Project 1 – 440 Fall 2023

Professor – Dr. Cowan

Students – Kush Patel(kp1085), Pavitra Patel(php51), Huzaif Mansuri(htm23)

Q. Explain the design and algorithm for your Bot 4, being as specific as possible as to what your bot is actually doing. How does your bot factor in the available information to make more informed decisions about what to do next?

Ans: Our bot 4 is implementing an A* Search Algorithm to get the optimal path at each time step. In our A* algorithm, we are using a heuristic function that calculates Manhattan distance from node of interest to goal position. We also implemented a cost function in our A* algorithm that helps determine the best neighbor to pick when the bot is at a given position in the grid. This cost function determines the weight that will be given to each neighbor depending on the factors defined below.

Description of the cost function:

$$cost = (D * minPB) - (1.2 * minPF)$$

Here, minPB is the minimum distance of the Player to Button, minPF is the minimum distance of the Player to Fire, and D is the size of the grid. The goal of the cost function is to do the following:

Distance between Player to Fire decreases → Cost Increase

Distance between Player to Fire increases → Cost Decrease

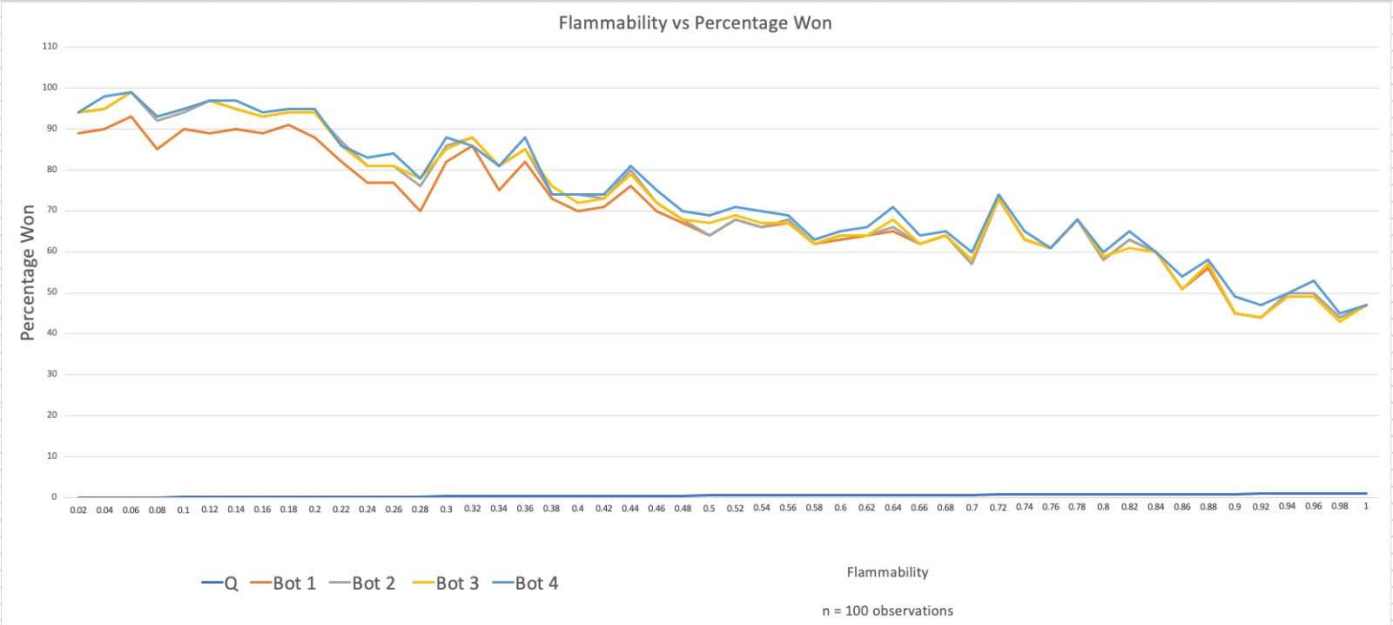
Distance between Player to Button increases → Cost Increase

Distance between Player to Button decreases → Cost Decrease

We imply that minPB is directly proportional to the cost whereas minPF is inversely proportional to the cost. D is a constant value that scales up our minPB as per the grid size to provide the bot with some heuristic on how much importance should be provided to the distance between Player and Button. Similarly, 1.2 is a constant that scales our minPF to provide the bot with some heuristic on how much importance should be provided to the distance between Player and Fire.

Q. For each bot, repeatedly generate test environments and evaluate the performance of the bot, for many values of q between 0 and 1. Graph the probability (or average frequency, rather) of the bot successfully putting out the fire. Be sure to repeat the experiment enough times to get accurate results for each bot, and each tested value of q.

Ans:



Grid Size Tested On is 50*50

Q. When bots fail or get trapped by the fire, why do they fail? Was there a better decision they could have made that would have saved them? Why or why not? Support your conclusions.

Ans:

In the case of Bot 1, it only knows the initial fire cell and calculates the path once it starts. Hence, the path for Bot 1 can be defined as blind and doesn't make smart decisions. As a result, Bot 1 steps on fire and fails. Bot 1 could have been better by knowing where the current fire is spreading.

In the case of Bot 2 and Bot 3, they know where the current fire cells are. Additionally, Bot 3 knows the neighbor cells of the fire. Both bots recalculate the paths at each time step. As a result, the major problem that Bot 2 and Bot 3 faced is that they kept avoiding the fire and got trapped resulting on no path possible at some point. Bot 3 works as Bot 2 when it doesn't find a path. Bot 2 had no idea how risky it is to go near fire. The second reason why Bot 2 and Bot 3 fails is that they went near fire and got trapped by fire. One thing we could do is introduce some weight to neighbors depending on how far they are from the goal and fire. If a neighbor is near fire, we can give it a high weight which the Bot will avoid and pick the other neighbor which has low weight. Similarly, if a neighbor is near goal, we can give it a low weight which the bot will pick and will avoid the other neighbor which has high weight. Finding a path from low weight neighbor would be deemed safe.

In the case of Bot 4, I am using A* Search algorithm with a heuristic and a cost function that determines the best neighbor as mentioned above. The problem that Bot 4 didn't solve is that in rare cases it would avoid going near to fire hence resulting in no path at some point. The

heuristic calculates Manhattan distance, and the program only gets rid of approximately 50% of the dead-end cells, some traps were caused due to dead end cells and the fire covering the only path possible. Instead of using Manhattan distance for heuristic, Bot 4 could have used BFS which would result in not getting trapped by dead-end cells.

Q. Speculate on how you might construct the ideal bot. What information would it use, what information would it compute, and how?

Ans: Our ideal bot would act smartly by calculating probabilities of fire at each path possible from its current position to the goal. The probabilities of fires at each cell can be calculated using the current outer fire cells in the grid at each time step. Using this probability, we can then find the probability of all available paths (Not only BFS) from the bot's position to the button. Then we can implement a best path factor that would be inversely proportional to the probability of the fire and path's length. This will help us determine which path is shorter and less risky for our bot by taking into consideration the distance between Player to Bot and Player to Fire at each time step. This would make our ideal bot smarter by using all available information.

Group Contributions:

Pavitra Patel:

1. Ship Layout
2. Opening 50% of dead-end cells
3. Helper methods: Outer Fire cells
4. Task for Bots and storing Fire for all Bots
5. Bot 1
6. Report

Huzaif Mansuri:

1. Initializing fire cell, button cell, bot cell on grid
2. Bot 4
    - A* Search Algorithm
    - A heuristic
    - A cost function
3. Report

Kush Patel:

1. Fire
2. BFS

3. Helper methods for BFS and Fire
4. Bot 2
5. Bot 3
6. Report