**Project 2 Write-up**

**Design and Algorithms for bot 2, 4, 6 and 9**

The basic strategy I used for bot 2, 4, 6 and 9 are pretty much the same, although implemented slightly differently based on the requirements for each bot.

The strategy I used is a combination of **two algorithms**:

1. First, I will try to **identify** if there are any **outliers**/really high probabilities in a certain area around the bot, simply by seeing if there’s probabilities that exceeds a certain threshold. Notice that this does not work all of the times, but if we do end up getting cells that has probabilities higher than the threshold, I will choose the one with the closest distance to the bot as the next destination.
2. If the first algorithm doesn’t work (no outliers found), I will move on to a slower but maybe better? way of determining the best cell to move to and that is: instead of moving to the highest probability **single** cell that might contain the leak, we locate different **clusters** with the highest local probabilities and move around there instead, which we may have a better chance of finding the area that the leak is in.
   1. Here are the steps I took:
      1. **Identifying Clusters**: There definitely is better ways of doing this (like flood fill) but all I did was for each cell in the probability matrix, I identified the cluster of that cell to be the detection area around it, similar to the bot’s detection area.
      2. **Selecting a Target Cluster**: Then I selected the best cluster by finding the one with the max local probabilities, which is just the sum of the probabilities around it. If there are multiple ones, choose the one closest to the bot
      3. **Selecting the Target Cell:** Lastly, I selected the cell with the max probability amongst those in the chosen cluster, similarly, if there are multiple, choose the closest.

**Bot 3/4 Probability Calculations**

For a Probability Matrix containing the probabilities:

where is a cell in the ship at position

When the bot enter cell

* **If the bot does not find the leak in ,** 
  + For cell

(normalize values)

* + For cell
* **If the bot hears a beep in ,**
  + For cell

where

* **If the bot does not hear a beep in ,**
  + For cell

where (Probability we calculated above)

**Bot 8/9 Probability Calculations**

For Knowledge Base containing the probabilities:

where and is a pair of cells in the ship and

When the bot enter cell

* **If the bot does not find the leak in ,** 
  + For any pairs (for j = i)

(normalize values)

* + For any pairs (for j != i)

where ,

and and for all values of in the knowledge base

Kinda confusing for me, this just means that if we loop through all the possible pairs of (cell\_a, cell\_b), if either cell\_a or cell\_b is i, the probabilities of having a leak for that pair gets set to 0 (since we just found that the leak is not in i), and since we just set all those probabilities to 0, we need to normalize the knowledge base. I think this works? Not 100% sure

* **If the bot hears a beep in ,**
  + For any pairs

* **If the bot does not hear a beep in ,**
  + For any pairs

(Probability we calculated above)

**Evaluating the Performance of the Bots**

Now I run simulations to teste the performance of the bots, to make things consistent, I pre-generated 10 different 50x50 ships, each bot will be tested on those same ships with different parameters.

For each Bot, I will run the simulation 10 times for each parameter (for now) and average the Actions taken and evaluate the performance on those I will be using a range of different k and a values below:

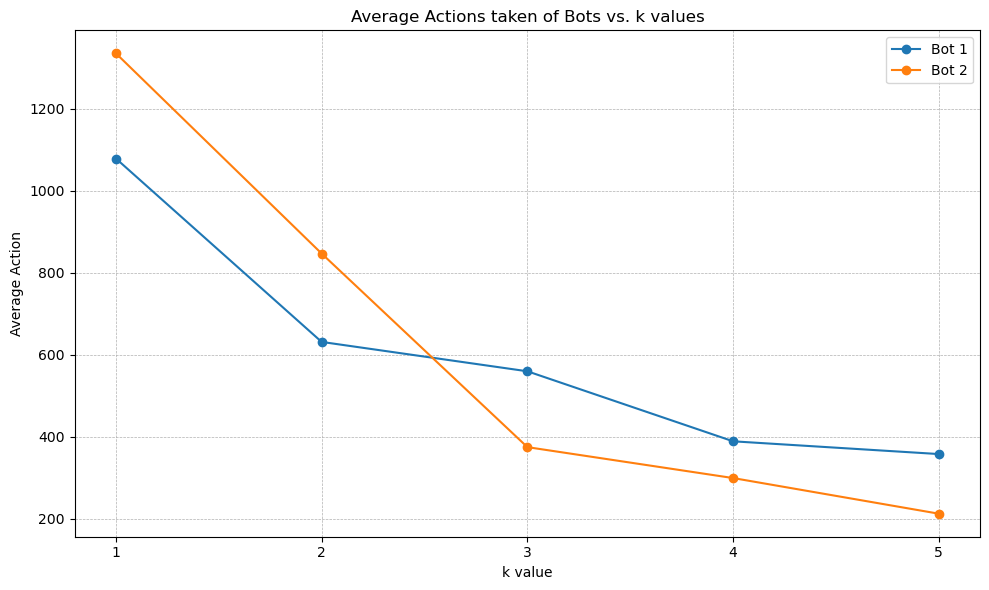
k\_values = [1, 2, 3, 4, 5]

a\_values = [0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5]

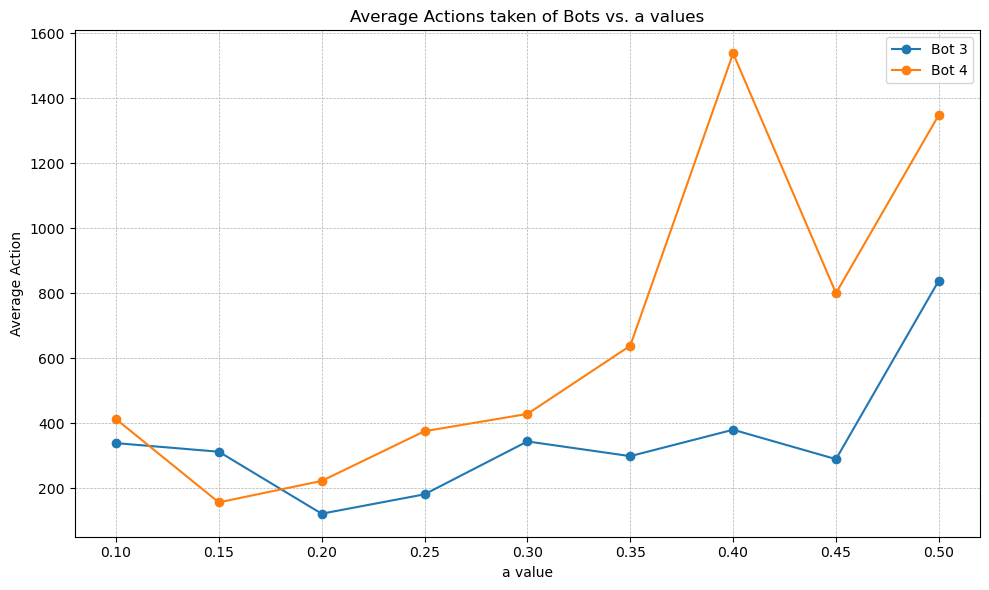
After collecting all the data, here is the graphs of the performance for the bots

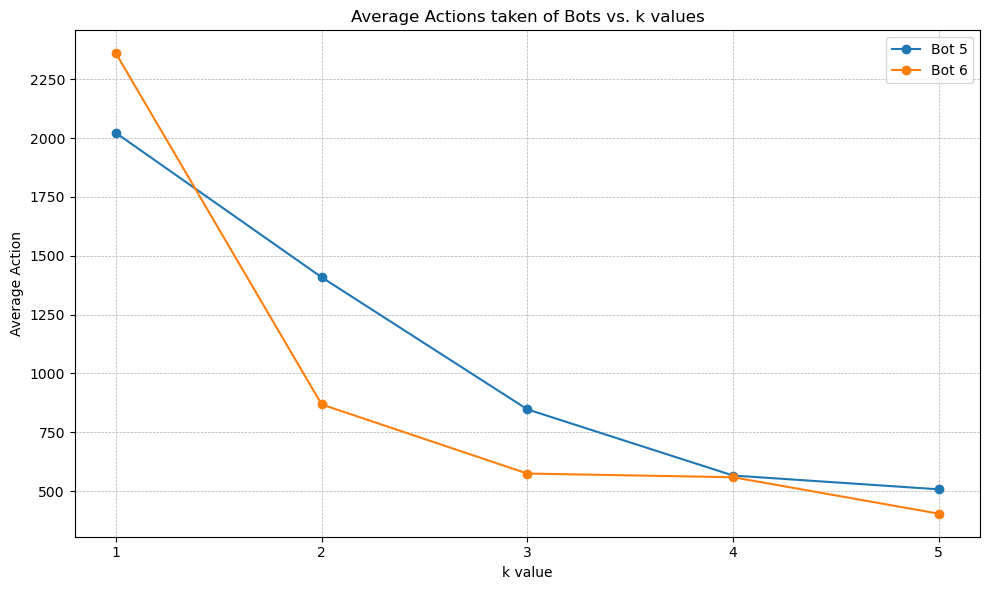
(data can be found in the testing\_results\_graphs jupyter notebook or testing\_logs.txt)

**Bot 1 vs Bot 2, as a function of k**

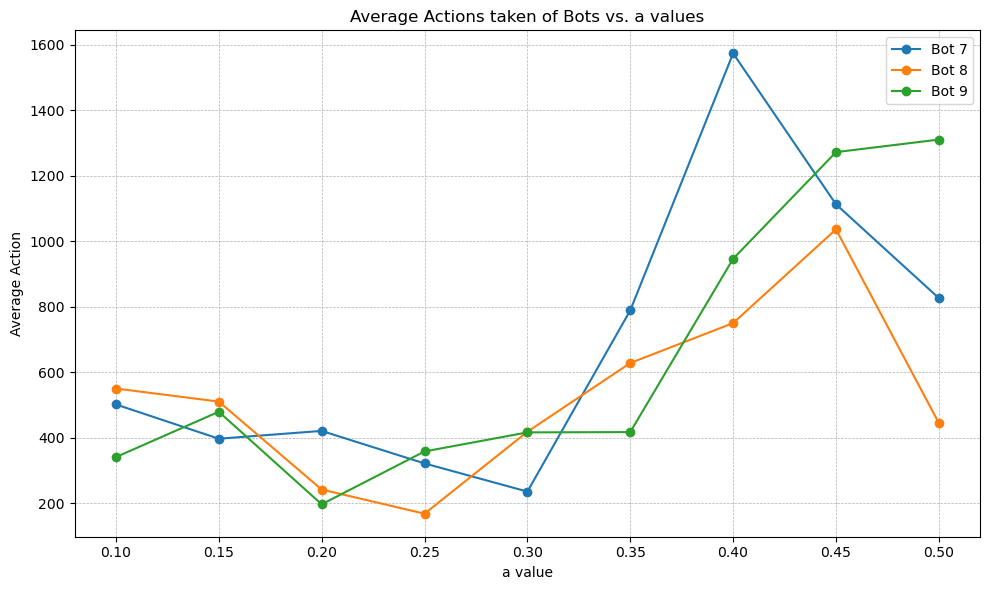


**Bot 3 vs Bot 4, as a function of a**

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



**Bot 7 vs Bot 8 vs Bot 9, as a function of a**



In conclusion, an ideal bot would definitely use more data in the knowledge base that we have and find more patterns using other algorithms, possibly incorporating machine learning to minimize the number of actions taken and improving its performance over time. Additionally, the bot should also be able to adeptly update its internal probability models based on new information, ensuring that its decision-making process remains relevant as the situation changes.