Kevin Chen, Demetri Ioakimidis, Vian Miranda
Professor Cowan
Introduction to Artificial Intelligence 440
15 November 2023

Kevin
- Developed bots 2, 6, 9
- Collected data for bots 2, 6, 8, 9
- Brainstorm bots 2, 4, 6
- Seed generation

Demetri
- Developed bots 3, 7, 8
- Collected data for bots 3, 7, 8, 9
- Probability update calculations for probabilistic bots
- Board and distance generation

Vian
- Developed bots 1, 4, 5
- Collected data for bots 1, 4, 5, 8, 9
- Dijkstras and BFS
- Graph generation

1. Explain the design and algorithm for the bots you designed, being as specific as possible as to what your bots are actually doing. How do your bots factor in the available information (deterministic or probabilistic) to make more informed decisions about what to do next?
    a. Bot 1 - Bot 1 has two choices to make at each timestep, either to sense or to move. At the very beginning, the bot will sense (scan the detection square of size $(2k+1)^2$ for the leak) as it has no data. Once the bot scans the detection square, there are two possibilities:
        i. The leak is in the detection square: In this case, the bot will use BFS (Breadth First Search) to find the closest unchecked leaky cell. An unchecked leaky cell is defined as one that has not yet been visited by the bot but was marked as leaky by the detection square. This repeats until the leak is found, upon which the bot returns the path it traversed and the time it took.
        ii. The leak is **not** in the detection square: In this case, the bot will mark all of the cells in the detection squares as “not_leaky”, which means we don’t need to check these cells for the leak again (they are all essentially visited in this one scan). Through this one scan, we leave the detection square to

the closest unchecked cell to repeat our scan. This is repeated till we detect the leak.

iii. Note that, as k increases (toward the ballpark of ~8), performance starts to decrease. This is because the detection square becomes very large, and the information it gives us is less valuable. For example, assuming we detect a leak in a completely unseen detection square at k=8, then we would have to comprehensively search a 17x17 square to find the leak, which takes a lot of timesteps to do.

b. Bot 2 - Bot 2 is similar to Bot 1 in concept with one crucial modification.
   i. Instead of traveling to the nearest unsensed cell, we run a full BFS search on the graph and look for the cell with the highest heuristic "score".
   ii. To compute the score of a given cell (i, j), we need to keep track of two things: the number of unsensed open cells in a hypothetical detection square centered at (i, j) as well as the length of the shortest path from the current position to cell (i, j). We use nested for loops to check if each cell in this hypothetical detection square is in "unseen" and if it is, then we increment the counter. Thus, the score is computed using the following formula: $score = \#(new\ cells\ scanned) * (ratio)^{dist\ to\ (i,\ j)}$. We then sense once when we reach our optimal cell to sense at (which is determined by the formula).
   iii. Initially, we were going to simply determine the score for a cell (i, j) using the number of new cells that would hypothetically get sensed at (i, j). However, this could make the bot travel to the unexplored corners of the ship and take unnecessary detours. Thus, we added the exponential decay term to penalize bots who want to travel the world and prevent them from doing so. In our results, we found that this yields marginal improvements upon bot 1.

c. Bot 3 - Probabilistically Finding a Single Leak
   i. Bot 3's knowledge base is represented as a 2D array, where *array[i][j]* contains the *P(Leak in cell (i, j))*. The bot operates using information gathered from 3 events: *Leak not found in cell (i, j), Beep heard in cell (i, j), Beep **not** heard in cell (i, j).* At every time step, the bot takes one of two actions: **sense** or **move**.
   **ii.** **Sense**: If the bot chooses to *sense*, it listens for a beep coming from the location of the leak. Depending on the result of this sense, probabilities are updated. The exact math for this is detailed below, but the Bayesian Probability Update computes *P(Leak in cell (i, j) | Beep/No Beep in cell (k,*

*l)* where *(k, l)* is the bot's current position. This probability is computed for all cells *(i, j)* in the grid that are open.

iii. **Move**: If the bot chooses to move, it first plans it's path. It does so by iterating over its current knowledge base and planning the ***shortest path*** to this cell from its current position. At each step it takes along this path, it checks if the cell it's on has the leak. If it does, the bot concludes its search. If it does not, another Bayesian Probability update must be done for all open cells. In this case, the probability *P(Leak in cell (i, j) | No leak in cell (k, l))* where *(k, l)* is the bot's current position, but be calculated for all open cells (*i, j)*. A significant point about this bot is that it does not replan it's path until it reaches its destination, meaning the updates that are done along the way are not accounted for until the entire path is traversed. This means that the bot can (and will) frequently travel along previously visited cells, or cells where *P(Leak in cell) = 0* when moving along the ***shortest path***.

d. Bot 4 - Bot 4 is similar to Bot 3 in concept, with a couple of modifications.

i. **Path Planning:** To plan the path, Bot 3 uses BFS to find the shortest path to the highest probability cell; however, Bot 4 takes advantage of the Dijkstra algorithm to find the path of the greatest probability. This is calculated by averaging out all the cells on a path and minimizing the number of already visited cells it takes to reach the best cell. It then uses a maxHeap to take the path of the greatest average probability to the best cell, giving it a chance to find the leak on its way to the best cell as well.

1. Once the Dijkstra path is computed, it also computes a BFS path like in Bot 3. It then compares the average probability of the BFS path to the Dijkstra path. If the BFS path has a higher average path probability than the Dijkstra path (which is possible as Dijkstra avoids taking visited cells), then it would be more beneficial to take the BFS path; thus, the path of greater average probability is taken.

ii. **Sense:** Bot 4 senses in two distinct situations: every (1% of # of open cells) moves *and* once the bot reaches the goal state

1. Every *n* steps where n is defined at (1% of the total # of open cells, which in our case happened to be roughly 5), we sensed for a beep again. Based on what we heard, we updated our probabilities for the cells again. If the updated probabilities pointed that there was a new best cell (compared to our previous one), we planned a new path to move towards this new best cell.
2. Once the goal state is reached, we scan twice should the first beep be heard (return true). If the first beep is false, we assume that the leak is further away and don't waste another timestep scanning;

however, if it is true, we scan again to check how close the leak could be. If the second time is false, it could be possible that the first beep was lucky or the leak is not close enough. However, if we hear it a second time, there is an assumption that the leak is close and we plan accordingly.

e. Bot 5 - Bot 5 is the same as Bot 1, except it continues on to search for the second leak once the first leak has been detected and plugged. This bot just repeats the process of Bot 1 while searching for this second leak, except now it is easier as more cells have already been checked.

f. Bot 6 - Bot 6 is very similar to bot 2, except it continues on to search for the second leak once the first leak has been found and plugged. This bot just repeats the process of Bot 1 while searching for this second leak, except now it is easier as more cells have already been checked.
    i. **NOTE**: bot 6's ratio parameter is set at .97 instead of .99 such as with bot 2. The intuition behind this is that, the closer a ratio is to one, the closer the score is to simply the number of new squares seen. Thus, the closer a score is to 1, the greedier it is. As per the problem description, we have two leaks to deal with in the case of bot 6. Thus, from a pure numbers standpoint, the ship is denser in terms of the number of leaks it has, so we should be more careful so as to not miss slivers of cells we might have missed by using a greedier approach to zig-zagging to each optimal cell. From an adversarial standpoint, the adversary has more "ammunition" to place the leaks in undesirable places for the bot, so we need to make sure we are not cutting corners when sensing, and thus need a ratio parameter value that tends more toward "exploration" than the greedy "exploitation" of bot 2's ratio=.99.
    ii. What happens after you find the first leak?
        1. After we find the first leak, we proceed with the same process to find the second leak. Since we have (in a typical case) scanned many cells, there are less cells to scan, so it should not take as long to find the second leak.
    iii. How do you decide where to move to next?
        1. We use the score formula described for bot 2. However, as previously mentioned, we use a ratio parameter of .97, which is less greedy than that of bot 2.
    iv. How do you decide when to trigger the detection square?

1. We use the score method to decide when to trigger the detection square, since it gives an "optimal" (from the standpoint from our heuristic) cell to scan at. In our results, we found that this yields marginal improvements upon bot 6. Note that the results for bots 5 and 6 converge as k increases, since the detection square becomes so large that it no longer gives valuable information. Moreover, since the detection square covers much more of the ship, we will have to scan less times, so the only difference between bots 1 and 2 (in how they choose their "optimal" squares to scan at) is activated less.

v. Note that, as k increases (toward the ballpark of ~8), performance starts to decrease. This is because the detection square becomes very large, and the information it gives us is less valuable. For example, assuming we detect a leak in a completely unseen detection square at k=8, then we would have to comprehensively search a 17x17 square to find the leak, which takes a lot of timesteps to do.

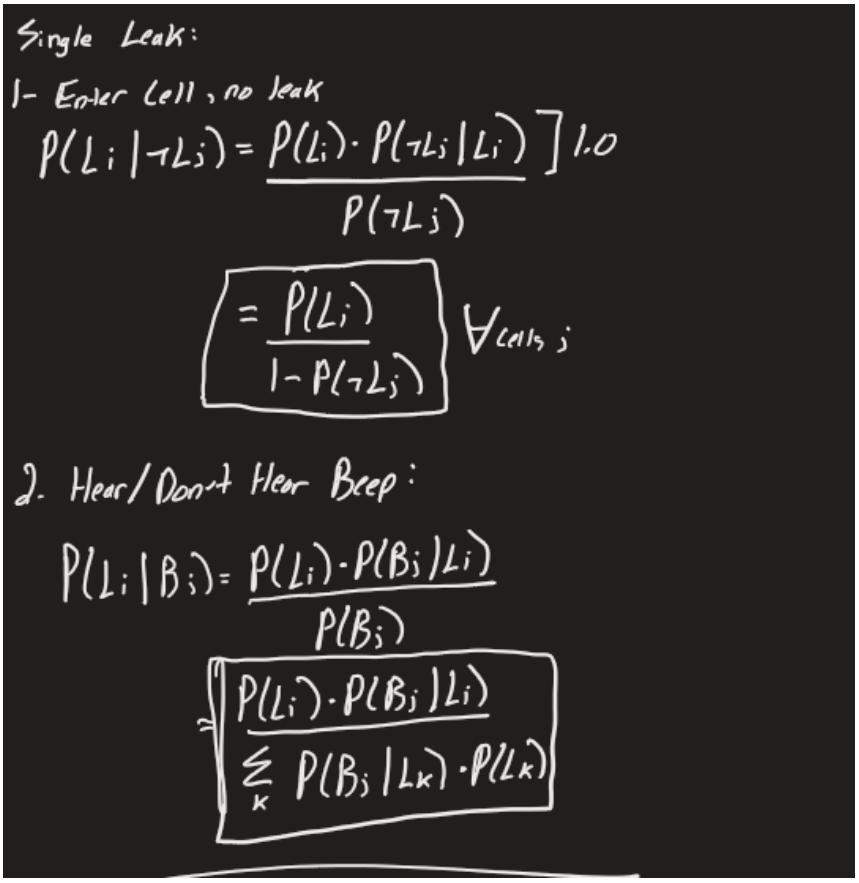
g. Bot 7 - Two Leak Situation with the same probability updates as Bot 3
   i. This bot uses the exact same algorithms as Bot 3 despite there being two leaks present instead of one. Probabilistically speaking, the knowledge base of this bot is not accurate based on the information it is receiving. Just like Bot 3, it can choose to ***move*** or ***sense*** at every step. For every move, it navigates to the cell of highest probability based on prior information. The updates done after every ***move*** or ***sense*** are exactly as Bot 3.
   ii. Once Bot 7 finds the first leak using the "incorrect" (as per the project writeup) information, it immediately becomes **exactly** Bot 3, as it is now only searching for 1 leak.

h. Bot 8 - Two Leak Situation with Corrected Probability Updates
   i. Bot 8's knowledge base, unlike Bot 3's, must account for multiple leaks at once. Because there are 2 leaks on the board, it is not sufficient to only account for *P(Leak in (i, j))* for all open cells *(i, j)*, but rather must keep track of *P(Leak in (i, j) AND Leak in (k, l))* for ***all pairs of cells (i, j), (k, l)***. This requires the knowledge base to be represented as a 4D array, where *array[i, j, k, l] = array[k, l, i, j] = P(Leak in (i, j) AND Leak in (k, l))*.
   ii. Once Bot 8 find the first leak, it instantly spawns an instance of Bot 3 where the position is Bot 8's current position, and the probability matrix is *prob_matrix[i, j, :, :]* of Bot 8. This is because the only probabilities left to account for are ones that involve *P(Leak in j)*, given the first leak was

found in cell j. After the Bot 3 instance finds the second leak, the timesteps are added to bot 8's, and the bot stops it's search.

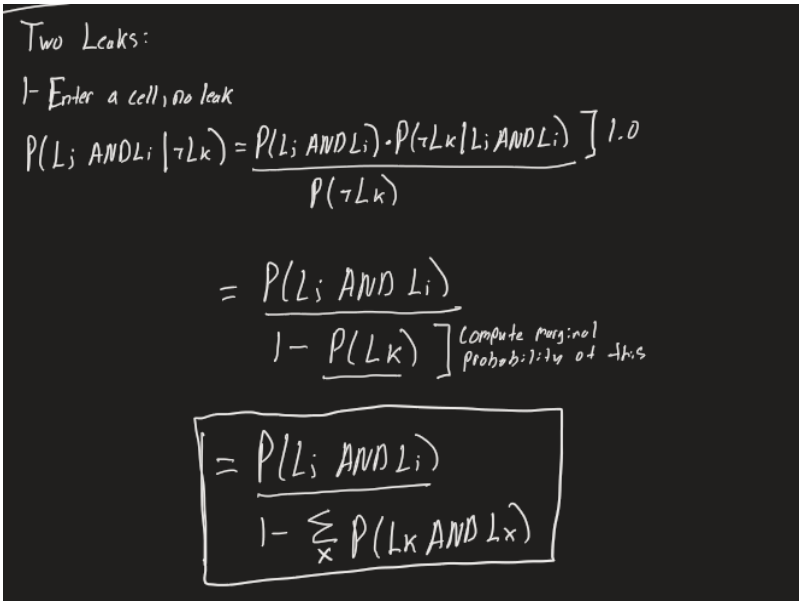
i. Bot 9 - Two Leaks making better use of path planning and information
    i. This bot uses the exact same algorithms as Bot 4. It plans it's path using the path of A. highest probability and B. least previously explored cells. Just as Bot 4 does, it also performs a **sense** action while traversing from time to time to get better information. Just as Bot 8 is transferred to an instance of Bot 3, Bot 9 is converted to Bot 4 once the first leak is found. The probability updates are exactly as they are in Bot 8.

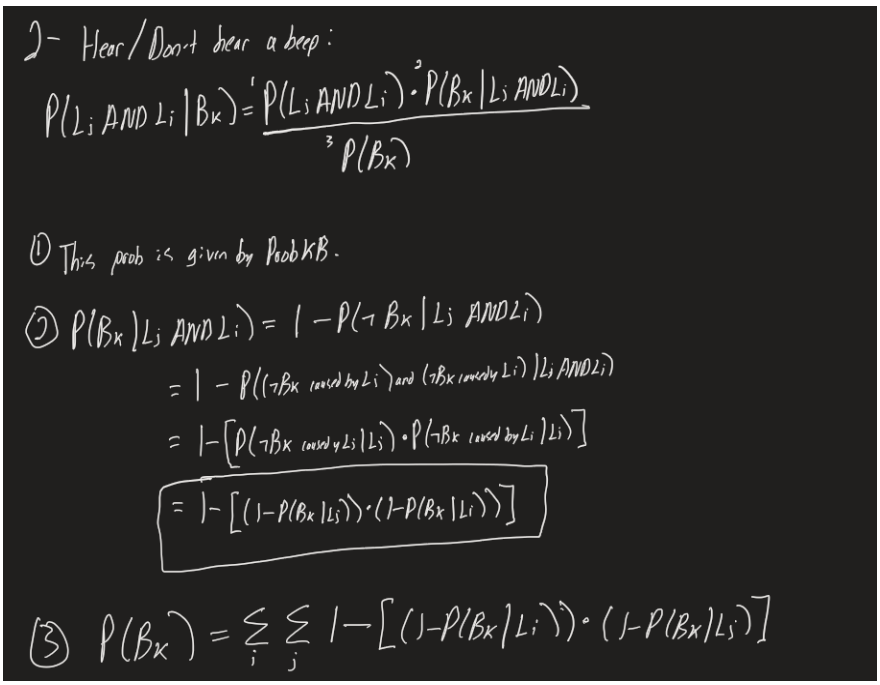
2. How should the probabilities be updated for Bot 3/4? How should the probabilities be updated for Bot 8/9?
    a. The probabilities for Bot 3/4 are calculated as such:




As seen in the second half, to compute the denominator *P(Beep heard in J)*, you must marginalize over the probability that *P(Beep heard in j | Leak in k) * P(Leak in k)* for all open cells *k*.

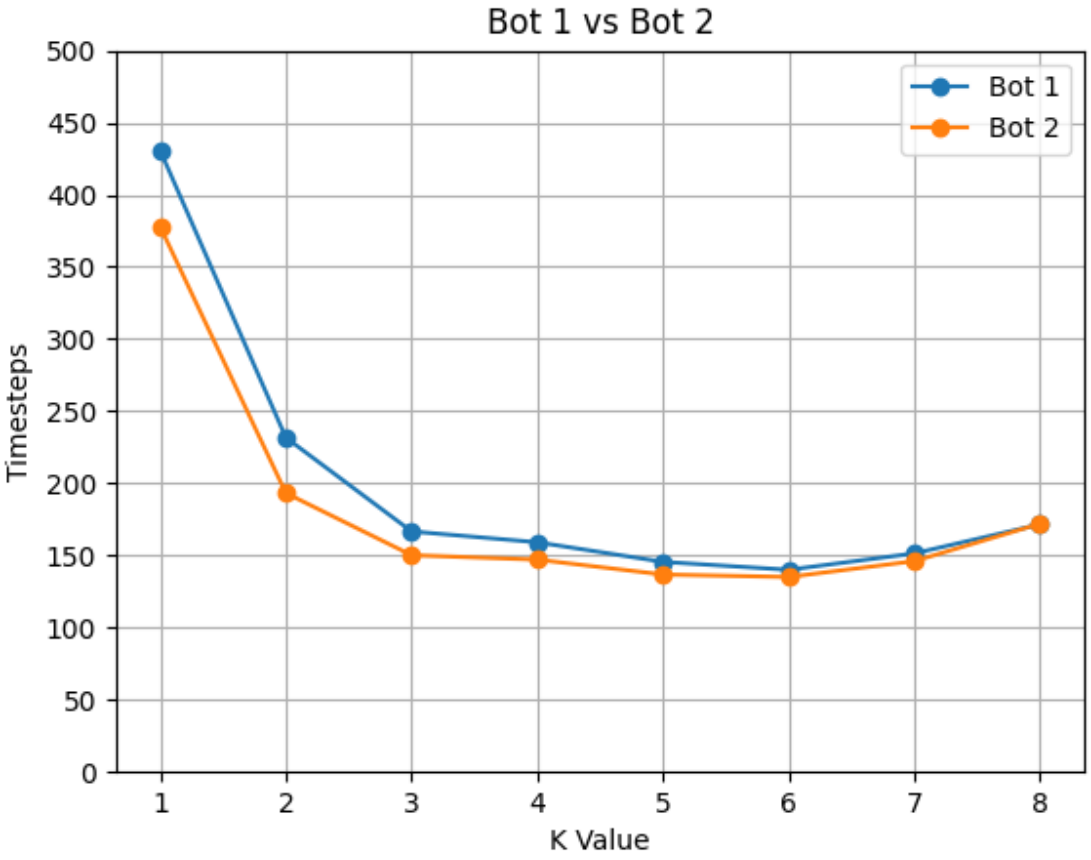
b. The Probabilities are Bot 8/9 are calculated as such:



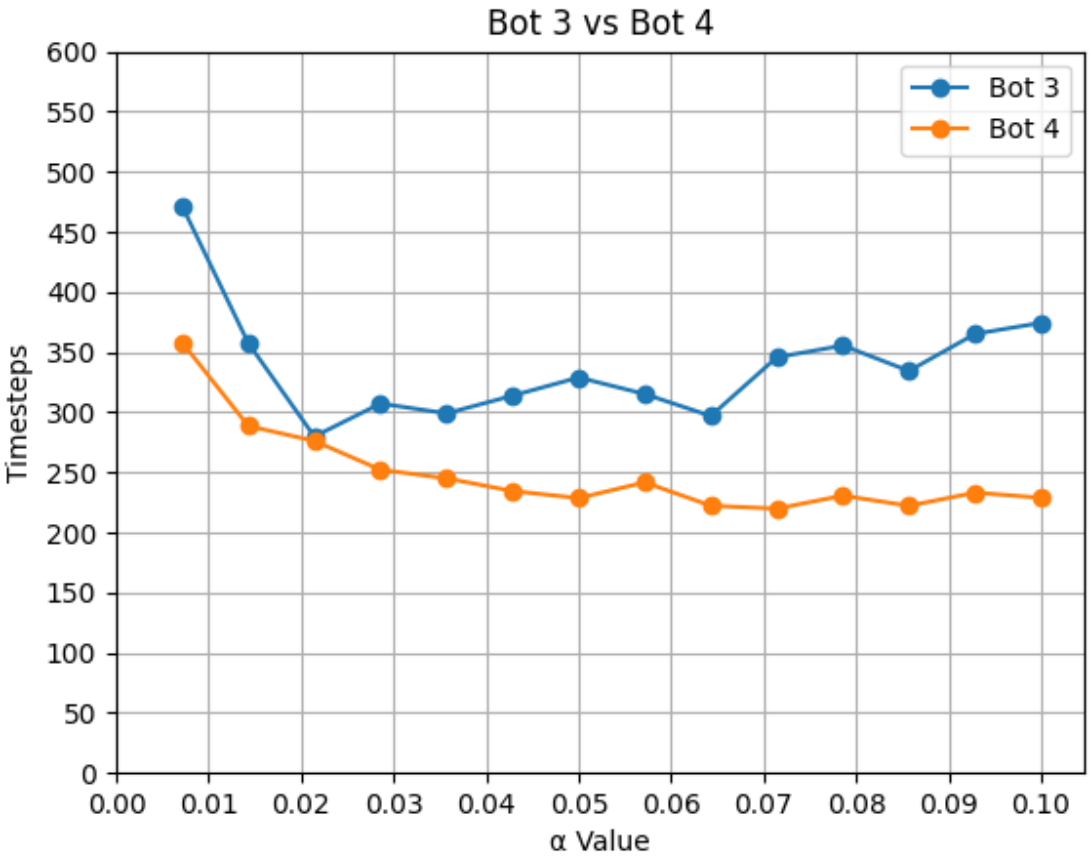

Just as with the 1 leak situation, the numerator *P(Beep heard in k)* must be computed as the sum of *hearing a beep at all* for *every pair of cells **i and j***. In order to compute *P(Hearing a beep at all | Leak in i and Leak in j)*.

3. Generate test environments to evaluate and compare the performances of your bots. Your measure of performance here is the average number of actions (moves + sensing) needed to plug the leak.
    a. Bot 1 vs Bot 2, as a function of *k*.



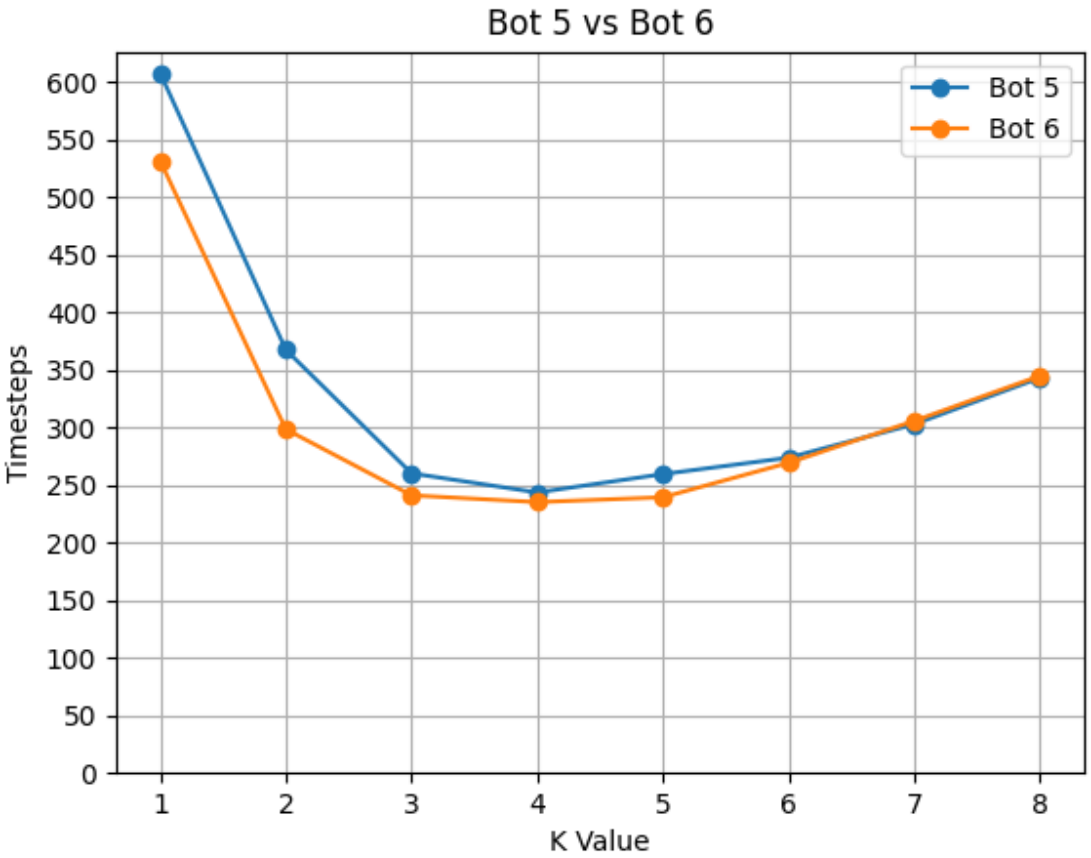

    i. Both tested on the same 100 seeds

b. Bot 3 vs Bot 4, as a function of $\alpha$.



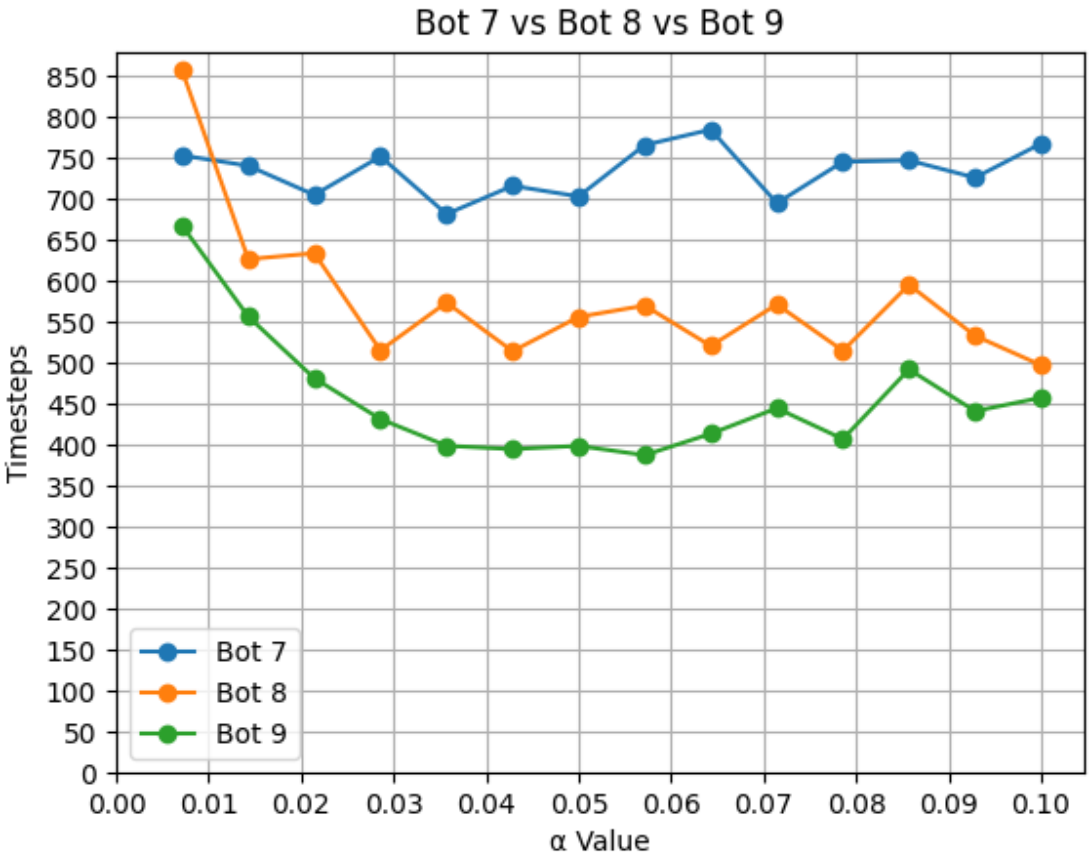

i. Both were tested on the same 98 seeds.

c. Bot 5 vs Bot 6, as a function of *k*.




i. Both were tested on the same 100 seeds.

d. Bot 7 vs Bot 8 vs Bot 9, as a function of $\alpha$.




i. All three were tested on the same 43 seeds.

4. Speculate on how you might construct the ideal bot. What information would it use, what information would it compute, and how?
   a. To construct the ideal deterministic bot, we would use information about the layout of the grid to compute a minimum-length path to scan a minimum number of times to cover the whole ship. This is because we would be able to sense the leak (on average) faster, since the leak is equally likely to be placed in any cell in the ship, so detecting it the fastest is equivalent to scanning the most cells the fastest.
   b. To construct the ideal probabilistic bot, we could potentially sense even more often (such as every step), since this might prevent bouncing around even more. Additionally, we could run simulations on how we anticipate the probability distribution to change so that we find the best course of action for the whole trial.