Project 2 – 440 Fall 2023

Professor – Dr. Cowan

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

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?**
2) **How should the probabilities be updated for Bot 3/4? How should the probabilities be updated for Bot 8/9?**
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.**

   – Bot 1 vs Bot 2, as a function of k.

   – Bot 3 vs Bot 4, as a function of α.

   – Bot 5 vs Bot 6, as a function of k.

   – Bot 7 vs Bot 8 vs Bot 9, as a function of α.

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

Ans 1, 2, 3:

Bot 1 and Bot 2:

Bot-1 Algorithm Design:

The ship can be assumed as a 2D grid that has cells (x, y) in it. Bot 1 uses a detection grid on the ship of size (2k) + 1. At every iteration of the algorithm, Bot 1 runs sense actions to detect. If the leak is present in the detection grid, then search for the leak, else continue running detection until the bot finds the leak. Initially all the open cells are considered as POTENTIAL LEAKS.

Pseudo code for the task for Bot 1:

while current position of bot is not equal to the leak location:

- Run method CreateDetector to create a detection grid on the bot location and increment its sense action += 1.
- Check if a leak is present in the detection grid.
- If a leak is present in the detection grid, then.
  - Mark all the remaining open cells as NO LEAK ("✅") on grid
  - Mark all the cells of the detection grid as POTENTIAL LEAK with ("❌") on grid.
  - Mark current bot position with ("✅") NO LEAK on bot position
  - Check neighbors of the Bot current position,
- If neighbors to bot position is present, then move bot to that cell, and increment move action with += 1

- else, find all the POTENTIAL LEAK cells and store them in a list. Now, find the BFS to each cell to a cell in that list from bot current position and its path length - all stored in the dictionary. Find the minimum path length from the dictionary and move to that cell with minimum path length. Now, we increment move action with += length(path)
- If No Leak present in detection grid, then
  - Mark all the cells as NO LEAK within ("✅") the detection grid
  - Find the outer detection grid cells (detection grid has at least one open neighbor), Run BFS on edge detection grid cells from bot current position, find the shortest path length and move to that cell. Bot is now moved to one of the edge cells in the detection grid.
  - Now, we move the bot to the open cell from the edge of the detection grid
  - increment the bot move actions by len(path) + 1
- Mark current Bot position with "😀"

Helper Method to Support Bot 1:

```
def CreateDetector(k, grid, botpos):
```

- CreateDetector takes 3 parameters to create a detection grid for Bot 1. The loop iterates over each cell in the detector grid, calculating the coordinates (inner_x, inner_y) for each cell in the grid. If the cells are within the bounds of (0 <= inner_x < len(grid) and 0 <= inner_y < len(grid)) then cells are added to innerGridList with no block cells.

```
def get_open(self, grid, bot):
```

- Get the Open cell which are within the bound of the grid size and whose neighbors are "⬜" or "🟥" or "❌"

```
def outer_detection_cells(grid):
```

- These method returns the list of cells which have at least one open cell to the NO LEAK ("✅") cells. So, all coordinates which are edge of the no leak cells ("✅") will be returned.

```
def find_shortest_path(index, original_grid, bot_no, start, end):
```

- This method finds and returns the shortest path possible from the start location to the end location in the 2D Grid. This Method implements BFS.

How does Bot 1 factor in available information to make more informed decisions on what to do next?

- Bot 1 runs a detection grid from its current location. The detection grid determines our Bot if there is a leak in the range it detected.

If the leak is present: the Bot marks all the open cells outside detection range as no leak cells. Bot 1 moves to a neighbor within the detection range and eventually explores cells until it finds a leak.

If the Bot was informed that there is no leak in the current detection range then it will move to the edge of the detection range first and then move to an open cell which is nearest to the current detection range. Now, the bot will run the detection grid again to see if there is a leak or not in that detection range and continues to do so until it finds a leak.

Bot-2 Algorithm Design:

All the details mentioned in Bot 1, are the same as in Bot 2. On top of that Bot 2 implements how much and where to move. Based on that, Bot 2 Optimize the Sense action. Now the update that we will make in the pseudo code for Bot 1 is that if we detect no leak in the current detection grid we move the bot such that it is 2k+1 steps away from the current bot position. If there is no such cell possible it will run as Bot 1.

Helper Methods to Support Bot 2:

All the methods will be the same as bot-1 except finding the Outer cells to create a detection grid by minimizing the overlapping.

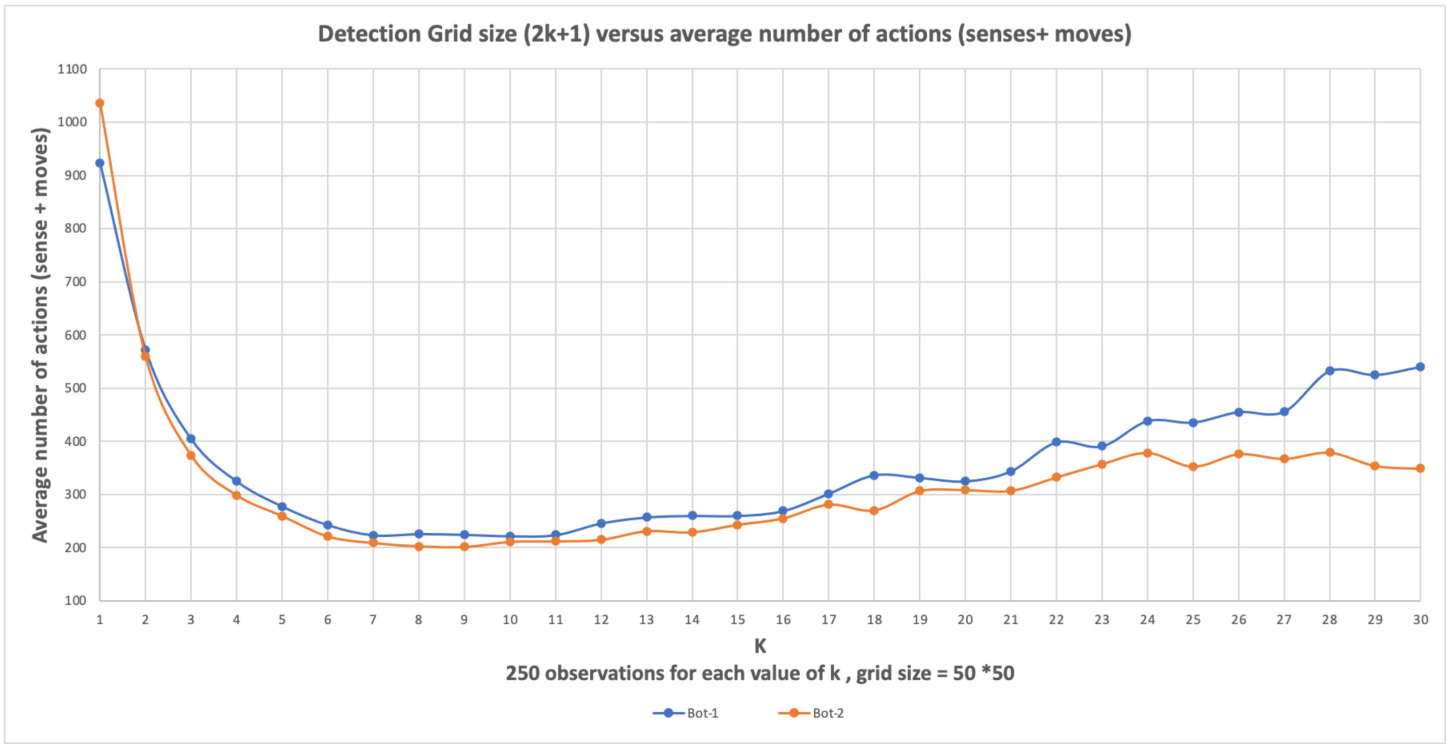
```
def out_cells_bot_2(k, grid):
```

- This method takes 2 parameters, the sides length of detection grid k, and grid. This method will calculate all the edge detected No leak from the grid. From all the cells, Calculating the distance by 2K. For example, if a bot side length of detection is k, then the bot will find the position at a distance of 2K+1. Method will return the result cells that can both visit to maximize the grid checking with less sensing with .

How does Bot 2 factor in available information to make more informed decisions on what to do next?

The limitation of Bot 1 is that it runs detection grid such that there will be checked cells already in the detection range. We optimize Bot 2 to avoid those overlapping cells which were already marked as no leak before and exploring new cells only. As a result, Bot 2 makes more informed decisions compared to Bot 1. If Bot 2 finds that there are no possible locations to move while avoiding overlapping, then Bot 2 will decide to move the nearest possible open cell.

Data and Analytics for Bot 1 and 2:



Summary:

- The above graph shows the relation between Bot 1 and Bot 2 at a value ranging from 1 to 25.

- Each value of k is implemented with 250 observations on 50 size grids. At every value of k averaging the number of senses and actions needed to find the leak with the number of observations performed on each bot.

Data Analysis:

- As per our algorithms for Bot 1 and Bot 2, we have optimized the next move to explore by minimizing the overlapping of the detection grid which is directly proportional to exploring more cells. Based on the above solution, we can clearly see from the chart that Bot-2 is taking less total number of actions compared to Bot-1 at each value k except at k = 1.

- As the K values increases from 1 to 7, the number of total actions to reach the leak for both Bot 1 and Bot 2 decreases, but Bot 2 has a lower number of total actions required to reach the leak compared to Bot 1.

- As the K values increases from 7 to 30, the number of total actions to reach the leak for both Bot 1 and Bot 2 increases, but Bot 2 has a lower number of total actions required to reach the leak compared to Bot 1.

Bot 3 and Bot 4:

Bot 3 Algorithm Design:

The ship can be assumed as a 2D grid which has cells (x, y) in it. Each cell has some probability of containing a leak. The probabilities will be stored in a dictionary. The keys will be (x, y) and the values will be probability values. Bot 3 starts with m open cells, and they all have equal probability of containing a leak = 1/m. In m, we exclude the start position of the bot because it's probability will be 0.

Pseudo code for the task for bot 3:

- While the current position of bot is not equal to the leak location:
    - Using Dijkstra's algorithm store shortest distance from start location to every other location in the grid in a list. To calculate probabilities and find path, we will use these distances instead of running BFS again and again.
    - Now the bot initiates a sense action to determine if there is a beep or not at that location. The P(beep at bot location | leak is at leak location) is compared with a random number generated between 0 to 1. If the probability calculated is greater than or equal to the random number generated the bot hears a beep or else, it doesn't.
    - If the bot hears a beep through its sense action we update P(leak in cell j | beep in bot location) else we update P(leak in cell j | no beep in bot location), where j is every other cell that might contain a leak. Here, we increment bot sensing by 1.
    - Now, we plan a shortest path from current bot position to a cell that has highest probability of containing a leak in the grid. To do that we first get the maximum probability from the dictionary. Then, we create a list that stores all keys or cells that have the highest probability. Now, we access the distances that we calculated using Dijkstra's algorithm and then we find the shortest path to get to each of those cells. Now we find minimum length to get to a cell that has highest probability. As there can be multiple cells of equal length, we will update the list of keys to store all keys or cells that have minimum length from bot position and highest probability

value. Finally, we choose a cell at random from this list which will be our end location. We get path by running BFS from bot position to end location.

- Now the bot explores the path that we found by updating what it knows about every cell at each move. While the length of path is not equal to 0:
  - We pop the 0^{th} key/cell from the path and move the bot to that location.
  - Then, we check if the bot location has a leak or not. If it has a leak, the task is complete or else we compute P(leak in cell j | no leak at bot position), where j is every other cell that has a probability of containing a leak.
  - Here, we increment the bot moves by 1.

Probability calculations for Bot 3:

Given: P(beep in cell i | leak in cell j) = $e^{(-\alpha(d(i,j)-1))}$, where d(i, j) is shortest path from indexes i = (a, b) and j = (f, g) in the 2D grid.

We are interested in calculating P(leak in cell j | leak not in cell i), P(leak in cell j | beep in cell i), and P(leak in cell j | beep not in cell i), where i = (a, b) is the location of the bot and j = (f, g) is every other cell that has a possibility of containing a leak in the 2D grid.

1. P(leak in cell j | leak not in cell i)

   $$= \frac{P(\text{leak in cell } j \wedge \text{leak not in cell } i)}{P(\text{leak not in cell } i)}$$

   $$= \frac{P(\text{leak in cell } j) * P(\text{leak not in cell } i \mid \text{leak in cell } j)}{1 - P(\text{leak in cell } i)}$$

   Since there is only one leak in our 2D grid for Bot 3, if there is a leak in cell j, there will not be a leak in cell i. Hence, P(leak not in cell i | leak in cell j) = 1

   $$= \frac{P(\text{leak in cell } j)}{1 - P(\text{leak in cell } i)}$$

2. P(leak in cell j | beep not in cell i)

   $$= \frac{P(\text{leak in cell } j \wedge \text{beep not in cell } i)}{P(\text{beep not in cell } i)}$$

   $$= \frac{P(\text{leak in cell } j) * P(\text{beep not in cell } i \mid \text{leak in cell } j)}{\sum_x P(\text{leak in cell } x \wedge \text{beep not in cell } i)}$$

   $$= \frac{P(\text{leak in cell } j) * P(\text{beep not in cell } i \mid \text{leak in cell } j)}{\sum_x P(\text{leak in cell } x) * P(\text{beep not in cell } i \mid \text{leak in cell } x)}$$

   $$= \frac{P(\text{leak in cell } j) * (1 - e^{-\alpha(d(i,j)-1)})}{\sum_x P(\text{leak in cell } x) * (1 - e^{-\alpha(d(i,x)-1)})}$$

3. P(leak in cell j | beep in cell i)

   $$= \frac{P(\text{leak in cell } j \wedge \text{beep in cell } i)}{P(\text{beep in cell } i)}$$

   $$= \frac{P(\text{leak in cell } j) * P(\text{beep in cell } i \mid \text{leak in cell } j)}{\sum_x P(\text{leak in cell } x \wedge \text{beep in cell } i)}$$

$$= \frac{P(\text{leak in cell } j) * P(\text{beep in cell } i \mid \text{leak in cell } j)}{\sum_x P(\text{leak in cell } x) * P(\text{beep in cell } i \mid \text{leak in cell } x)}$$

$$= \frac{P(\text{leak in cell } j) * (e^{-\alpha(d(i,j)-1)})}{\sum_x P(\text{leak in cell } x) * (e^{-\alpha(d(i,x)-1)})}$$

Helper methods to support the task for bot 3:

`def all_distances_bfs(index, original_grid, bot_no, start):`

- This method takes a start location in a 2D grid and finds the shortest distance from that cell to every other cell that has a possibility of containing a leak. This method implements Dijkstra's algorithm.

`def find_shortest_path_bot3(index, original_grid, bot_no, start, end):`

- This method finds and returns a shortest path possible from the start location to the end location in the 2D grid. This method implements BFS.

`def beep_in_i_given_leak_in_j(a, grid, start, end, distances):`

- This method implements formula that would calculate P(beep in cell i | leak in cell j). Here, a is value of alpha, original_grid is the grid, start is start cell location, end is end cell location, and distances are the precalculated distances returned by all_disatnces_bfs(index, original_grid, bot_no, start).

`def no_beep_in_i_given_leak_in_j(a, grid, start, end, distances):`

- This method calculates P(no beep in cell i | leak in cell j).

`def prob_leak_given_no_beep(a, grid, cell_probability_dict, botpos, distances):`

- This method calculates P(leak in cell j | no beep in cell i). Here, the denominator of the formula is only calculated once for faster performance.

`def prob_leak_given_beep(a, grid, cell_probability_dict, botpos, distances):`

- This method calculates P(leak in cell j | beep in cell i). Here, the denominator of the formula is only calculated once for faster performance.

`def leak_in_j_given_no_leak_in_i(cell_probability_dict, botpos, leak_in_i):`

- This method calculates P(leak in cell j | leak not in cell j).

How does Bot 3 factor in available information to make more informed decisions on what to do next?

- Bot 3 always plan the shortest path to a cell that has a high probability of containing a leak in the 2D grid. When a bot explores a new location and doesn't find a leak, the probability of that cell containing a leak will be 0. As a result, the probability of containing a leak for other cells goes up. When it hears a beep, the bot knows that the leak is nearby. When the bot is next to the leak the probability of getting a beep will be 1. When it doesn't hear a beep at its current location, the probability of the next cell containing a leak will be 0 as there is no leak nearby. So, the bot will move far from where it was towards the highest probability in hopes of finding a leak.

Bot 4 Algorithm Design:

All the details mentioned in Bot 3 are the same in Bot 4. On top of that Bot 4 implements how much to sense. Based on that, Bot 4 determines where to move next.

- The bot will only sense once at every location until it finds the first beep and then continues sensing for some fixed amount (≥ 0) of time to see if it can hear a beep every time it moves. The modification is very simple as it sounds.

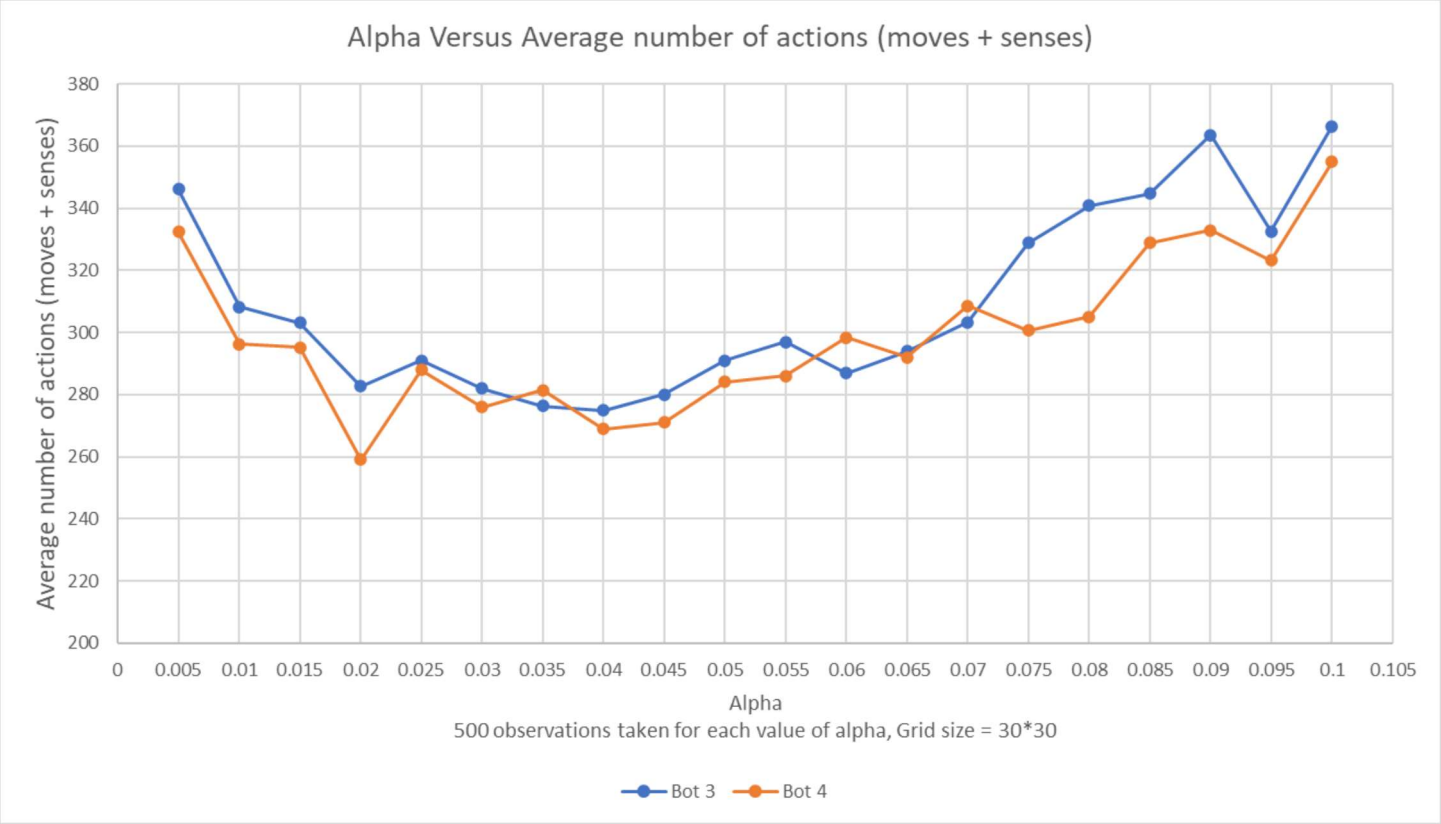
How does Bot 4 factor in available information to make more informed decisions on what to do next?

- When Bot 4 hears no beep, it knows that the leak is not nearby. As a result, the bot will sense only once at the current location and then will move on to the next location. This will continue until it doesn't hear a beep. As soon as the bot hears beep for the first time the sensing power is increased from 1 to (alpha*20). Now, the bot knows that the leak is nearby. Moving forward, the bot will sense at most (alpha*20) times. In other words, the bot keeps on sensing for maximum of (alpha*20) times until it finds a beep at that location. If it senses (alpha*20) times and doesn't find a beep, then the bot confirms there is no leak nearby. However, the bot heard a beep previously and knows that the leak will be found very soon and so its sensing power is permanently increased to (alpha*20) at every location it moves to until it finds the leak.
- The intuition behind increased sensing once a beep is heard is that the bot knows that the leak is nearby and if it hears no beep after it moves then the bot knows that the no beep might be fake.

Why is sensing increased to (alpha*20)? What is the intuition behind the formula? Why not (alpha*30) or some other value?

- We ran multiple experiments with different sensing powers to see how the Bot 4 performs. One of the key observations in all experiments was that for the alpha values between 0.005 and 0.045, Bot 4 heard beeps even if it was far from leak. In other words, the beeps that Bot 4 heard provided no useful information at all. So, when we plug these alpha values in the formula (alpha*20) we get 0 (typecasting to integer). 0 means Bot 4 wouldn't sense at all. It is fair to not sense because beeps provide no useful information. If our alpha was 0.1, (alpha*20) would return 2. So, our sensing power would increase to 2. Similarly, as the alpha values increased, the sensing power increased. To summarize, the sensing power determined by the formula (alpha*20) gave us the best results.

Data & Analysis for Bot 3 and Bot 4:



Summary: The graph above shows the performance or average number of actions (moves + senses) for Bot 3 and Bot 4 as a function of α. The dataset collected was on a grid size of 30*30. 500 observations were taken for each value of alpha. Alpha is between 0.005 and 0.1.

Data Analysis:

- Bot 3 is different from Bot 4 as they contribute to the total number of actions differently. Bot 3 always senses once. Senses for Bot 4 are determined by the formula (alpha*20). On an average, with increased sensing power moves for Bot 4 will be less than moves for Bot 3.
- Disregarding noise for couple of alpha values in the graph above, where Bot 4 didn't perform well, we observed that Bot 4 performed well mostly for all alpha values. The reason being there was balance achieved between senses and moves needed to reach the leak depending on the alpha value. Using the formula (alpha*20), we balanced out when to sense, how much to sense, and where to move. As a result, Bot 4 always got useful information with the beep it sensed between these alpha values and performed well.

Bot 5 and Bot 6:

Bot-5 Algorithm Design:

All Code and logic implemented for Bot 1 is the exactly same for Bot 5. The difference is only checking for the two leaks in the ship. Bot will find the first leak and remove it when entered and then find the second leak until it is removed. Bot will track the leak cells individually.

How does Bot 5 factor in available information to make more informed decisions on what to do next?

Bot 5 makes decisions the same way as Bot 1 makes decisions. The Limitation of Bot 5 is that it thinks of leaks as independent of one another, which will find one leak and then find the second leak. This would increase the total action to find the leaks in the ship.

Bot-6 Algorithm Design:

Bot 6 contains two leaks, instead of finding one by one, creating a data structure which stored the keys and values. Keys are the combinations of the open cells in the grid and its default value 1.
Using the combinations, we track the POTENTIAL LEAK and NO LEAK for each possible pairs of (cell_i, cell_j) in the grid. Based on the detection grid results for leak, it will update the combinations. This uses the similar concept of how a Bot 2 decides where to move with the minimum overlapping. We use a Priority list, to Move to the nearer cells first before exploring the cells that are farther from the current bot position.

Helper Method for task bot 6

```
def get_next_move(self, combination_dict, detection, bot_pos):
```

- if global len(priority list) != 0 , then if bot_pos in priority remove from priority.
- Now, find the priority cell distance from the bot_pos using Manhattan distance. Get the minimum distance and its cell. return method cell

If the priority list is empty: then find the cells from the combination. And create a cell_distance dictionary.

1) First calculate the distance from the bot pos to combinations[0]
2) Check with if condition that distance > k and distance <= 2k+1 and (bot_pos[0] == combination[0][0] or bot_pos[1] == combination[0][1])
3) Add all the distances as value and combinations[0] as keys to cell_distance dict
4) Find the maximum distance and its respected key (cell), return maximum_key

If maximum_key is empty, then return None.

```
def find_distance(self, start, end):
```

- Used to calculate the manhattan distance from the starting and ending cell.

```
def dict_clean_up(self, index, remove_list):
```

- dict_clean_up takes 2 parameters in which at index 0, if the item is present in combination, then it will be removed from the dictionary. On the other hand, at index 1, if the item is present in combination, other than the present item all other items will be removed.

Pseudo code for the task for Bot 6:

- Get All the Open cells from the grid and use the dictionary to store the combinations as keys. (To create combinations used from itertools import combinations)

While True:

- Check all the conditions to break the while loop
- If first leak found, then increment leak += 1, dict_clean_up(leakpos_1)
- If second leak found, then increment leak += 1, dict_clean_up(leakpos_2)
- If first leak found and leak == 2 then break,
- If second leak found and leak == 2 then break
- Run method CreateDetector to create a detection grid on the bot location and increment its sense action += 1.

- Check if leak is present in detection grid.

- If leak is present in detection grid, then.
- Mark all the cells of the detection grid as POTENTIAL LEAK with ("❌") on grid if any of the leak is present in the detection grid. Also, append the cells into the priority list
- If, neighbors present move to that cell, and increment move action with += 1. Also add cell to visited list.
- else, find all the POTENTIAL LEAK cells store into negative cell list. Now, find the BFS to each cell to negative cell from bot current position and its path length - all store in dictionary. Find the minimum path length from the dictionary and move to that cell with minim path length. increment move action with += length and add the cell to visited

- If No Leak present in detection grid, then
  - Mark all the cells as NO LEAK with ("✅") on detection grid
  - The confirmed NO LEAK cell in detection grid, remove those cells from the priority list. and add those cells to visited.
  - Remove duplicated in the priority cells.
  - Find the possible nearest cells that allows bot to minimize the overlapping of the detection grid based on the combinations, detection grid, and bot current position.
  - If end Cell is not None
    - If that nearest possible cell found in endCell, then apply BFS to that cell from the current bot position.
    - Check if the leaks are present in the path
      - If leak is present in the path, then increment the move action with += steps in path
      - Also, increment the number of leaks += 1
      - If both leaks found in path, then, if leak == 3, then break
    - else, No leak present in the path:
      - bot current position = last cell of path
      - increment the moves needed complete the path move action += Len(path).
      - Also, Add the path to visited list and if cells in path in priority remove it. (No priority)
  - else no possible nearest endCell = None found then find the nearest possible open cell in the grid.
  - Update the botpos and add the move action increments += Len(path)
  - Also, if cells in path in priority, then remove it from priority. (No priority)

If visited:

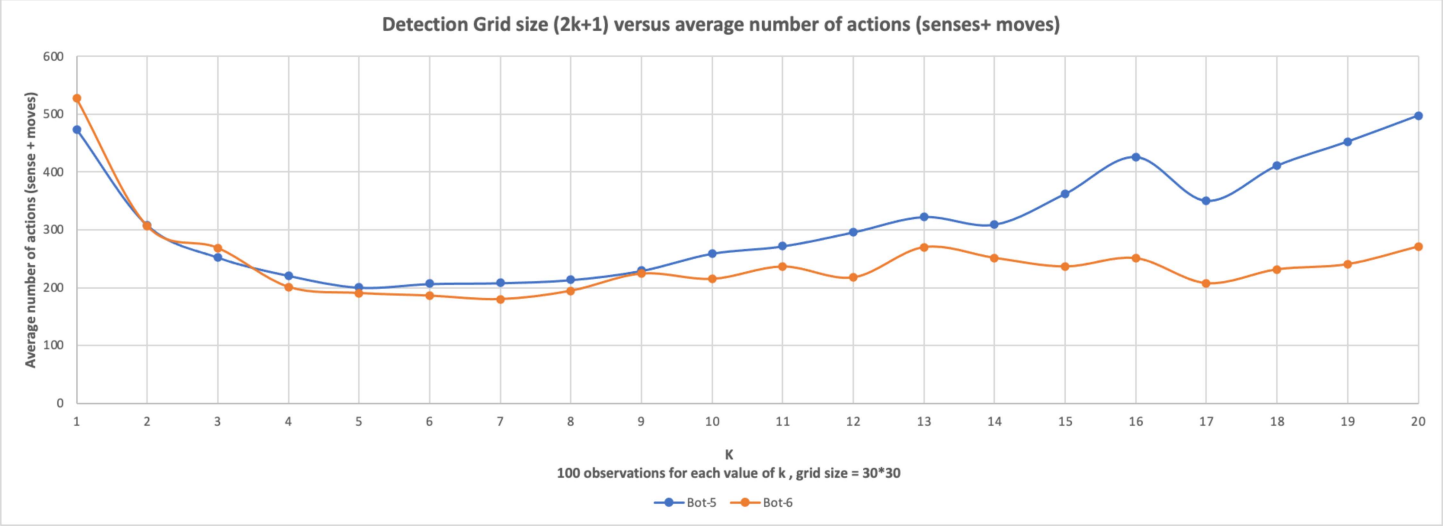
Leak in visited: then remove from visited cell.

Perform dictionary clean based on the visited cell. (Remove all the visited cells which are present in combinations)

How does Bot 6 factor in available information to make more informed decisions on what to do next?

Bot 6 takes more informed decisions and steps to find the leak in a very smaller number of actions. Firstly, using the combinations to keep track of remaining pairs available and prioritize those combinations based on the distance from the bot position. Every iteration, the prioritized cell is selected in such a way that the position avoids the overlapping. If the Bot detects in the detection grid, if leak is present then the cell containing the detection grid priorities are very high. This avoids bot 6 to explore unnecessary cells in the grid. Once, the first leak is found then, remove all the remaining pairs in combinations and only keep the pairs of leaks found. Dictionary will only have (one, many) → (leak_cell, possible_cell) combinations. Using a priority list, we can find the pair to visit and find the second leak.

Data and Analytics for Bot 5 and Bot 6:



Summary:

- The above graph is the relation between Bot 5 and Bot 6 at a value of ranging 1 to 20 .
- Each value of k is implemented with 100 observations on 30 size grids. At every value of k averaging the number of senses and actions needed to find the leak with the number of observations performed on each bot.

Data Analysis:

- As per our algorithms for Bot 5 and Bot 6, Since Bot 5 only focuses on finding leaks individually compared to Bot 6 finding the leaks in Pairs with combinations and avoiding the overlapping of detection grid results into the significantly less total action taken by Bot 6 compared to Bot 5. Overall, with the optimization of the overlapping detection grid, using the pair combinations helped Bot 6 find the double leak with the least number of actions compared to Bot 5 finding cells individually.

- As the K values increases from 1 to 4, the number of total actions to reach the leak for both Bot 5 and Bot 6 decreases, but Bot 6 has a higher number of total actions required to reach the leak compared to Bot 5. Bot 6 didn't perform well because the size of the detection grid is small. As the detection is too small, there are few

cell which are island cells meaning that they are missed out by the detection range and to explore those cells total moves increases and Bot ends up not performing well. Until the island cells are not explored the combinations relating to those cells still exist and exploring them will take more moves. As the k values gets higher chances of having island cells get lower and the bot performs well.

- As the K values increases from 4 to 20, the number of total actions to reach the leak for both Bot 5 and Bot 6 increases, but Bot 6 has a lower number of total actions required to reach the leak compared to Bot 5.

Bot 7, Bot 8, & Bot 9:

Bot 7 Algorithm Design:

All the code and logic implemented for Bot 3 is the same for Bot 7. The difference is that there are two leaks on the grid. Bot 7 doesn't update the probabilities the right way. It updates thinking that there is just one leak in the grid. The Bot starts by finding one leak at a time. While finding the first leak, the Bot may also find second leak if it comes on the way.

How does Bot 7 factor in available information to make more informed decisions on what to do next?

- Bot 7 makes decisions the same way Bot 3 makes decisions. The limitation of Bot 7 is that it thinks of leaks as independent of one another, which is not correct as finding one leak would influence the probability of finding another leak in the grid.

Bot 8 Algorithm Design:

The ship can be assumed as a 2D grid which has cell pairs ((x1, y1), (x2, y2)) in it. Each pair has some probability of containing a leak. The probabilities will be stored in a dictionary. The keys will be ((x1, y1), (x2, y2)) = (i, j) and the values will be probability values i.e. P(leak in i, j). Bot 8 starts with m open cells, and so total number of open cell pairs possible are m*(m-1)/2. At start, all the pairs have equal probability of containing a leak = 2/m*(m-1). In m, we exclude the start position of the bot because it's probability will be 0.

Pseudo code for the task for Bot 8:

- While both leaks not found:
    - Using Dijkstra's algorithm, store shortest distance from start location to every other location in the grid in a list. To calculate probabilities and find a path, we will use these distances instead of running BFS again and again.
    - Now the bot initiates a sense action to determine if there is a beep or not at that location. The P(beep at bot location | leaks at leak locations) is compared with a random number generated between 0 to 1. If the probability calculated is greater than or equal to the random number generated the bot hears a beep or else, it doesn't.
    - If the bot hears a beep through its sense action, we update our dictionary using P(leak in cell i, j | beep at bot location) else we update using P(leak in cell i, j | no beep at bot location), where i, j is all possible pairs that might contain the leaks. Here, we increment bot sensing by 1.
    - Now, we plan a shortest path from current bot position to a cell that has highest probability of containing a leak in the grid. To do that we first get the maximum probability pairs from the

dictionary. Then, we create a list that stores all pairs that might have the highest probability. For each unique cell in these pairs, we find the cells with the highest probability of containing a leak using method get_cell_probability(pair_dictionary, cell), which marginalizes to get the individual probability of a given cell from the dictionary of probability of leak in pairs. We then find the keys or cells that have the highest probability. Now, we find the shortest path length to reach one of these high probability cells using the distances that we calculated by Dijkstra's algorithm. Since there can be multiple cells that can be reached with same path length, we create a list of all such cells i.e. we now have a list of cells with highest probability and minimum path length. Finally, we choose a cell at random from this list which will be our end location. We get path by running BFS from bot position to end location.

- Now the bot explores the path that we found by updating what it knows about every cell at each move. While the length of path is not equal to 0 or both leak locations are found:
  - We pop the 0^{th} key/cell from the path and move the bot to that location.
  - Then, we check if the bot location has a leak or not.
    - If it has a leak, we then update the probabilities in our dictionary by calling the method to compute P (leak in cell i, j | leak in k), where i, j are all the possible pairs that might contain the leaks. If this is the second leak, the task is complete.
    - If it doesn't have a leak and it's not the position of leak previously identified, we then update the probabilities in our dictionary by calling the method to compute P(leak in cell i, j | no leak in k), where i, j are all the possible pairs that might contain the leaks.
  - Here, we increment the bot moves by 1.

Probability calculations for Bot 8:

Given: P(beep in cell k | leak in cell i, j) = P(beep in cell k | leak in cell i) + P(beep in cell k | leak in cell j) - P(beep in cell k | leak in cell i)* P(beep in cell k | leak in cell j) = $e^{(-\alpha(d(k,i)-1))} + e^{(-\alpha(d(k,j)-1))} - e^{(-\alpha(d(k,i)-1))} * e^{(-\alpha(d(k,j)-1))}$, where d(a, b) is shortest path from cell a = (x1, y1) to cell b = (x2, y2) in the 2D grid. cv

We are interested in calculating P(leak in cell i, j | leak not in cell k), P(leak in cell i, j | leak in cell k), P(leak in cell i, j | beep in cell k), and P(leak in cell i, j | no beep in cell k), where k = (a, b) is the location of the bot and i, j = (cell_1, cell_2) is every other pair of cells that has a possibility of containing a leak in the 2D grid.

1. P(leak in pair i, j | leak not in cell k)
   $= \frac{P(leak\ in\ i,j)P(leak\ not\ in\ k\ |leak\ in\ i,j)}{P(leak\ not\ in\ k}$
   Since there are only two leaks in our 2D grid for Bot 8, if there is a leak in cell i,j, there will not be a leak in cell k. Hence, P(leak not in cell k | leak in i,j) = 1
   $= \frac{P(leak\ in\ i,j)}{P(leak\ not\ in\ k)}$

2. P(leak in pair i, j | leak in cell k)
   $= \frac{P(leak\ in\ i\ and\ leak\ in\ j\ and\ leak\ in\ k)}{P(leak\ in\ k)}$
   $= \frac{P(leak\ in\ i\ and\ leak\ in\ j)*P(\ leak\ in\ i\ and\ leak\ in\ j)}{P(leak\ in\ k)}$
   => If k != i and k != j: P(leak in k | leak in i and leak in j) = 0
   Hence, here P(leak in i, j | leak in k) = 0

=> Otherwise: P(leak in k | leak in i and leak in j) = 1

So, here we get: $P(leak\ in\ cell\ k) = \frac{P(leak\ in\ i\ and\ leak\ in\ j)}{P(leak\ in\ k)}$

Now we need to find P(leak in k):

$P(leak\ in\ k) = \sum_{k,k'} P\left(leak\ in\ k,\ k'\right)$, where (k, k') is a pair that might be the leaks

3. P(leak in cell i AND leak in cell j | beep in cell k)

$= \frac{P(leak\ in\ cell\ i\ AND\ leak\ in\ cell\ j)*P(beep\ in\ cell\ k \mid leak\ in\ cell\ i\ AND\ leak\ in\ cell\ j)}{P(beep\ in\ cell\ k)}$

$= \frac{P(leak\ in\ cell\ i\ AND\ leak\ in\ cell\ j)*P(beep\ in\ cell\ k \mid leak\ in\ cell\ i\ AND\ leak\ in\ cell\ j)}{\sum_{(a,b)} P(leak\ in\ cell\ a\ AND\ leak\ in\ cell\ b)*P(beep\ in\ cell\ k \mid leak\ in\ cell\ a\ AND\ leak\ in\ cell\ b)}$

⇨ Now, we need to find P(beep in cell k | leak in cell i AND leak in cell j)

P(beep in cell k | leak in cell i AND leak in cell j)

= P((beep in cell k DUE TO leak in cell i) OR (beep in cell k DUE TO leak in cell j) | leak in cell i AND leak in cell j)

= P(beep in cell k DUE TO leak in cell j | leak in cell i AND leak in cell j) + P(beep in cell k DUE TO leak in cell j | leak in cell i AND leak in j) – P((beep in cell k DUE TO leak in cell i) AND (beep in cell k DUE TO leak in cell j) | leak in cell i AND leak in cell j)

= P(beep in cell k DUE TO leak in cell i | leak in cell i) + P(beep in cell k DUE TO leak in cell j | leak in cell j) – P(beep in cell k DUE TO leak in cell i | leak in cell i)*P(beep in cell k DUE TO leak in cell j | leak in cell j)

$= e^{(-\alpha(d(k,i)-1))} + e^{(-\alpha(d(k,j)-1))} - e^{(-\alpha(d(k,i)-1))} * e^{(-\alpha(d(k,j)-1))}$

4. P(leak in cell i AND leak in cell j | no beep in cell k)

$= \frac{P(leak\ in\ cell\ i\ AND\ leak\ in\ cell\ j)*P(no\ beep\ in\ cell\ k \mid leak\ in\ cell\ i\ AND\ leak\ in\ cell\ j)}{\sum_{(a,b)} P(leak\ in\ cell\ a\ AND\ leak\ in\ cell\ b)*P(no\ beep\ in\ cell\ k \mid leak\ in\ cell\ a\ AND\ leak\ in\ cell\ b)}$

Now, we need to find P(no beep in cell k | leak in cell i AND leak in cell j)

= 1 – P(beep in cell k | leak in cell i AND leak in cell j)

Helper methods to support the task for bot 8:

```
def get_cell_probability(cell_pair_probability_dict, cell):
```

- This method returns probability of a cell given the dictionary of probabilities of cell pairs.

```
def clean_up(cell_pair_probability_dict):
```

- This method deletes pairs from dictionary where probabilities of leak is determined to be 0.

`def leak_in_i_j_given_leak_in_k(cell_pair_probability_dict, k):`

- Method to update probabilities of leak in each possible pair (i, j) given leak in cell k

`def leak_in_i_j_given_no_leak_in_k(cell_pair_probability_dict, k):`

- Method to update probabilities of leak in each possible pair (i, j) given no leak in cell k

`def beep_in_k_given_leak_in_i_and_j(a, grid, start, i, j, distances):`

- Method to calculate beep probability in k given leak in (i, j)

`def no_beep_in_k_given_leak_in_i_and_j(a, grid, start, i, j, distances):`

- Method to calculate no beep probability in k given leak in (i, j)

`def prob_leak_given_beep(a, grid, cell_pair_probability_dict, botpos, distances):`

- Method to update probabilities of leak in each possible pair (i, j) given beep in cell k

`def prob_leak_given_no_beep(a, grid, cell_pair_probability_dict, botpos, distances):`

- Method to update probabilities of leak in each possible pair (i, j) given no beep in cell k

How does Bot 8 factor in available information to make more informed decisions on what to do next?

- Bot 8 always plan the shortest path by first shortlisting the pairs that have the highest probability of containing the leak, then choosing the cell with highest leak probability and minimum path length distance from the grid. When a bot explores a new location and doesn't find a leak, the probability of containing a leak for all the pairs with that cell will be 0. As a result, the probability of containing a leak for all other pairs goes up. When it hears a beep, the bot knows that at least one of the leaks is nearby. When the bot is next to the leak the probability of getting a beep will be 1. When it doesn't hear a beep at its current location, the probability of the next cell containing a leak will be 0 as there is no leak nearby. So, the bot will move far from where it was towards the highest probability in hopes of finding a leak.

Bot 9 Algorithm Design:

Our Bot 9 was built on top of the Bot 8 idea, so all the things described in Bot 8 are the same in Bot 9. On top of that Bot 9 implements how much to sense. Based on that, Bot 9 determines where to move next.

- The bot will only sense once at every location until it finds the first beep and then continues sensing for some fixed amount (≥ 1) of time to see if it can hear a beep every time it moves. We have implemented a function to determine the degree of sensing factor for optimality.

Q. How does Bot 9 determine the degree of sensing factor?

- Since, we know that lower alpha values generate more beep and vice-versa, hence in order to create a balance between the sensing factor due to alpha and this additional factor we create our function as follows:
  - Check = 30 * alpha
  - If check < 1: check =1

  We came up with formula of (30*alpha) with the similar concept as mentioned in Bot 4.

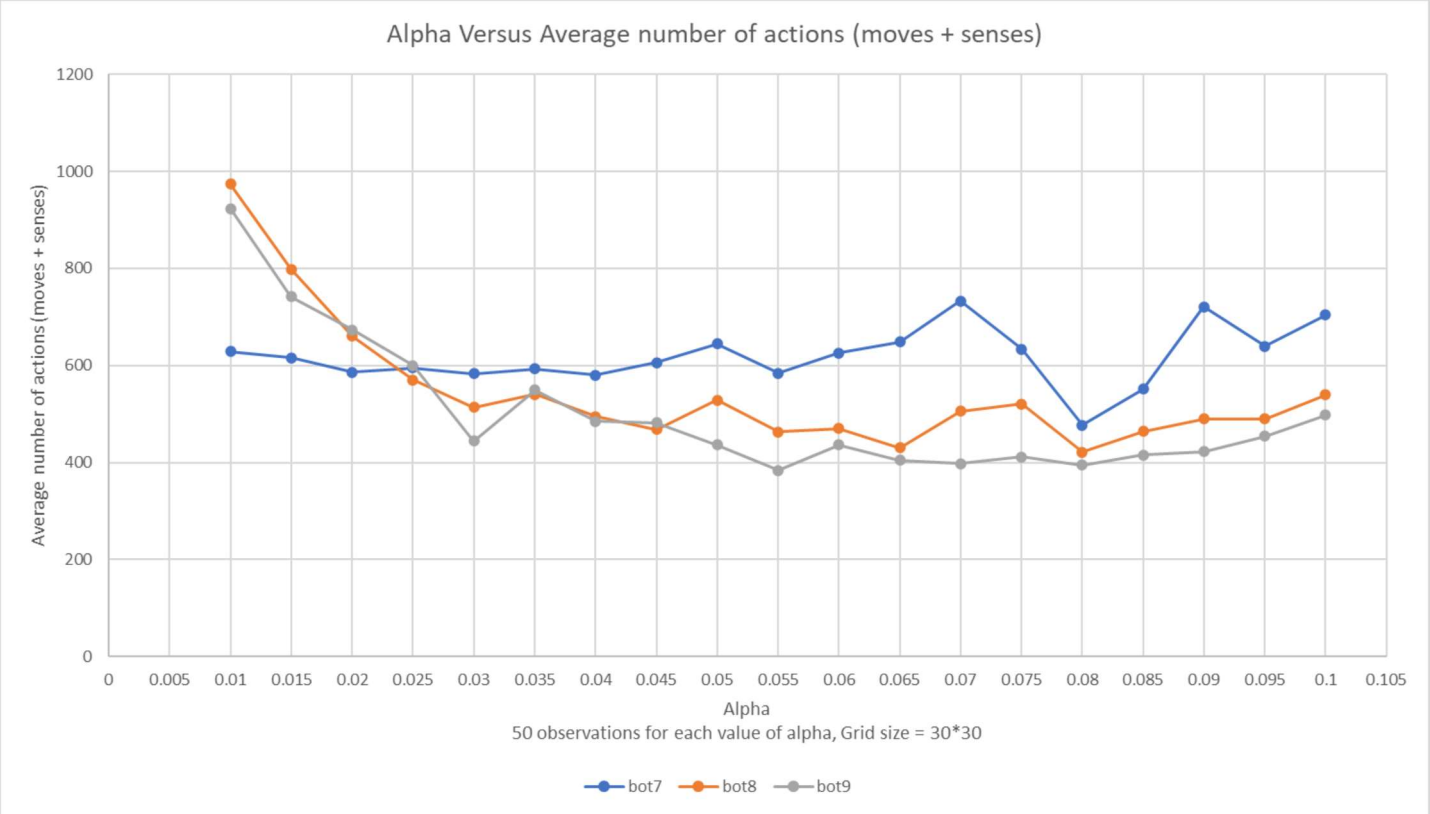
  This function is directly proportional to alpha, meaning as alpha increases and thus the original sensing power of bot decreases, we increase the bot's sensing power through our additional check variable, which is the number of counts the bot senses for beep once it realizes that it is nearby a leak. We also put

a condition of check < 1 to equate to check = 1 so that we at least make our bot check once i.e. sense as per its original sensing factor in case of lower values of alpha

Q. How does Bot 9 factor in available information to make more informed decisions on what to do next?

- When Bot 9 hears no beep, it knows that there is no leak nearby. As a result, the bot will sense only once at the current location and then will move on to the next location. This will continue until it doesn't hear a beep. As soon as the bot hears beep for the first time the sensing power is increased from 1 to check variable. Now, the bot knows that the leak is nearby. Moving forward, the bot will sense at most check times. In other words, the bot keeps on sensing for maximum of (check) times until it finds a beep at that location. If it senses for (check) times and doesn't find a beep, then the bot confirms there is no leak nearby and thus sets its sensing factor i.e. check variable back to 1.
- The intuition behind increased sensing once a beep is heard is that the bot knows that the leak is nearby and if it hears no beep repeatedly after it moves then it can again adjust its sensing power accordingly to not use the sensor inefficiently, in other words, to minimize the total number of action counts.

Data & Analysis for Bot 7, Bot 8, and Bot 9:



Summary: The graph above shows the performance or average number of actions (moves + senses) for Bot 7, Bot 8 and Bot 9 as a function of α. The dataset collected was on a grid size of 30*30. 50 observations were taken for each value of alpha. Alpha is between 0.01 and 0.1.

Data Analysis:

- Bot 7 is different from Bot 8 and Bot 9 as Bot 7 applies the idea of independent leaks whereas Bot 8 and Bot 9 try to find the leak in pairs.

- Bot 8 and Bot 9 vary as they contribute to the total number of actions differently. Bot 8 has less senses compared to Bot 9, and so the tradeoff Bot 8 has more moves than Bot 9.
- For alpha values between 0.01 and 0.025, Bot 9 didn't perform well in terms of average of total number of actions taken to find a leak. It didn't do well due to increased sensing power. For alpha values between 0.01 and 0.025, Bot 9 heard frequent beeps even when it was far from the leak.
- For alpha values between 0.025 and 0.1, Bot 9 started performing well. The reason being there was balance achieved between senses and moves needed to reach the leak. The balance was achieved because the beeps were now getting less frequent. The Bot started to get useful information with the beep it sensed between these alpha values.

Ans 4:

My ideal Bot would have following features/improvements:

- Depending on the value of alpha, first we determine how many times our Bot can sense every time it moves. This depends on alpha because we want to get useful information out of beeps we hear and avoid unnecessary beeps or no beeps that our Bot would find not useful. Once we know, how much and when our Bot can sense, we could somehow find a way to store and use the frequency of beeps heard at each timestamp to get useful information. By this way we will know if we are going near the leak or not.

  For instance, let's say Bot's sensing power is 10. The Bot will sense 10 times every time it moves and stores the frequency of how many beeps it heard. As the Bot progresses, it should make sure it follows the path of increased beep frequency. This will increase the number of senses for the ideal Bot but we would benefit from less moves once the beep is detected for the first time.

- Having implemented that, the frequency of beeps heard at a location can be used as key observation for future predictions. Prediction would be useful to verify and avoid noise created by beeps frequency. We can create a temporal model where our initial distribution will be Bot's starting location and observation can be frequency of beeps. The model will be useful due to the random nature of beeps generated. We can have our model determine or predict if we are moving towards the leak or not.

  For instance, Bot is moving towards the leak, but the beeps frequency at current location is low compared to previous location's beep frequency. This would be a false alarm for the Bot. If we used this model, prediction would consider it as a noise and still move towards leak based on previous data.

- If the moves determined by prediction are confirmed to be accurate in future, the Bot can update the observations for previous moves as it can assist with better prediction in future.

Group Contributions:

Kush Patel:

- **Bot 1, Bot 2, Bot 5, Bot 6, Report**

Pavitra Patel

- **Bot 3, Bot 4, Bot 7, Report**

Huzaif Mansuri:

- **Bot 7, Bot 8, Bot 9, Report**