Intro to AI Project 2

Author: Sihua Zhou, NetID: sz583

Note: Had discussion with TA Patrick, README is not included, please reach out to me when need to run the program.

Project 2: Airing Out

Purpose of this project: "It is another day on the deep space vessel Archaeopteryx, and you are a lonely bot. You're responsible for the safety and security of the ship while the crew is in deep hibernation. As an example, in the case that the ship springs a leak, you need to find that leak and plug it." (Cited from project 2 writeup).

Setups:

- Ship – is the same ship setup but with bot and leaks 50 by 50, 30 by 30.
- Bots – instead moving toward the button to put out fire, the bots now would choose to move or sense at each time stamp.
- Leaks – a leak is selected in random of an open cell, the leak only would be plugged when bot steps on it. Plugging all leaks is the task.

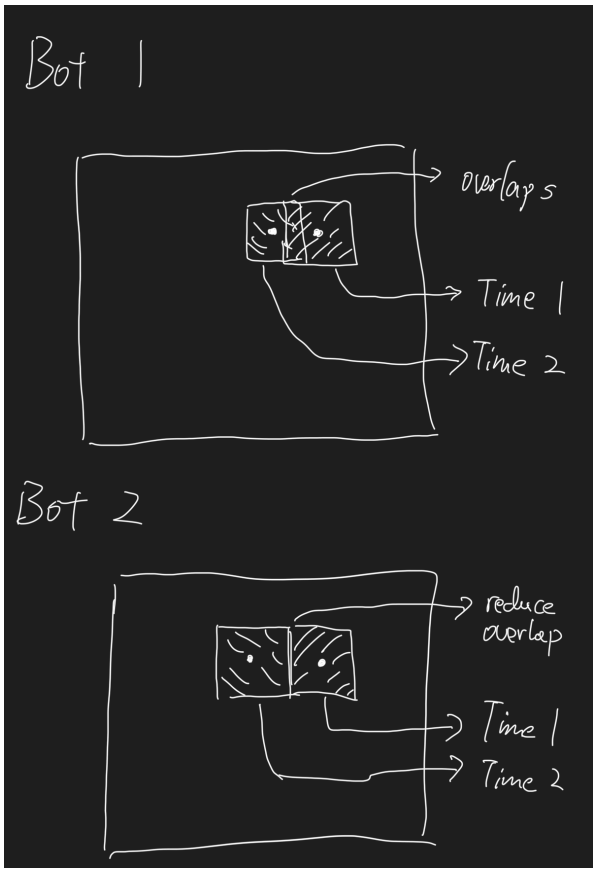
Data and Analysis:

1. Design and algorithm:

- Part 1: Deterministic Leak Detectors
    - General idea: Both bot1 and bot2 is using Detection Square(DS) to scan the leak, by checking the square range with a K value (k >= 1) of 2*k+1 by 2*k+1 square, returning true if a leak is found within the square. In this scenario, there is only one leak involved.
    - Bot1:
        - General Actions: Bot1 would perform a series of actions, including scan using Detection Square, update probability according to scan result, find the nearest next_location that has the chance of having a leak there, move to that cell, and

check if the current bot location is at leak cell. Bot1 would do these actions above in the order I listed them and until it finds the leak and arrived at the leak.

- Detail Discussion:
    - Detection Square(DS): with K value as a variable of the detection square with size (2*k+1 by 2*k+1), the DS is scanning the area where the bot is located. The bot is the center of the square; therefore, the range would be:
        - Start_row = bot_location_row – k_value
        - End_row = bot_location_row + k_value
        - Start_col = bot_location_col – k_value
        - End_col = bot_location_col + k_value
        - Then iterate through probability table (2d array) if it is within the range check if it is a leak.
    - Update probability table:
        - If scan result come back with true/positive, rule out all other cells that is not in the DS. Mark these cells as well as the cell bot is located as false or 0% probability of having a leak.
        - If scan result come back with false/negative, rule out all cells that is within the DS. Mark these cells as false or 0% probability of having a leak.
    - Find the nearest next_location with probability of containing a leak:
        - Uses of BFS to find the nearest path to a leak that has chance of containing a leak.
        - Ending condition of the BFS probability > 0% (Also can be possible to have a leak == true).
    - Move to the next location:
        - Set the bot to next_location cell.
    - Check if the bot location (cell) is at the leak.
        - Compare bot coordinate with leak coordinate.
            - If true, task complete, return total actions.
            - If false, continue the cycle of all the action above in order.

- Bot2:
  - General Action: Bot2 is doing the same routine as what Bot1 does, but one part is different from Bot1. At action of “Find the nearest next_location with probability of containing a leak”, Bot2 would go further to reduce the overlap between detection square and scan/cover area of the ship more at next stamp, because the quicker the detection square return with positive result the faster we can rule out more cells that don’t have chance of containing leak.
  - Visualization:
    - 

  - Detail Discussion:
    - Detection Square(DS): same as bot1.
    - Update probability table: same as bot1.
    - Finding the next location:
      - Still using BFS to find the shortest path, but with additional ending condition, probability > 0% AND the depth(length of the cell) > a proportion of K_value (K_alter).
      - Ship_size = 50

        - 2*k+1 <= Ship_size/8
            - Uses K_alter = 2*k
        - 2*k+1 <= Ship_size/4
            - Uses K_alter = k*3/2
        - 2*k+1 <= Ship_size/2
            - Uses K_alter = k*3/4
        - Other k_value except the k_value above
            - Uses K_alter = k
        - If there is no such cell (location), then we go back BFS to find the shortest path with the same ending condition probability > 0%.
    - Move to the next location: same as bot1.
    - Check if the bot location(cell) is at the leak: same as bot1.

- Part 2: Probabilistic Leak Detectors:
    - General idea: Both Bot3 and Bot4 with probabilistic detection, every open cell start with a equivalent probability of 1/total_open_cells. Using the sensor to receive a "BEEP" from the leak, but whether receiving a "BEEP" or not is depended on the probability. The crucial factors to determine the probability is the Alpha value (alpha > 0) and the distance from bot to the leak with a formula of (e ^ ((-1)*alpha*(distance-1))). Negative power gives fractions which (1/ (e ^ (alpha * (distance-1))), where the closer to leak from bot location the higher chance to receive a beep. In this scenario, there is only one leak involved.
    - Bot3:
        - General Actions: Bot3 would perform series of action in order, including check if bot location is at leak, update the probability table after entering a cell that is not a leak, listen for a beep, gather distance of each cell from the bot location, update probability table according to result of beep, find the next location that consist with the highest probability, move along the path and update the probability table if entered a cell is a leak(Task complete if it is).
        - Detail Discussion:
            - Check if bot location is at leak:
                - Use bot location(cell) compared to the leak position.
                    - If true, task complete return total actions.

        - If false, continues.
- Update probability table entering a cell that is not a leak.
    - Since bot entered a non-leak cell mark the current cell as 0% probability of having a leak.
    - Then calculate P(leak in i | leak not in bot_location) for each i in probability table.
        - Detail calculation in question(part) 2.
- Listen for a beep:
    - Calculate the probability = e ^((-1) * alpha_value * (distance_from_leak_to_bot - 1))
    - Get a random value.
        - If random value <= probability
            - Heard a beep.
        - Else
            - Didn't hear a beep.
- Gather distance of each cell from the bot location
    - Using BFS without any early ending condition, every time pop a cell from queue and add the depth(length) to a distance table for further reference in "update probability table according beep result".
- Update probability table according beep result:
    - If heard a beep
        - Calculate P(leak in cell j | heard beep in bot_location) for each cell j in probability table.
    - Else
        - Calculate P(leak in cell j | heard no beep in bot_location) for each cell j in probability table.
    - Detail calculation in question(part) 2.
- Find the next location that has the highest probability:
    - Using BFS without any early ending, compare the probability in that cell, if the current cell is great than the max_probability cell,

update the new max_probability cell to current cell and BFS through all the cell on ship.

- Return the max_probability cell at the end.

- Move the bot along the path to max_probability cell:
  - Set the bot location to next coordinate in the path, until it reach the max_probability cell.
  - Every time move to a new cell, check if it is at the leak
    - If true, task complete, return total action
    - If false, "update probability table entering a cell that is not a leak." From above. And then continue the for each loop.

- Bot4:
  - General Actions: Bot4 is doing the same routine as bot3 except in "move the bot along the path max_probability cell", that there is a chance the bot to stop in the middle of the path and do another listen on beep and find new way to go.
  - Detail Discussion:
    - Check if bot location is at leak: same as bot3.
    - Update probability table entering a cell that is not a leak: same as bot3.
    - Listen for a beep: same as bot3.
    - Gather distance of each cell from the bot location: same as bot3.
    - Update probability table according beep result:
      - same as bot3.
      - Detail calculation in question(part) 2.
    - Find the next location that has the highest probability: same as bot3.
    - Move the bot along the path to max_probability cell:
      - Set the bot location to next coordinate in the path, until reach the max_probability cell.
      - Setting a step_count = 0
      - Every time move to a new cell, check if it is at the leak
        - If true, task complete, return total action
        - If false, "update probability table entering a cell that is not a leak.", then check step_count%3 == 0 in this case,

then use the formula (e ^ ((-1) * alpha * (next_loc.length – step_count))) to determine the probability compare with a random probability generated if a bot should break from current for each loop and perform a listen to the beep and update probability table and adjust the way to go. In this case, the further the bot go will likely to stop and because closer the highest probability cell is likely to hear a beep. However, if the distance from bot current location to next location is very close, then it would not be likely to break from the for each loop.

- Part 3: Multiple Leaks:
    - In this case, all bot from 5 to 9 is solving the problem with two leaks.
    - General idea: In this part every bot 1 and 2 would upgrade to bot 5 and 6, bot 3 and 4 would upgrade to 7 to 9 to better solve two leaks situation. A Boolean array with size 2 is used to store information of whether the leak is visited or not.
    - Bot5:
        - General actions: Bot5 is really nothing different than Bot1 except plugging two leaks and doing some different action when detection square scan result is true. Bot5 would mark any visited leak in leak array as true
        - Detail discussion:
            - Detection Square(DS): with K value as a variable of the detection square with size (2*k+1 by 2*k+1), the DS is scanning the area where the bot is located. The bot is the center of the square; therefore, the range would be:
                - Start_row = bot_location_row – k_value
                - End_row = bot_location_row + k_value
                - Start_col = bot_location_col – k_value
                - End_col = bot_location_col + k_value
                - Then iterate through probability table (2d array) if it is within the range check if it is a leak.

        - Only different here in bot 5 is when detecting true, at least one leak is in that DS.
    - Update probability table:
        - If scan result come back with true/positive and mark the cell bot is located as false or 0% probability of having a leak .
        - Different from bot1, Bot5 would perform like Bot1, but only within the detection square area. Once all cells within the detection square is visited, mark visited leak in leak array as true, and then check if both leaks are visited (because we don't know if both cells are in the DS, so we have to visit all cells within the DS). if not, set every cell in the detection square as 0%, because we've visit every cell here and continue the routine. If yes, task complete and return total actions.
        - If scan result come back with false/negative, rule out all cells that is within the DS. Mark these cells as false or 0% probability of having a leak.
    - Find the nearest next_location with probability of containing a leak:
        - Uses of BFS to find the nearest path to a leak that has chance of containing a leak.
        - Ending condition of the BFS probability > 0% (Also can be possible to have a leak == true).
    - Move to the next location:
        - Set the bot to next_location cell.
    - Check if the bot location (cell) is at the leak.
        - Compare bot coordinate with leak coordinate.
            - If true, mark visited leak true in leak array
        - If both leaks visit, task complete, return total actions.
            - If not, continue the cycle of all the action above in order.

- Bot6:
    - General Actions: Bot6 perform almost the same as Bot5, except "finding the next location" that bot6 will go a little bit further to reduce overlap on detection square.

- Detail Discussion:
    - Detection Square(DS): same as bot5.
    - Update probability table: same as bot5.
    - Finding the next location:
        - Still using BFS to find the shortest path, but with additional ending condition, probability > 0% AND the depth(length of the cell) > a proportion of K_value (K_alter).
        - Ship_size = 50
        - 2*k+1 <= Ship_size/8
            - Uses K_alter = 2*k
        - 2*k+1 <= Ship_size/4
            - Uses K_alter = k*3/2
        - 2*k+1 <= Ship_size/2
            - Uses K_alter = k*3/4
        - Other k_value except the k_value above
            - Uses K_alter = k
        - If there is no such cell (location), then we go back BFS to find the shortest path with the same ending condition probability > 0%.
    - Move to the next location: same as bot5.
    - Check if the bot location(cell) is at the leak: same as bot5.

- Bot 7:
    - General Actions: Bot7 exactly the same as Bot3, but Bot3 only have one leak to plug, Bot7 has two. Bot7 will perform actions just like Bot3 until both leaks are plugged, in other words, until the visited leaks array with size 2 contains only true in it.
    - Detail Discussion:
        - Check if bot location is at leak:
            - Mark visited leak as true in the leak array, if bot is at the leak.
            - If both leaks solved, task complete return total actions.
        - Update probability table entering a cell that is not a leak:
            - Since bot entered a non-leak cell mark the current cell as 0% probability of having a leak.

        - Then calculate P(leak in i | leak not in bot_location) for each i.
            - Detail calculation in question(part) 2.
    - Listen for a beep:
        - Calculate the probability = e ^((-1) * alpha_value * (distance_from_leak_to_bot - 1))
        - Get a random value
            - If random value <= probability
                - Heard a beep
            - Else
                - Didn't hear a beep
        - In this case, we use OR (||) to get the result,
            - Result for leak 1 (heard or no) || Result for leak 2 (heard or no)
            - If the Boolean from above is true, then heard a beep
            - If false, then no beep heard.
    - Gather distance of each cell from the bot location: same as Bot3.
    - Update probability table according beep result:
        - Same as Bot3
        - Detail calculation in question(part) 2.
    - Find the next location that has the highest probability: same as Bot3.
    - Move the bot along the path to max_probability cell:
        - Set the bot location to next coordinate in the path, until it reach the max_probability cell.
        - Every time move to a new cell, check if it is at the leak
            - If true, mark visited leak true in leak array.
        - If both leaks visit, task complete, return total actions.
            - If false, "update probability table entering a cell that is not a leak." From above. And then continue the for each loop.

- Bot8:
    - General Actions: Bot8 performs actions same as Bot7, but how the update probability table after entered a non-leak cell, update probability table based on

beep are changed with correct probability involving a hashmap to store a set of two coordinate for each cell I and cell j as key and the probability as value of it.

- Detail Discussion:
    - Check if bot location is at leak: same as Bot 7.
    - Update probability table entering a cell that is not a leak:
        - Since bot entered a non-leak cell mark the current cell as 0% probability of having a leak.
        - Then calculate P(leak in i and leak in j | leak not in bot_location) for each pair of cell i and cell j that i != bot_location and j != bot_location in probability pair set.
        - Update probability table use marginalization P (leak in j) for each cell in probability table.
        - Detail calculation in question(part) 2.
    - Listen for a beep: same as Bot7.
    - Gather distance of each cell from the bot location: same as Bot7.
    - Update probability table according beep result:
        - If heard a beep
            - Calculate P(leak in i and leak in j | heard beep in bot_location) for each pair of cell i and cell j that i != bot_location and j != bot_location in probability pair set.
            - Update probability table use marginalization P (leak in j) for each cell in probability table.
        - Else
            - Calculate P(leak in i and leak in j | heard no beep in bot_location) for each pair of cell i and cell j that i != bot_location and j != bot_location in probability pair set.
            - Update probability table use marginalization P (leak in j) for each cell in probability table.
        - Detail calculation in question(part) 2.
    - Find the next location that has the highest probability: same as Bot7.
    - Move the bot along the path to max_probability cell: same as Bot7.

- Bot9:

- General Actions: Bot9 is same as Bot8 but some changes in “probability table according to beep result”, that after failed to heard a “BEEP” and based on some probability, the bot will choose to stay in position to keep listen for “BEEP”.
- Detail Discussion:
    - Check if bot location is at leak: same as Bot8.
    - Update probability table entering a cell that is not a leak:
        - Same as Bot8
        - Detail calculation in question(part) 2.
    - Listen for a beep: same as Bot8.
    - Gather distance of each cell from the bot location: same as Bot8.
    - Update probability table according beep result:
        - If heard a beep
            - Calculate P(leak in i and leak in j | heard beep in bot_location) for each pair of cell i and cell j that i != bot_location and j != bot_location in probability pair set.
            - Update probability table use marginalization P (leak in j) for each cell in probability table.
        - Else
            - Calculate P(leak in i and leak in j | heard no beep in bot_location) for each pair of cell i and cell j that i != bot_location and j != bot_location in probability pair set.
            - Update probability table use marginalization P (leak in j) for each cell in probability table.
            - Using formula probability = (e ^ ((-1) * alpha * (current_total_action)))
                - then if random_probability <= probability
                    - use keyword “continue” to skip the actions below “Update probability table according beep result” and continue another iteration for the outer while loop.
                - Else continue the rest of the actions.

        - Detail calculation in question(part) 2.
    - Find the next location that has the highest probability: same as Bot8.
    - Move the bot along the path to max_probability cell: same as Bot8.

2. Detail Discussion on how should Bot3/4's and Bot8/9's probability be update.

- Bot 3/4:
    - Recall probability we have to calculate from previous question.
    - After entered a cell that is not a leak. For all cell j in probability table,
        - P (Leak in cell j | Leak not in Bot location) for each cell j in probability table
            - P (Leak in cell j AND Leak not in Bot location) / P (Leak not in Bot location)
            - P (Leak not in Bot location | Leak in cell j) * P (Leak in cell j) / P (Leak not in Bot location)
                - P (Leak in cell j) – this can be found in probability table (2d array that store each cell j's probability)
                - P (Leak not in Bot location | Leak in cell j) – because leak is not in Bot location, so this part equals to 1.
                - P (Leak not in Bot location) = summation { each cell j in probability table } { P (Leak in cell j AND Leak not in Bot location) } (Marginalization)
                    - P (Leak not in Bot location) = summation { each cell j in probability table } { P (Leak in cell j) * P (Leak not in Bot location | Leak in cell j) }
                    - Where we reduce to the problem we solved previous.
    - After heard a beep. For all cell j in probability table,
        - P (Leak in cell j | Heard beep in Bot location)
            - P (Leak in cell j AND Heard beep in Bot location) / P (Heard beep in Bot location)
            - P (Heard beep in Bot location | Leak in cell j) * P (Leak in cell j) / P (Heard beep in Bot location)
                - P (Leak in cell j) – this can be found in probability table (2d array that store each cell j's probability)

        - P (Heard beep in Bot location | Leak in cell j) = (e ^ ((-1) * alpha * (distance from cell j to Bot location – 1)))
        - P (Heard beep in Bot location) = summation { each cell j in probability table } { P (Leak in cell j AND Heard beep in Bot location) } (Marginalization)
            - P (Heard beep in Bot location) = summation { each cell j in probability table } { P (Leak in cell j) * P (Heard beep in Bot location | Leak in cell j) }
            - Again, reduced to a problem solved before.
    - After heard no beep. For all cell j in probability table,
        - P (Leak in cell j | Heard no beep in Bot location)
            - P (Leak in cell j AND Heard no beep in Bot location) / P (Heard no beep in Bot location)
            - P (Heard no beep in Bot location | Leak in cell j) * P (Leak in cell j) / P (Heard no beep in Bot location)
                - P (Leak in cell j) – this can be found in probability table (2d array that store each cell j's probability)
                - P (Heard no beep in Bot location | Leak in cell j) = 1 - P (Heard beep in Bot location | Leak in cell j) = 1 - (e ^ ((-1) * alpha * (distance from cell j to Bot location – 1))) (from previous situation)
                - P (Heard no beep in Bot location) = 1 - P (Heard beep in Bot location) (from previous situation)
                - Reduced to a problem solved before.

- Bot 8/9:
    - Recall probability we have to calculate from previous question.
    - After entered a cell that is not a leak. For all pair of cell i and j in probability pair set,
        - P (Leak in cell i and Leak in cell j | Leak not in Bot location)
            - P (Leak in cell i and Leak in cell j AND Leak not in Bot location) / P (Leak not in Bot location)
            - P (Leak not in Bot location | Leak in cell i and Leak in cell j) * P (Leak in cell i and Leak in cell j) / P (Leak not in Bot location)

        - P (Leak in cell i and Leak in cell j) - this can be found in probability pair set (Hashmap that store pair of cells as key and probability as their value)
        - P (Leak not in Bot location | Leak in cell i and Leak in cell j) - because leak is not in Bot location, so this part equals to 1.
        - P (Leak not in Bot location) = summation { pair set cell I and cell j in probability pair set } { P (Leak in cell i and Leak in cell j AND Leak not in Bot location) } (Marginalization)
            - P (Leak not in Bot location) = summation { pair set cell I and cell j in probability pair set } { P (Leak in cell i and Leak in cell j) * P (Leak not in Bot location | Leak in cell i and Leak in cell j) }
            - Reduced to a problem solved before.
- After heard a beep.
    - P (Leak in cell i and Leak in cell j | Heard beep in Bot location)
        - P (Leak in cell i and Leak in cell j AND Heard beep in Bot location) / P (Heard beep in Bot location)
        - P (Heard beep in Bot location | Leak in cell i and Leak in cell j) * P (Leak in cell i and Leak in cell j) / P (Heard beep in Bot location)
            - P (Leak in cell i and Leak in cell j) - this can be found in probability pair set (Hashmap that store pair of cells as key and probability as their value)
            - P (Heard beep in Bot location | Leak in cell i and Leak in cell j)
                - P((Heard beep in Bot location DUE TO leak in cell i) OR (Heard beep in Bot location DUE TO leak in cell j) | leak in cell i AND leak in cell j)
                - P(Heard beep in Bot location DUE TO leak in cell i | leak in cell i AND leak in cell j) + P(Heard beep in Bot location DUE TO leak in cell j | leak in cell i AND leak in cell j) - P((Heard beep in Bot location DUE TO leak in cell i) AND (Heard beep in Bot location DUE TO leak in cell j) | leak in cell i AND leak in cell j)

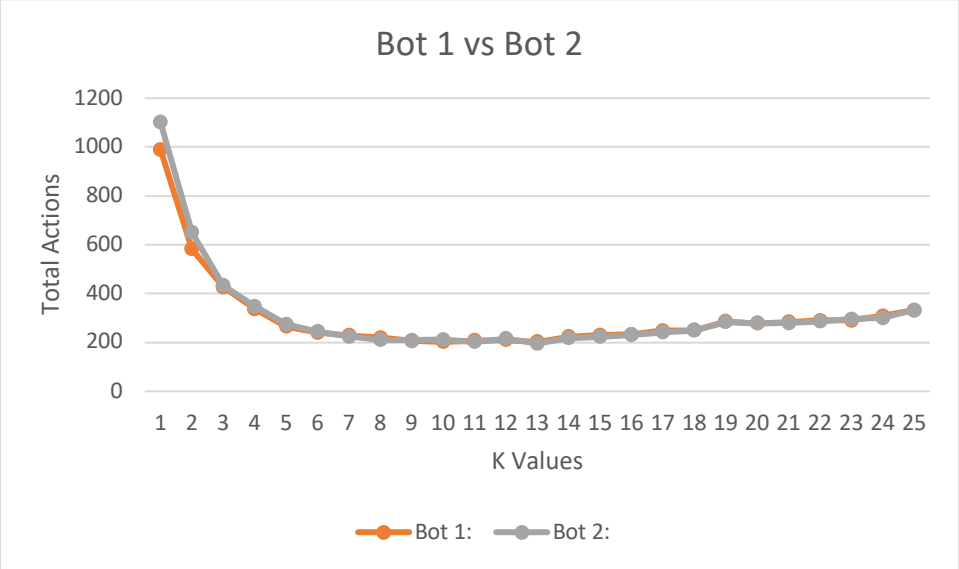
        - P(Heard beep in Bot location DUE TO leak in cell i | leak in cell i) + P (Heard beep in Bot location DUE TO leak in cell j | leak in cell j) - P(Heard beep in Bot location DUE TO leak in cell i | leak in cell i) * P(Heard beep in Bot location DUE TO leak in cell j | leak in cell j)
        - P (Heard beep in Bot location | leak in cell i AND leak in cell j) = e^(- a * (d(k,i) - 1)) + e^(- a * (d(k,j) - 1)) - e^ - a * (d(k,i) - 1)) * e^(- a * (d(k,j) - 1))
        - From Dr. Cowan office hour note. d(k, i) representing distance from bot location (k) to cell I, etc.
    - P (Heard beep in Bot location) = summation { pair set cell I and cell j in probability pair set } { P (Leak in cell i and Leak in cell j AND Heard beep in Bot location) } (Marginalization)
        - P (Heard beep in Bot location) = summation { pair set cell I and cell j in probability pair set } { P (Leak in cell i and Leak in cell j) * P (Heard beep in Bot location | Leak in cell i and Leak in cell j) }
        - Reduced to a problem solved before.
- After heard no beep.
    - P (Leak in cell i and Leak in cell j | Heard no beep in Bot location)
        - P (Leak in cell i and Leak in cell j AND Heard no beep in Bot location) / P (Heard no beep in Bot location)
        - P (Heard no beep in Bot location | Leak in cell i and Leak in cell j) * P (Leak in cell i and Leak in cell j) / P (Heard no beep in Bot location)
            - P (Leak in cell i and Leak in cell j) - this can be found in probability pair set (Hashmap that store pair of cells as key and probability as their value)
            - P (Heard no beep in Bot location | Leak in cell i and Leak in cell j) = 1 - P (Heard beep in Bot location | Leak in cell i and Leak in cell j)
            - P (Heard no beep in Bot location) = 1 - P (Heard beep in Bot location)

        - Reduced to a problem solved before.
    - Update probability table
        - P (Leak in cell j) for each cell in probability table
            - summation { pair set that contain cell j in probability pair set } { P (Leak in cell i and Leak in cell j) }

3. Performance Comparison bot1 vs bot2, bot3 vs bot4, bot5 vs bot6, bot7 vs bot8 vs bot9

- Bot1 to bot6 used 50 by 50 and ran 300 time per k or a value
- Bot7 to bot9 used 30 by 30 and ran 100 time per a value for time efficiency

- bot1 vs bot2

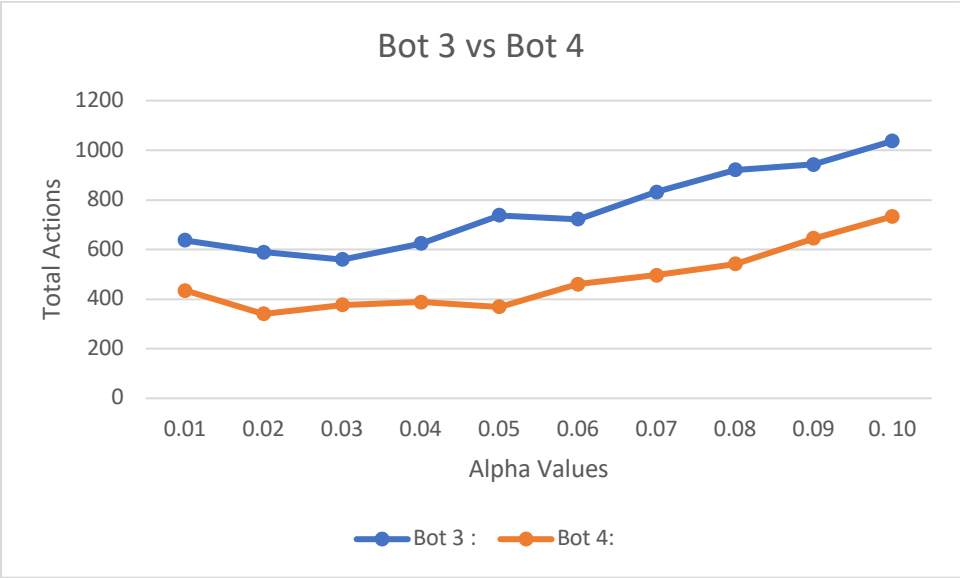


- Raw Data:

K Val:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Bot 1:	990	582	427	337	265	240	229	219	207	202	208	210	202	223	230	233	248	250	286	279	284	290	290	308	332
Bot 2:	1102	651	433	348	274	244	225	210	209	211	203	215	196	218	224	231	242	250	284	278	280	286	296	301	332

- **Summary:** Overall, bot2 doesn't really make significant change on performance compared to bot1, but still at some points bot2 has better performance especially at k = 10 to 20. You can tell the gray line is under the orange one.
- **Justification:** maybe the change from K value to K alter should have a better proportion to assign to get a better result.

- bot3 vs bot4

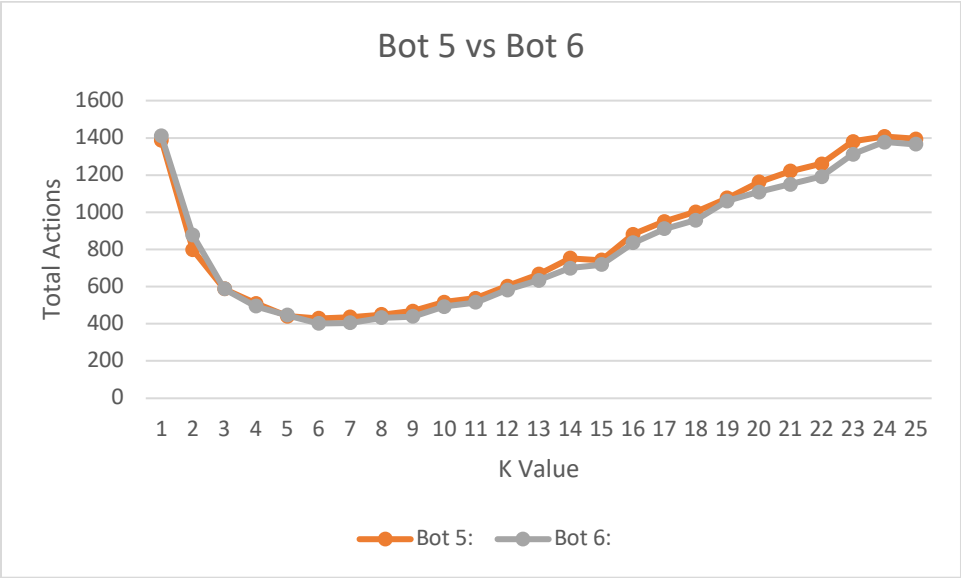


- Raw Data:

A Val:	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09	0.10
Bot 3:	637	589	559	624	738	722	831	921	942	1037
Bot 4:	434	340	376	388	369	460	496	542	645	733

- **Summary:** Bot4 in this case has outcompete Bot3 the performance gap is significant and easily to be observe, meaning the strategy used in bot4 is good on improving bot3.
- **Justification:** In this case, performing more listen to beep helped bot solve the problem faster, because heard beep help change the probability table to help bot make correction on the way it is heading.

- bot5 vs bot6

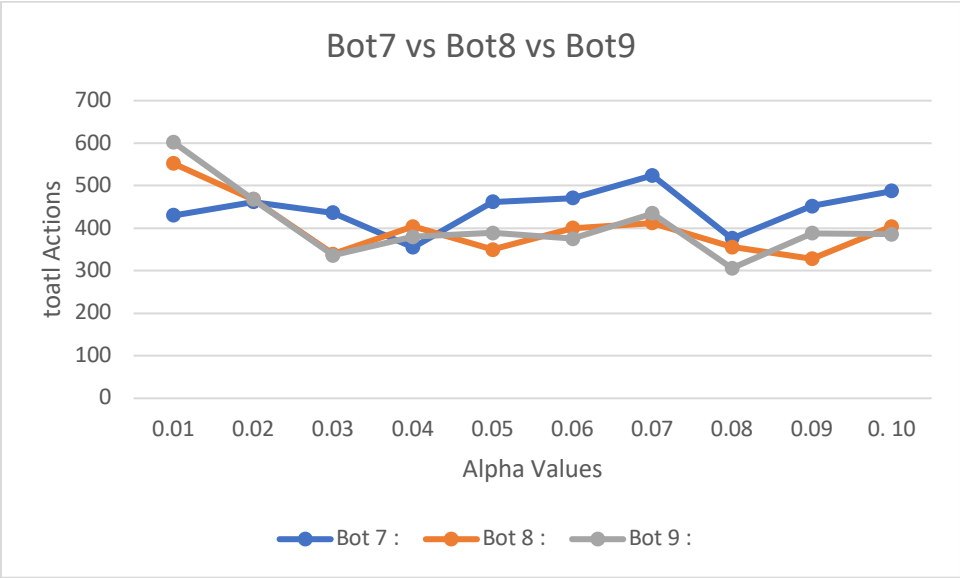


- Raw Data:

K Val:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Bot 5:	1387	798	588	510	442	429	437	450	469	518	537	604	667	752	742	882	950	1003	1076	1163	1221	1260	1381	1407	1394
Bot 6:	1410	876	587	494	447	402	405	433	440	493	515	583	634	700	718	836	911	958	1061	1108	1150	1191	1312	1377	1365

- **Summary:** It is not hard to observe that bot6 is having better performance than bot5, especially from k = 6 to 25 (end of the k value), bot6 maintained slightly better performance compared with bot5.
- **Justification:** maybe if there were more leaks involved the bot can performance even better, because the further you go from the previous location, the more leak can be found and solve, causing the increase in performance.

- bot7 vs bot8 vs bot9



- Raw Data:

A Val:	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09	0.10
Bot 7:	430	462	436	355	462	471	524	376	452	487
Bot 8:	552	468	339	404	350	400	412	356	328	404
Bot 9:	602	468	336	380	389	375	435	306	388	386

- **Summary:** Bot8 and Bot9 at most of the time having better performance than bot7, and I believe the performance gap between bot8 and bot9 is quite small, that might indicate the strategy used in bot9 is not that good, but at some situation helped bot9 to solve the problem faster.
- **Justification:** In this case, showing the correct probability does help improving performance, but the performance between bot8 and bot9 indicates keep staying might not be a good idea and not necessarily able to contribute in improvement on performance.

4. How to construct an ideal bot?

- Though we haven't done a Bot with a combination of detection square with probability table (hearing beep thing) in this project. However, I believe the combination of these two techniques and dramatically increase the performance of the bot on solving this leak problem. DS bot would be able to have the correct probability update, and Beep bot have a wider range on ruling out cells that don't need to go. That conclude my thought about ideal bot.