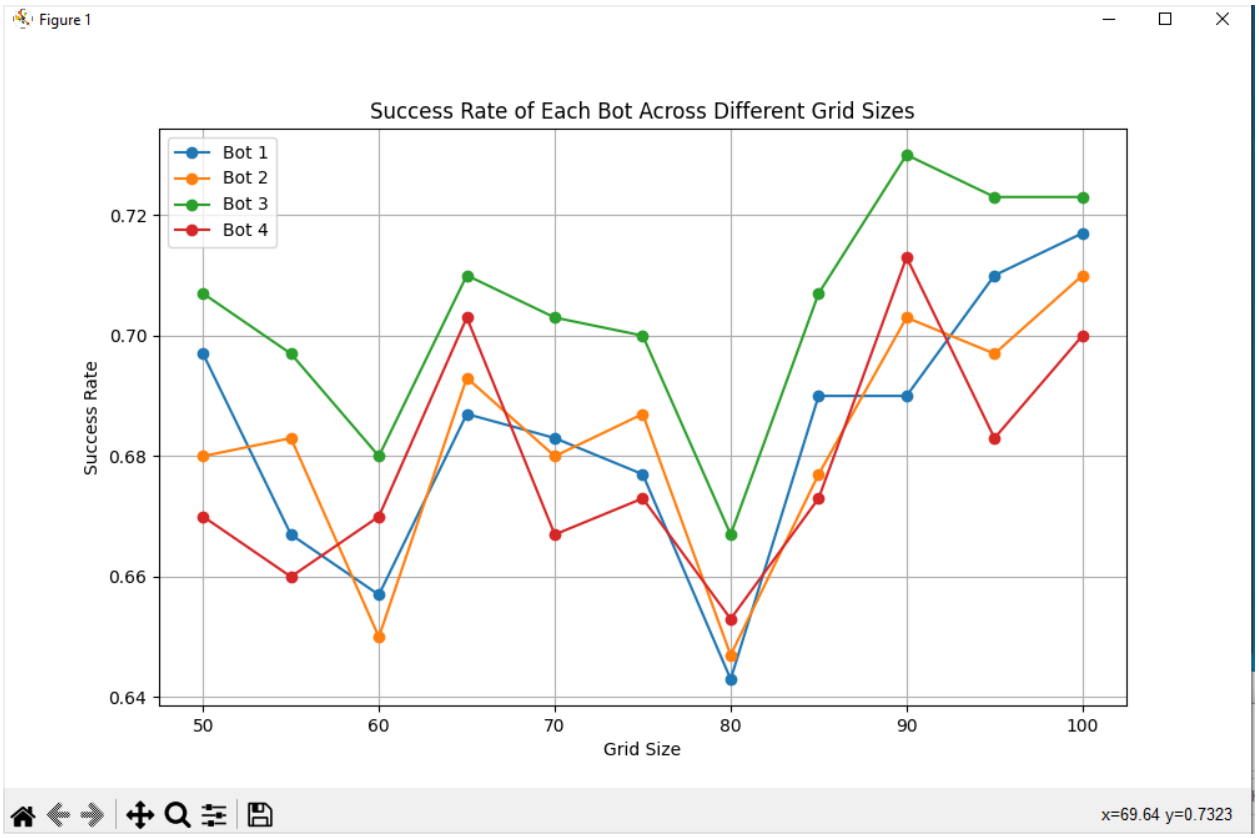
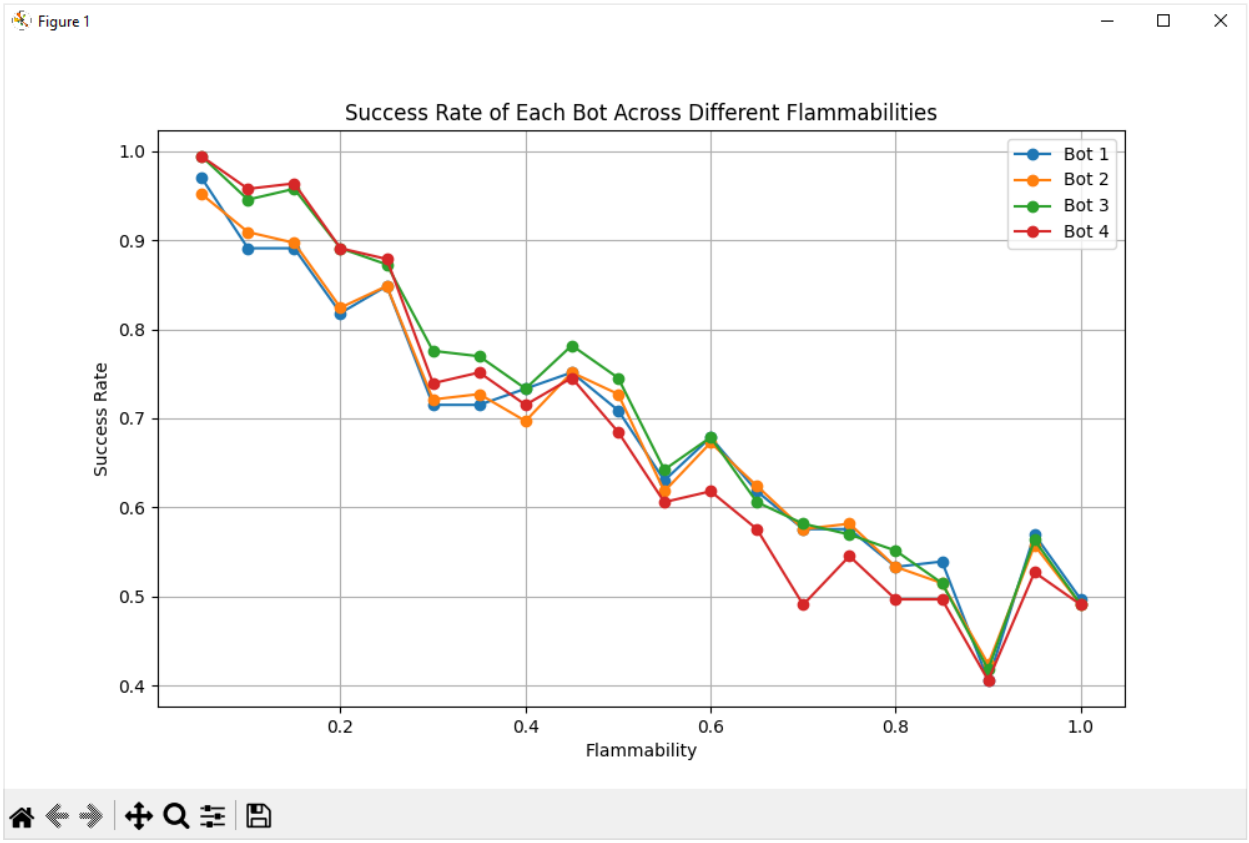
Project 01 - Write Up

Diego Castellanos
Dac392
01:198:440

Summary and Data Set Overview
For my implementation, I built a ship_t class which would maintain the main game state and would take care of much of the programs information and execution. This way, I would be able to run the same game for all 4 bots and be able to better compare each bot to each other as they are playing out similar games. After each bot would finish its execution, the game board would be reset and the next bot would be able to play out its game. Therefore, my dataset is comprised of 13,214 records, with 3,301 unique ship IDs for 4 unique bots. I ran my experiments on 12 distinct grid sizes ranging from 50x50 up to 100x100 and 20 unique flammabilities ranging from 0.05 up to 1. My program primarily utilizes A* for its path finding calculations with the main heuristic being the Manhattan distance from point A to point B

Results
- The vast majority of tests were successful (9,071), while 4,129 tests were unsuccessful.
- The reasons for failure varied from
  - “goal caught on fire” (2,315 times),
    - Bot 1: 427 times
    - Bot 2: 433 times
    - Bot 3: 633 times
    - Bot 4: 822 times
  - “bot caught on fire” (1,750 times),
    - Bot 1: 602 times
    - Bot 2: 599 times
    - Bot 3: 327 times
    - Bot 4: 222 times
  - “bot could not find a path to the goal” (64 times)
    - Bot 1: 16 times
    - Bot 2: 16 times
    - Bot 3: 16 times
    - Bot 4: 16 times





Bot 4 Approach

My approach to bot 4 was to generate simulated games at each time step and recalculate the shortest path based on the average fire spread from the simulated games. The a number of simulations would be run to x many time steps and averaged to attempt to obtained a good enough estimate in order to make a better move. Thus, for each real time step in the game, the bot would recalculate the best path to the goal by running x number of simulation each made up of t number of time steps. This allowing the bot to avoid making moves that would likely result in him getting caught by the fire. This is certainly reflected by the results, as in my experiments, but 4 was caught by the fire the least number of times, however the times it was unsuccessful were as a result of the fire spreading to the goal faster than the bot was able to get to the goal.

For my experiments, I chose to calculate 15 simulations made up of 3 time steps each, as I thought this would be the best option as it ran at an acceptable speed that did not seem too long and would provide me a somewhat good enough estimate. However, I would have preferred to use a much higher simulation size of 50+ to get a more accurate estimate and perhaps a more successful bot 4. For bot 4, I am primarily using A* for my path finding calculations, however, I am estimating the length to the goal as MD() + SP - DZ() where:

- MD is Manhattan_distance(neighbor, goal): the Manhattan distance of the possible move and the goal
- SP is Simulated_probability(neighbor): simulated probability of the possible move being on fire in the next few time steps
  - < flammability * simulated_probability * 100 >
- DZ is distance_to_safe_zone(neighbor, simulation): calculating the distance to a safe zone in case that the bot chooses to move into a position that can catch on fire

Performance Metrics

The average success rate of the bots, when considered across all grid sizes, showed that bot 3 had the highest average success rate of 70.3%, followed by bot 4 at 69.2% bot 2 at 68.1% and bot 1 at 67.5%. Overall, bot 4 was most successful for q < 0.5, however, was less successful as a higher flammability meant that the bot would be more avoidant than bot 3. This meaning bot 4 would take too long to reach the goal and the goal would be caught by the fire.

Bots 1 and 2 fail most often due to getting caught by the fire as they are attempting to get to the goal as fast as possible with minimal interest in the fire. In these cases, there are plenty to better moves that they could have made. Bots 3 and 4 on the other hand, fail most often due to the goal catching on fire. These bots are more interested in self preservation that they are on getting to the goal as quick as possible. This was likely my main failure in implementing bot 4 as I placed too high of a priority in

self preservation and not as much importance in trying to reach the goal as quickly as possible. A possible fix would be ease the restrictions on what to avoid as for my experiments I decided to avoid any sorts that had a a probability of catching fire higher Tham 0.3.

Speculation

The ideal bot would most likely be one that could have as much information as possible. This would likely be one that could simulate many games to completion in order to find the best path. With my machine I was only able to run a minimal amount of simulations for minimal amounts of steps, however by increasing the number of simulations along with the number of steps each simulation is calculated up to, the bot would likely be able to find a much more accurate estimate and a much more accurate best path to take. A seemingly perfect approach could potentially also take closed cells into account as they would stop/slow down the fire meaning that the bot could possibly find a paths that are slightly more strategic