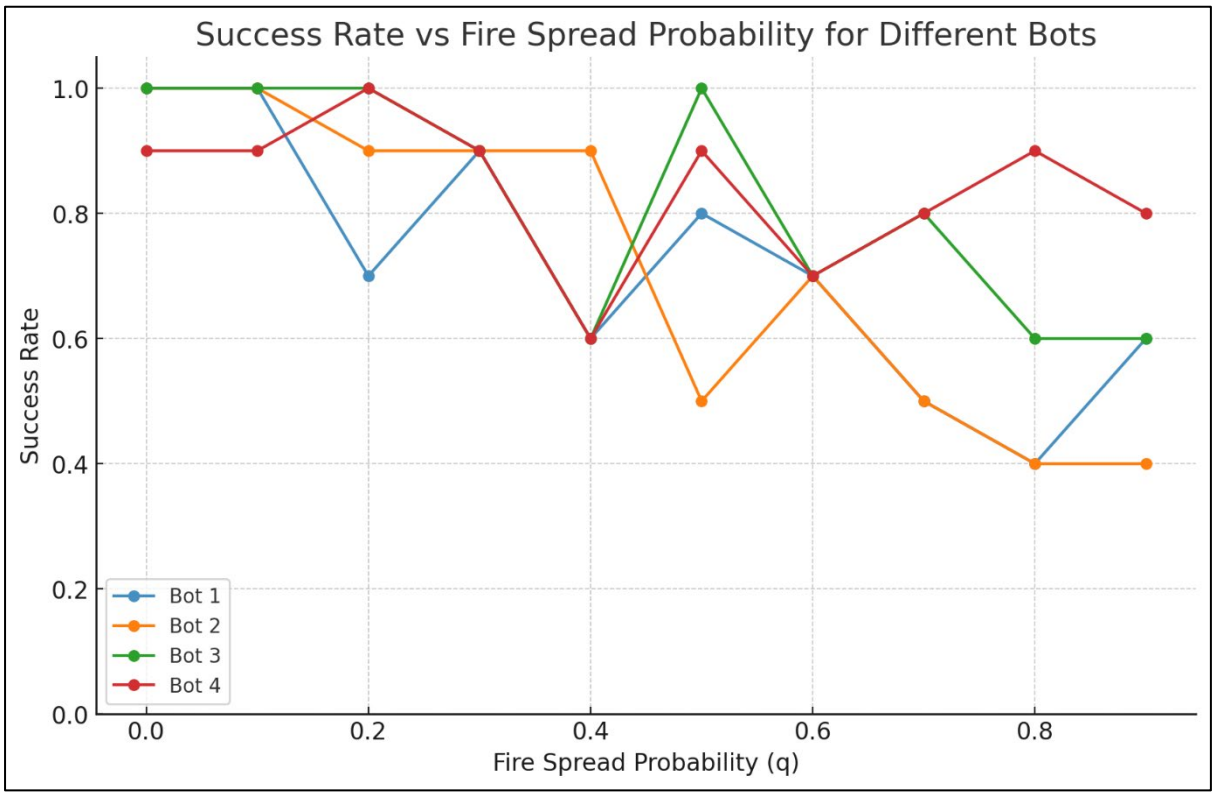
4.4 Path Selection

At each step, Bot 4 recalculates its path using the A* algorithm with the custom heuristic. This allows it to adapt to the changing fire conditions in real-time, potentially altering its course if a safer route becomes available or if its current path becomes too dangerous.

5. Performance Evaluation

To evaluate the performance of each bot, we ran simulations across different fire spread probabilities (q) ranging from 0 to 0.9 in steps of 0.1. For each q value, we ran 100 trials per bot.

5.1 Success Rate Graph



Success rate Vs Fire Spread

The graph shows the success rates of all four bots across different fire spread probabilities. Key observations:

1. Bot 1 (naive approach) performs well at low q values but its performance drops rapidly as q increases. It struggles because it does not adjust its path based on fire spread.

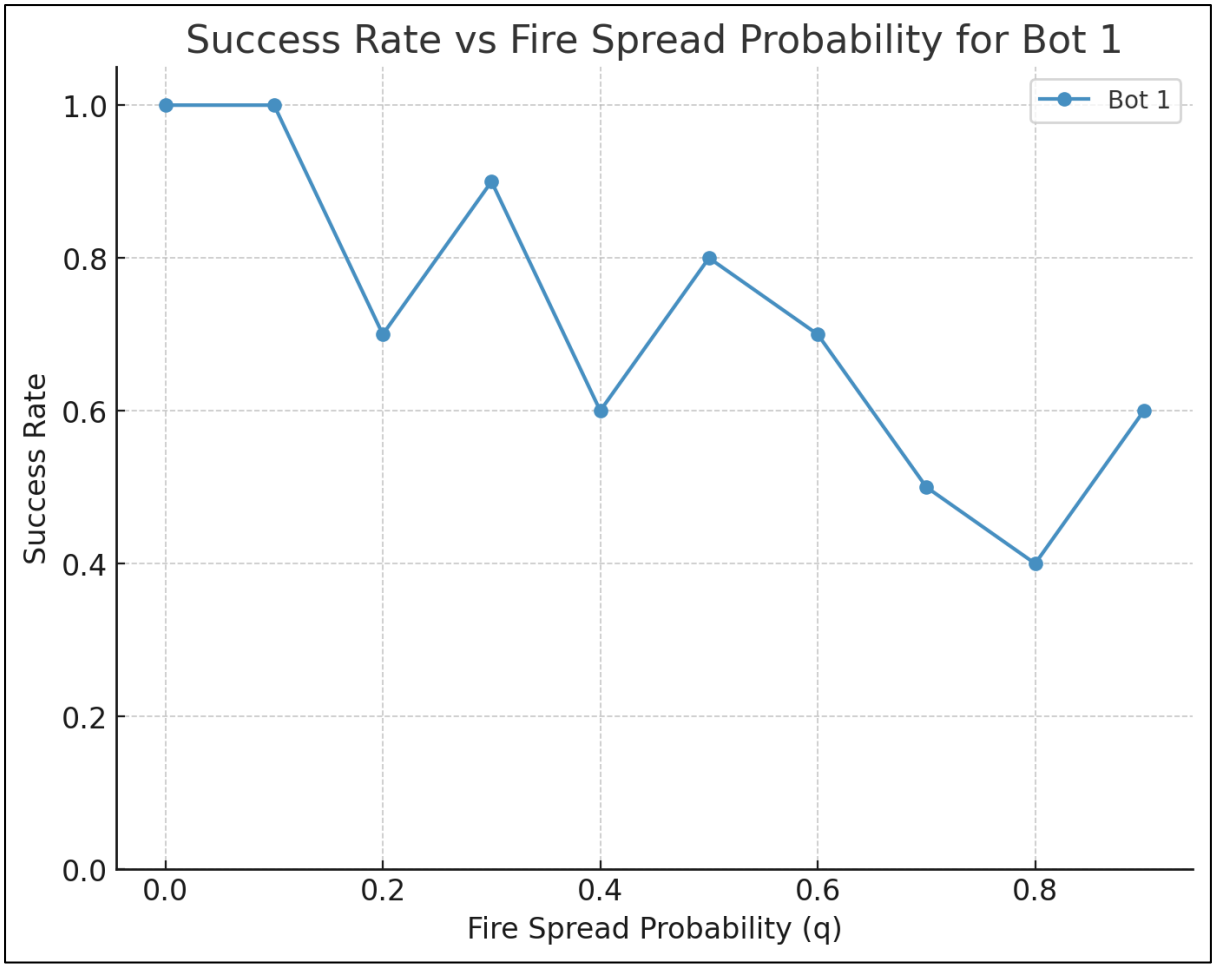
2. Bot 2 and Bot 3 show improved performance over Bot 1, especially at higher q values.

3. Bot 4 consistently outperforms the other bots across all q values, demonstrating the effectiveness of its more sophisticated decision-making process.

6. Analysis of Bot Failures

6.1 Bot 1 Failures

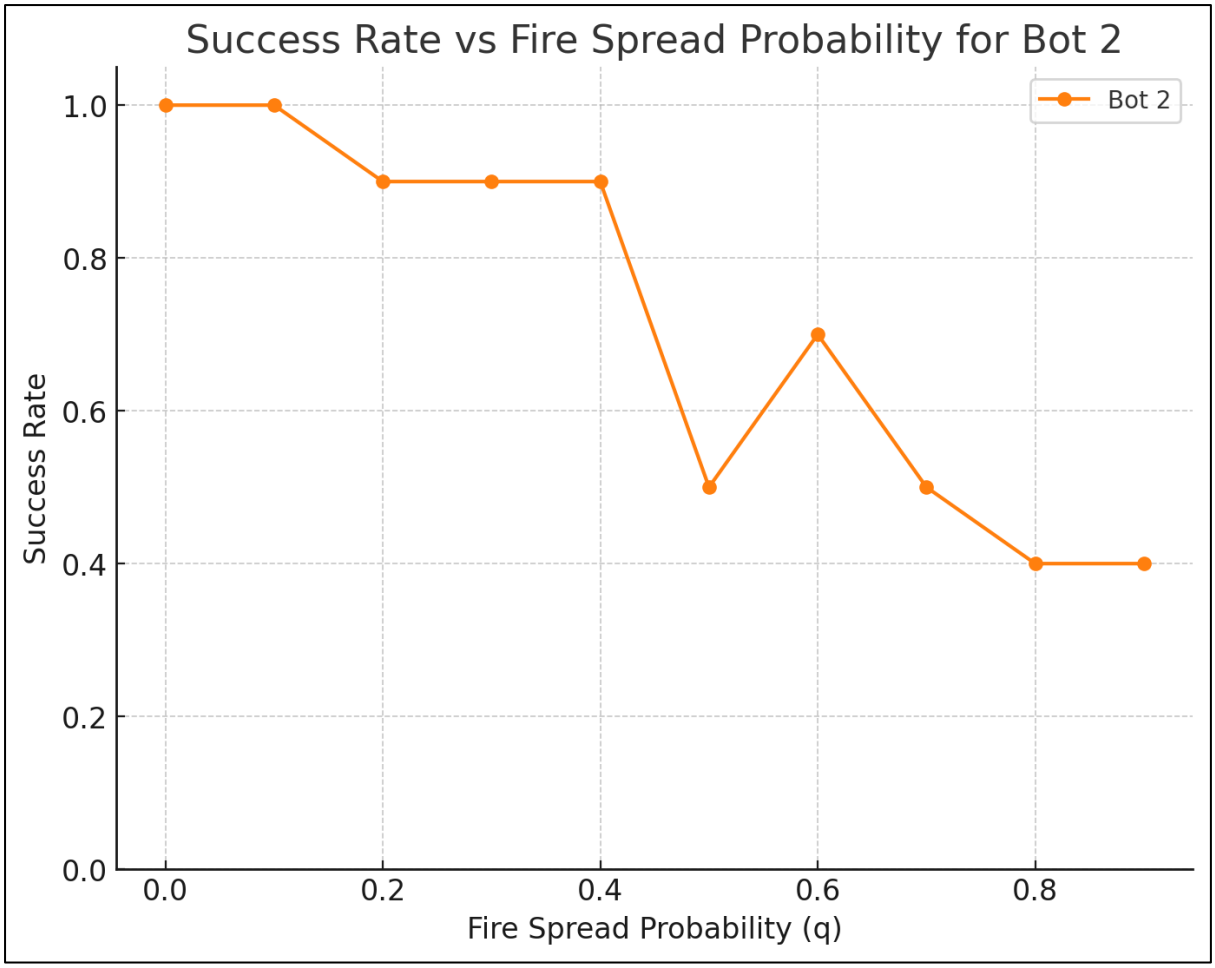
Bot 1 fails because its implementation doesn't take into account for the fire spread at all. It walks directly into fire cells or gets engulfed by spreading fire. A better way would be to re-evaluate its path at each step, like all other bots do.



Success rate Vs Fire Spread for Bot 1

6.2 Bot 2 Failures

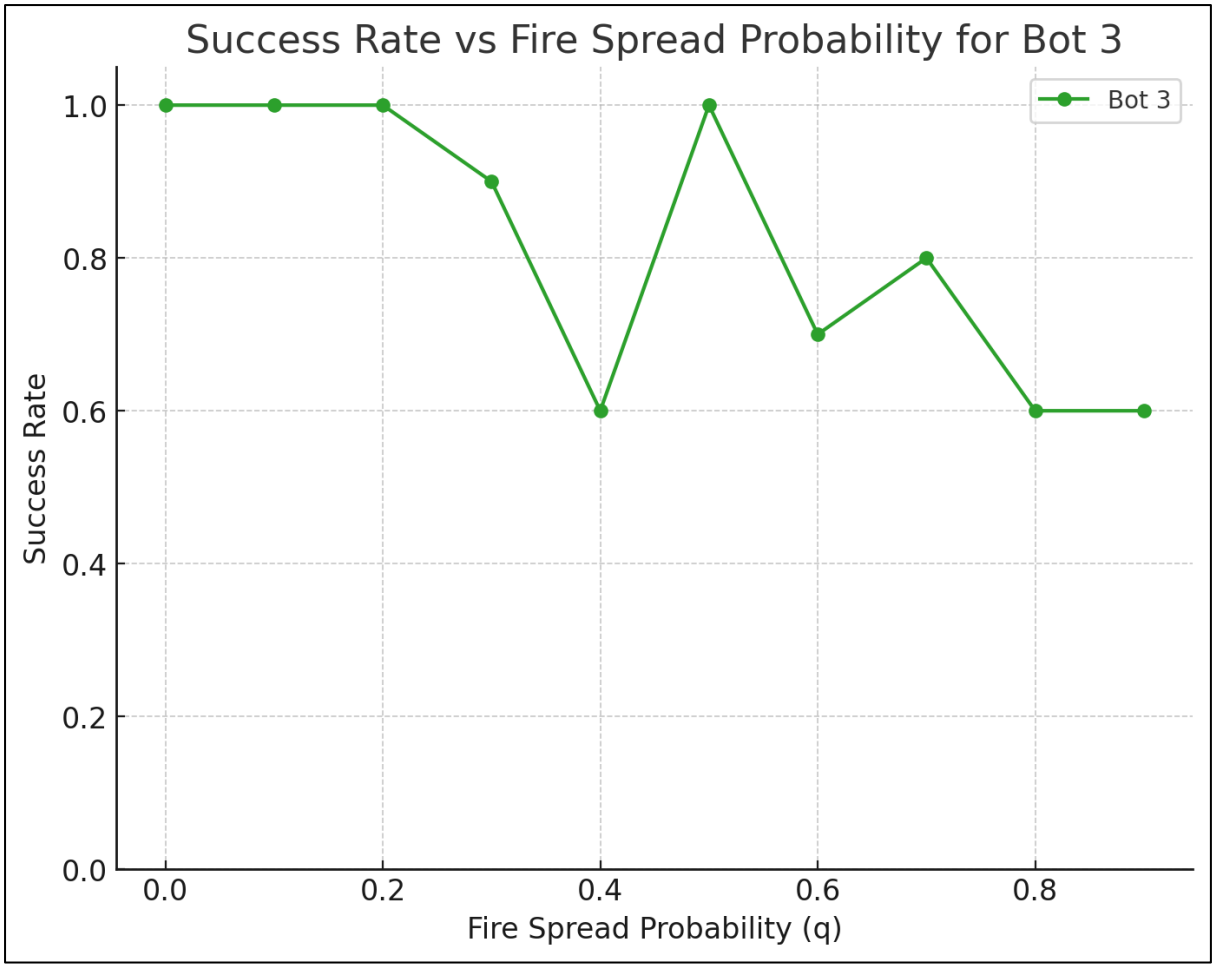
Bot 2 builds and improves upon Bot 1 by avoiding fire cells, but it can still get trapped by fire when fire surrounds it. It could also make better decisions by not only re-evaluating its path at each step but also avoiding cells adjacent to fire, just like Bot 3 does.



Success rate Vs Fire Spread for Bot 2

6.3 Bot 3 Failures

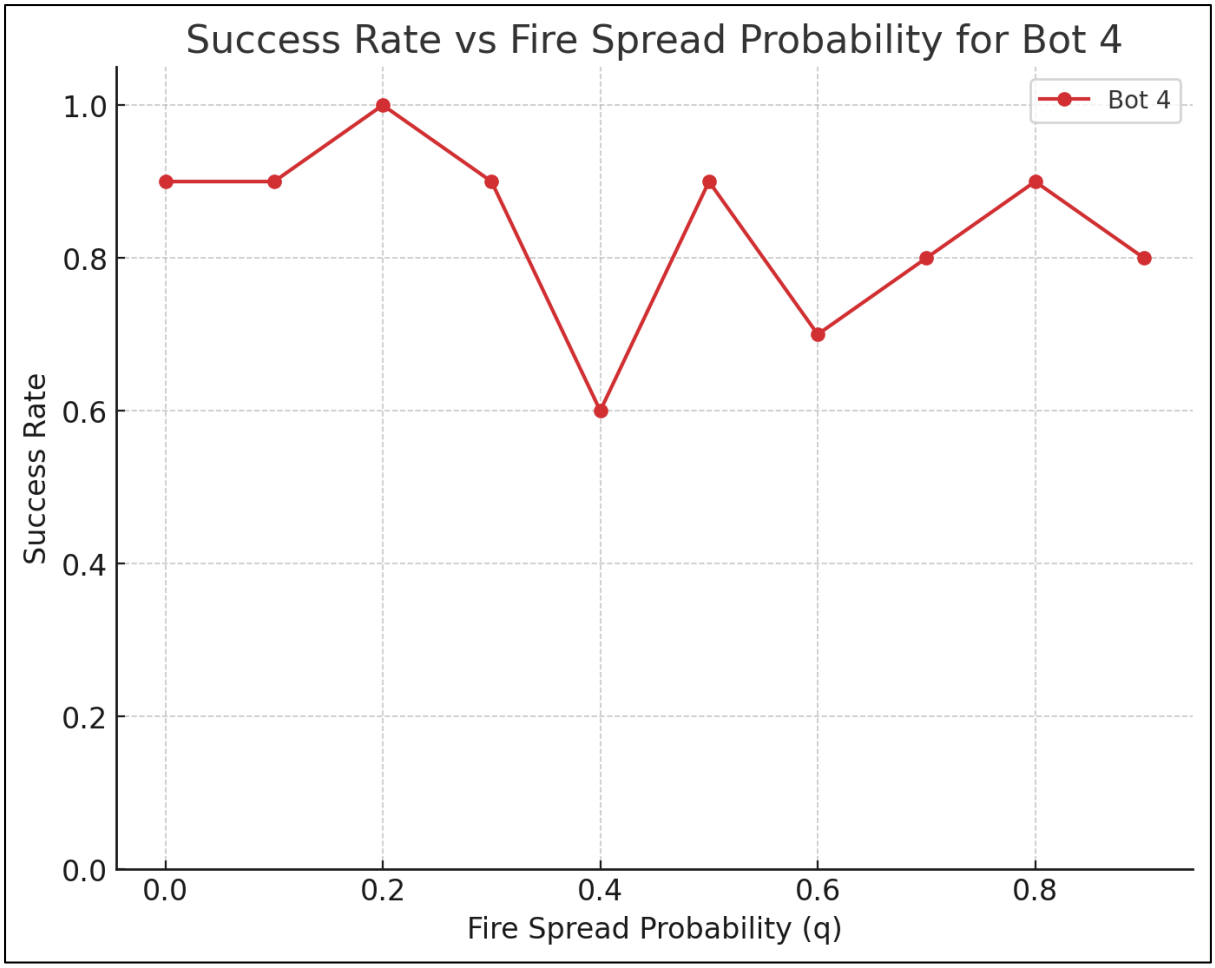
Bot 3 is more cautious and judicious than Bot 2, but it can sometimes be overly cautious, taking longer routes that increases fire spread in the ship. It gets trapped in a situation where all nearby cells are either on fire or likely to catch fire, leading to a deadlock. It can improve and take a better decision by taking a calculated risk through a cell adjacent to fire.



Success rate Vs Fire Spread for Bot 3

6.4 Bot 4 Failures

Bot 4 generally makes the best optimal decisions among all the four bots, but it can still fail in scenarios where fire spreads very quickly (flammability 'q' is very high) or where the initial ship grid layout is particularly challenging and difficult. In some cases, considering the given the information available at each step, there might not have been a better decision the bot could have made.



Success rate Vs Fire Spread for Bot 4

Could They Have Made Better Decisions?

In many cases, no better decision could have been made once the fire had spread to a certain extent or scenarios where fire spreads very quickly (flammability 'q' is very high) or where the initial ship layout is particularly difficult. Bots 1 and 2 could have made better decisions by incorporating future fire risks into their planning, as Bot 3 does. However, even Bot 4, which is the most adaptive, cannot always find a successful route if fire blocks all possible paths.

7. Speculations on Ideal Bot Construction

An ideal bot could potentially improve upon Bot 4's strategy in several ways:

1. *Predictive Fire Modelling*: Instead of just considering current fire cells, the bot could model potential future fire spread based on the known q value. This would allow it the flexibility to avoid paths that are likely to become dangerous in the future.

2. *Dynamic Risk Assessment*: The bot could adjust its risk tolerance based on its distance from the goal and the current fire situation. It might take more risks when it is far away from the goal rather than playing it safe which results in failure.

3. *Dynamic Pathfinding*: The bot should re-calculate and re-evaluate its path after every step.

4. *Multiple Path Consideration*: Instead of committing to a single path, the bot could maintain multiple potential paths and switch between them as required..

5. *Escape Route Preservation*: The bot could also prioritize moves that keeps multiple paths open, which provides it with a escape route in situations where it gets trapped.

6. *Ship Layout Analysis*: The bot could analyse the overall ship layout at the start to identify choke points and wide-open areas, and then take into consideration this information to inform its pathfinding.

To implement these improvements, we would need to:

1. Implement a fire spread prediction algorithm, possibly using Monte Carlo simulations.
2. Develop a more advanced and complex heuristic function that incorporates the above factors/information.
3. Modify the A* algorithm to consider multiple paths simultaneously.
4. Implement a ship layout analysis algorithm, using graph theory concepts to identify critical paths and areas.

8. Bonus: Optimal Ship Layout

To find a ship layout that maximizes the probability of bot success, we can use a genetic algorithm approach:

1. First, we initialize a population of random ship layouts.
2. For each layout, we run multiple simulations with different fire start locations and bot types.
3. Then we can assign a fitness score to each layout based on the average success rate across all simulations.
4. Then, select the top-performing layouts for "reproduction".
5. Create new layouts by combining features of the selected layouts (crossover).
6. Introduce random mutations to maintain diversity.
7. Repeat steps 2-6 for many generations.

Here is the pseudocode for this :

```
Input:
   population_size    // Number of layouts in each generation
   grid_size          // Size of the ship grid (assuming square)
   num_generations    // Number of generations to run
   num_parents        // Number of top layouts to select as parents
   mutation_rate      // Probability of mutation for each new layout


Output:
   best_layout        // The layout with the highest fitness score

Define ShipLayout class
Class ShipLayout:
   grid: 2D array of integers (0 for open, 1 for blocked)
   fitness: float

   Function Initialize(size):
      Create grid of size x size, filled with 0s

   Function Randomize():
      For each cell in grid:
         Set cell to random choice of 0 or 1
      Set start (0,0) and end (size-1, size-1) cells to 0

   Function Mutate():
      Select random cell
      Flip cell value (0 to 1 or 1 to 0)

// Main Algorithm
Function OptimizeLayout():
   population = []

   // Initialize population
   For i = 1 to population_size:
      layout = new ShipLayout(grid_size)
      layout.Randomize()
      Add layout to population

   For generation = 1 to num_generations:
      // Evaluate fitness
      For each layout in population:
         layout.fitness = EvaluateFitness(layout)
```

```
        // Select parents
        parents = SelectTopLayouts(population, num_parents)

        new_population = copy of parents

        // Create new generation
        While size of new_population < population_size:
            parent1, parent2 = RandomlySelect(parents)
            child = Crossover(parent1, parent2)

            If Random(0,1) < mutation_rate:
                child.Mutate()

            Add child to new_population

        population = new_population

        Print "Generation ", generation, " best fitness: ", BestFitness(population)

    Return layout with highest fitness in population

Function EvaluateFitness(layout):
    open_cells = Count cells in layout.grid where value is 0
    Return open_cells / (grid_size * grid_size)

Function Crossover(parent1, parent2):
    child = new ShipLayout(grid_size)
    For each cell in child.grid:
        If Random(0,1) < 0.5:
            Set cell value to corresponding cell in parent1
        Else:
            Set cell value to corresponding cell in parent2
    Return child

Function SelectTopLayouts(population, num_parents):
    Sort population by fitness in descending order
    Return first num_parents layouts

// Run the algorithm
best_layout = OptimizeLayout()
PrintLayout(best_layout)
```

The optimal ship layout has the following characteristics:

1. Multiple branching paths from start to goal
2. Wide corridors in some areas to allow fire avoidance
3. Some narrow corridors to slow fire spread
4. Strategically placed "safe zones" with multiple exits

9. Conclusion

The project demonstrates the complexities involved in pathfinding through a dynamic, hazardous environment where the performances of different bot strategies highlights the importance of considering multiple factors in decision-making processes. Our main paint of focus was the Bot 4's superior performance which showcases the effectiveness of combining distance-based pathfinding with risk assessment.

Building on the current developments, future work could improve upon and involve implementing the forementioned improvements for an ideal bot, and further optimize the ship layout generation, and expanding the simulation to include more complex environmental factors.

Division of Labor

Abhishek Jani : *Ship Layout and Pathfinding*

1. Designed and implemented the ship layout representation
    a. Developed a grid system for representing the ship
    b. Implemented methods for creating, modifying, and validating layouts

2. Created pathfinding algorithms for bots
    a. Implemented A* algorithm for optimal path finding
    b. Developed variations for different bot strategies (e.g., avoiding fire, risk assessment)

3. Optimized pathfinding for performance

Mustafa Adil : *Fire Simulation and Bot Behaviour*

1. Developed fire spread mechanics
    a. Implemented probabilistic fire spread algorithm

b. Created functions to update fire state in each simulation step

2. Designed and implemented bot behaviours
    a. Created base Bot class with common functionalities
    b. Implemented specific behaviours for each bot type (Naive, Avoiding Fire, etc.)

3. Integrated bot movement with the ship layout and fire spread

Shrey Patel : *Simulation Engine and Data Analysis*

1. Created the main simulation engine
    a. Developed functions to run single and multiple simulations
    b. Implemented time step management and event sequencing

2. Designed and implemented data collection mechanisms
    a. Created data structures to store simulation results
    b. Implemented logging functions for various metrics (success rate, path length, etc.)

3. Developed analysis and visualization tools
    a. Created functions to calculate success rates and other key metrics
    b. Implemented visualization tools for ship layouts, bot paths, and fire spread

Collaborative Tasks

1. System Integration: All participants worked together to integrate their components into a single cohesive system

2. Performance Optimization: Collaborated on identifying and resolving performance bottlenecks

3. Testing and Validation: Designed and conducted comprehensive tests to ensure accuracy of the simulation