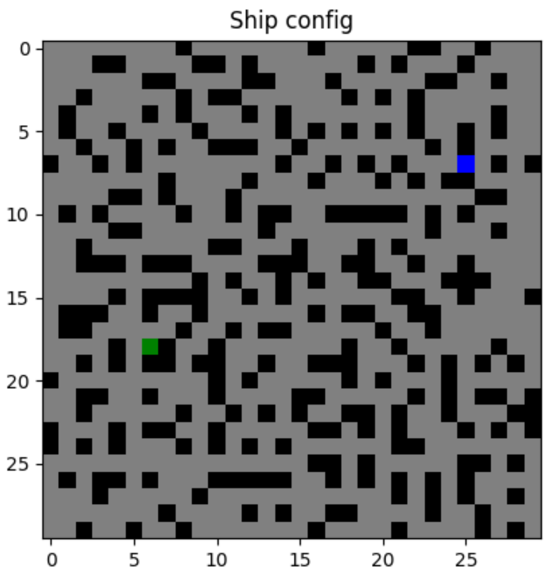
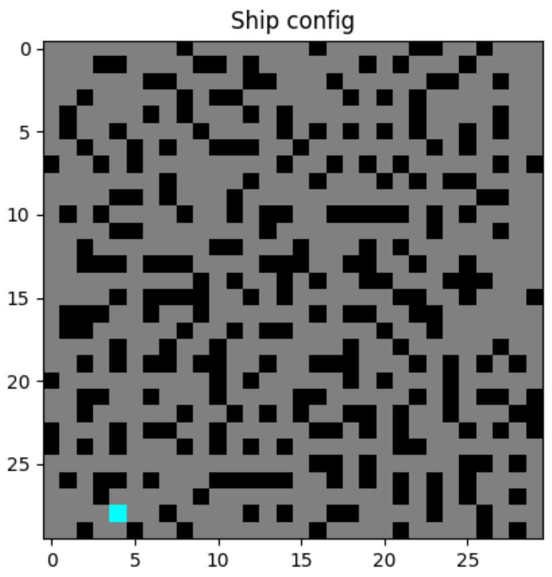




Space Rats

PROJECT 2 WRITEUP

Abhishu Oza | Intro to AI | 9 April 2025

-1) The Code

- **project_2.py/project_2_rat_moves.py** contains the main code with bot1 and bot 2 run for the cases when rat does not move / does move
- **results.csv/results_rat_moves.csv** stores results of all simulations with count of actions for the cases when rat does not move / does move
- **writeup_plots.py** creates plots for the writeup using a csv of results
- **other_bot_simulations** contains code and results for various iterations of bots I tried

0) Project_2.py Outline

Setup –

- The grid values are **BLOCKED, OPEN, BOT, RAT** and **BOT_RAT**. BOT_RAT indicates both bot and rat are on same cell. Since it the bot does not know the rat is in its cell until it senses for a beep, we add this state as a grid possibility.
- **build_ship(size)** is from the previous project which builds a ship of size x size.
- **place_bot_rat(grid)** randomly places the bot and the rat on distinct open cells

Bot Actions –

- **sense_neighbors()** senses how many of the 8 surrounding cells are blocked
- **detect_rat_ping()** senses for a ping which happens with probability $e^{-\alpha(d-1)}$
- **bot_attempt_move()** where bot attempts to move and reports success if it is able to move (success/failure value used during phase 1 to eliminate possibilities)

Utility functions –

- **dist(a, b)** calculates Manhattan distance from cell a to b
- **is_open()** returns a boolean for whether a cell is open or has the rat
- **beep_probabilities()** returns an array which stores the probability of getting a beep for all possible distances
- **astar_path()** returns a path found using the A* algorithm with weight of all cells as 1 and Manhattan distance as the heuristic.

<u>**Bot1** is the baseline bot, and **bot2** is the best bot among my tested bots</u>

1) Probabilistic Knowledge Base Update (update_Ps())

The Probabilistic Knowledge Base is represented by a set of probabilities for each open location, **p_rat** in the bot functions. We represent the event of the rat being in cell j as p_rat[j] = $\boldsymbol{R_j}$, and the event of hearing a beep at the location i as $\boldsymbol{B_i}$. Then from the project description, we calculate the value of $P(\,B_i \mid R_j\,) \;=\; e^{-\alpha(dist(i,j)-1)}$. And we initialize $P(R_j)$ as p_rat by giving equal probability to all open locations.

So we need for each cell j, the probability that the rat is in j given that we hear a beep/no beep. This can be calculated using the bayes formula -

$$P(R_j|B_i) = \frac{P(\,B_i \mid R_j\,)\cdot P(R_j)}{P(B_i)}, \qquad P(R_j|\overline{B_i}) = \frac{P(\,\overline{B_i} \mid R_j\,)\cdot P(R_j)}{P(\overline{B_i})}$$

Here the denominator values can be calculated using the sum over all cells where the rat can be –

$$P(B_i) \;=\; \sum_j P(\,B_i \mid R_j\,)\cdot P(R_j), \qquad P(\overline{B_i}) \;=\; \sum_j P(\,\overline{B_i} \mid R_j\,)\cdot P(R_j)$$

So in update_Ps() function, if a beep is sensed, it calculate the set of values **p_beep_given_rat_times_p_rat** = $\{\boldsymbol{P(\,B_i \mid R_j\,)\cdot P(R_j)} : \boldsymbol{j \text{ is an open cell}}\}$, and use it to calculate the bayes equations. We update the knowledge base by setting **p_rat[j]** = $\boldsymbol{P(R_j|B_i)}$. Similarly for the no beep case.

2) My best bot, bot2

For improvement over baseline bot, I found the idea of making multiple ping sensings as the most promising.

The idea is that if we sense multiple times from the same position, cells too near to the bot cell will have a low probability, and cells too far will also have a low probability. Therefore, if the bot is in cell i and rat is in cell j, cells that are approximately at a distance of dist(i,j) (the Manhattan distance) away from cell i will end up having a high probability. Thus a fuzzy radius of dist(i,j) will form around the bot of high probabilities.

So after doing this we move some distance away from our initial position and sense multiple times again. This will create another radius of probabilities and the intersection of these is where the bot is most likely to be.

Ideally, we need to sense from 3 separate positions to get a unique localized intersection which we can move towards with confidence.

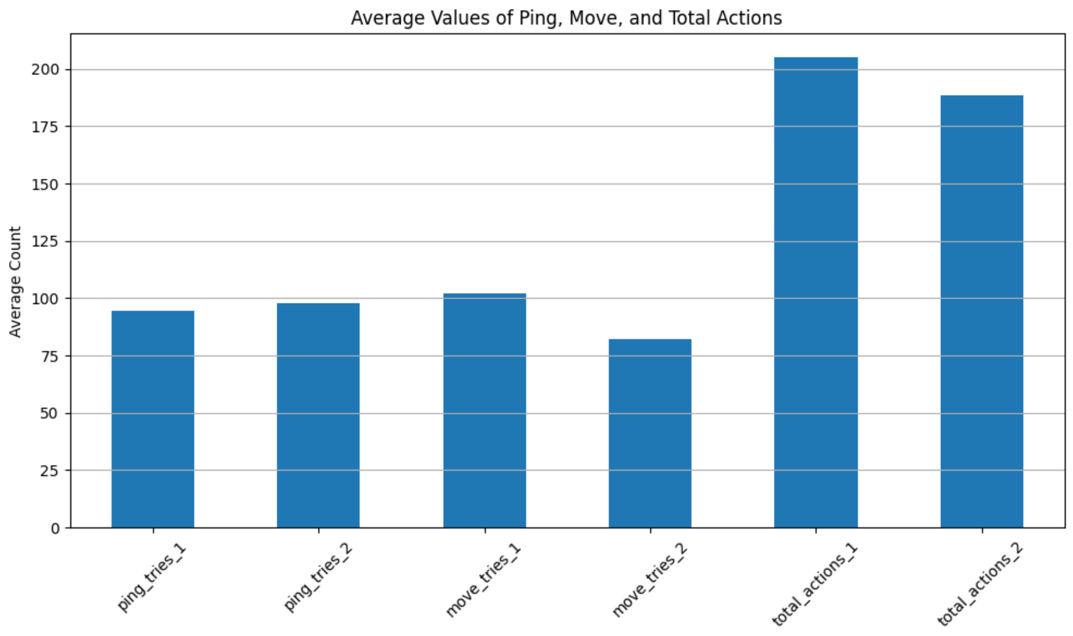
In practice, I tested many different arrangements of sensings and movements, which are recorded in other_bot_simulations. But the bot the performed the best has been reported in project_2.py as bot2. The algorithm is as follows –

1. Create a uniformly distributed set of probabilities over all non-blocked cells.
2. Sense by calling detect_rat_ping() 15 times and each time –
    a. If rat is found, stop and declare success
    b. If not, update probabilities using update_Ps().
3. Now loop forever
    a. Re-plan the shortest path to cell with maximum probability using **astar_path()**
    b. Move towards that cell for 5 steps or till cell is reached.
    c. Sense by calling detect_rat_ping() 5 times and each time –
        i. If rat is found, stop and declare success
        ii. If not, update probabilities using update_Ps().

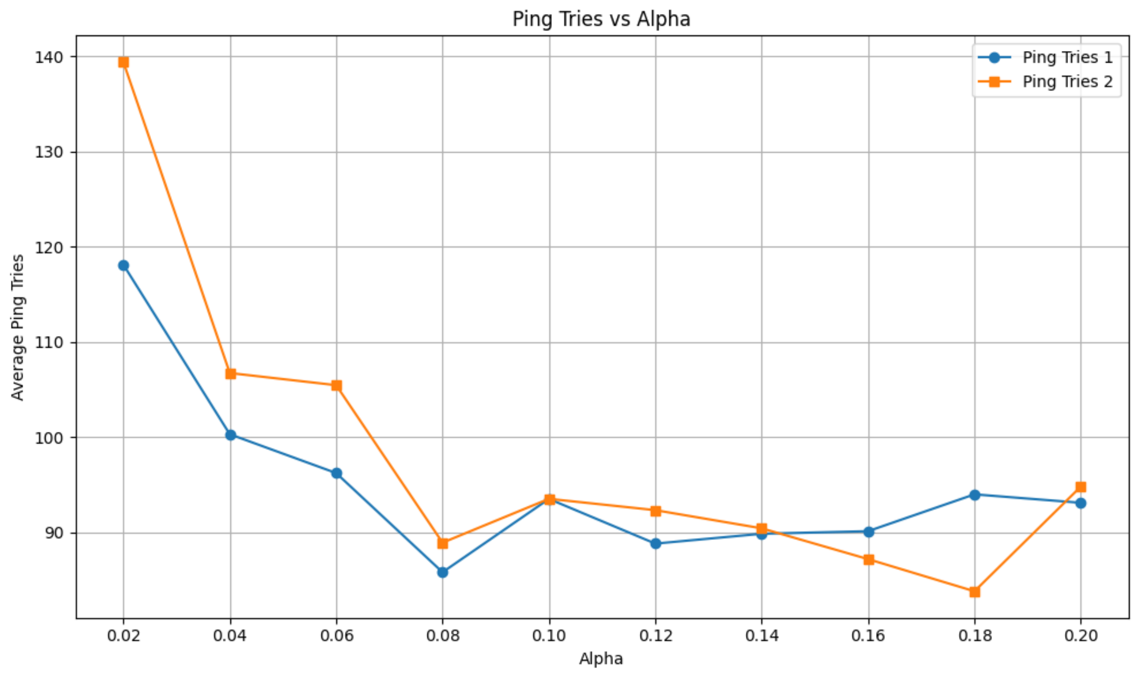
The values of 15 and 5 come empirically from testing multiple strategies.

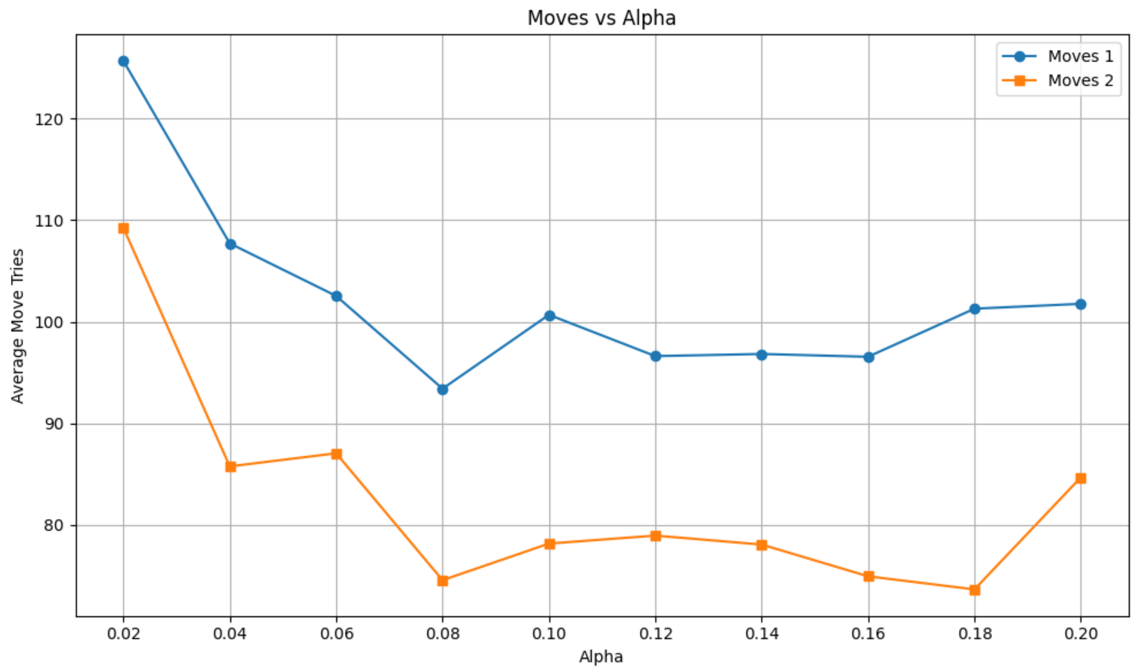
3) Bot evaluation, project_2.py

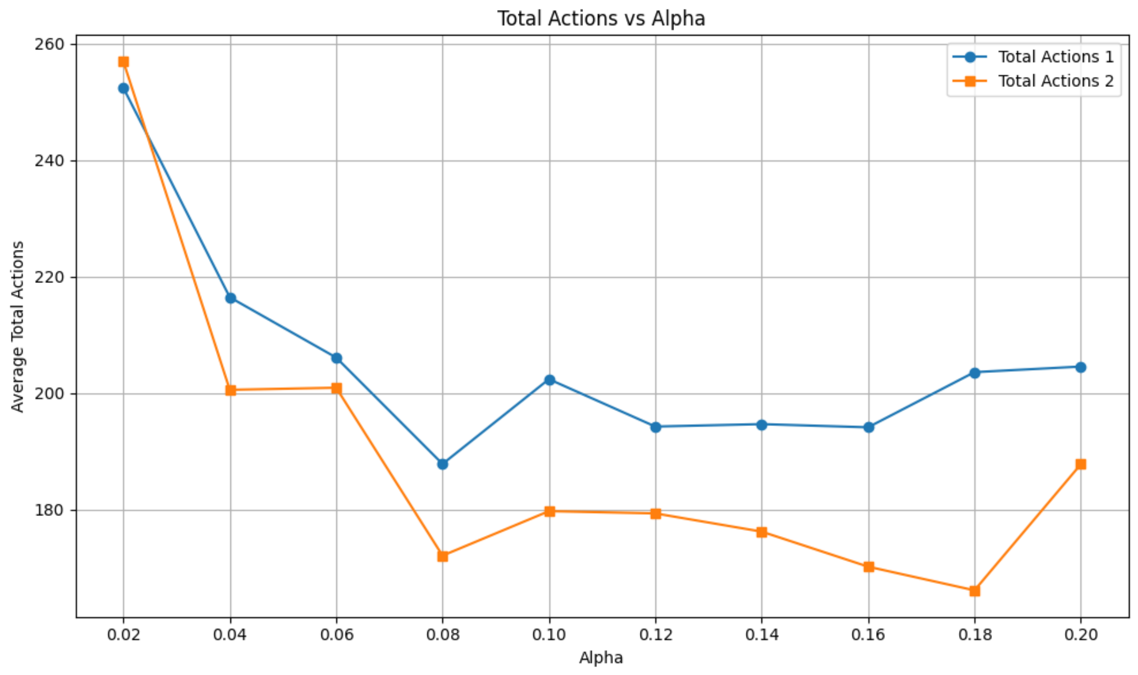
We tested bot1 and bot2 on 1000 simulations, each of which randomly selected one of 10 values of alpha between 0 and 0.2 (0.02, 0.04, 0.06, … 0.2). We plot the average of total values for ping tries, moves and total actions = ping_tries + move_tries + neighbor_sensings (neighbor sensings are only done in phase 1 and averages to around 8 actions) . The values on the bar plot are - 94.7, 98.0, 102.0, 82.3, 205.1, 188.6. **Total actions of bot2 is 91% of total actions of bot2, i.e. a 9% improvement.**



Following are the plots for actions as a function of alpha –







Observations –

- Number of ping tries for bot 2 were actually a bit larger, especially for a alpha = 0.02. This was expected because the sensor is very insensitive at low alpha.
- We were able to noticeably reduce the average number of moves by about 20%.
- As expected, performance is worse for very small alpha values but basically stagnates for values between 0.1 and 0.2. Eyeballing the total actions line plot gives improvement of about 20% in this range.
- Best performance is of bot2 at alpha = 0.18.

My speculation is that this bot performs well because around 15 pings is a good amount of information to start acting on it with moves. And then to make sure we get intersections of radii to do well we need to move at least a few moves away from our original sensing position. We also need to keep correcting ourselves every few number of moves (5 in our case) to make sure we are not led too astray.

I tried the ideal strategy of intersection of 3 sharp radii (using 25-30 pings from 3 positions 7 moves away from each other) but this did not yield in a good result and the number of pings increased by a lot

If we think about a lower bound of moves taken by the best bot, we can estimate that the bot must take an average of about 20 to 30 moves even if it knew exactly where the rat was. On top of that, we also need to keep moving and triangulate where the rat is likely to be using knowledge updates. This tells us that although the average number of moves taken by bot 2 (82.3) could be reduced further, there would be a lower bound of around 40 to 50 moves which we are not too far from.

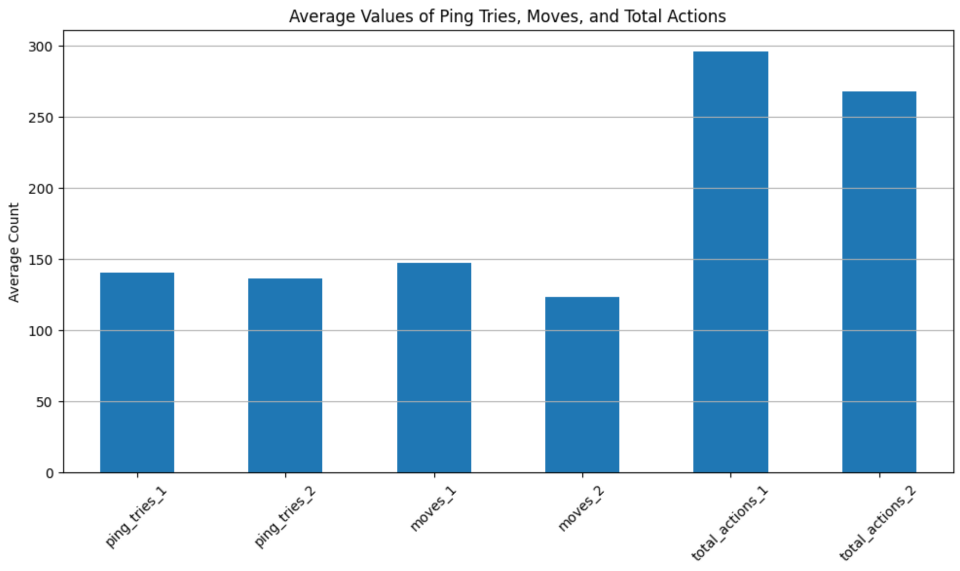
4) Moving Rat, project_2_rat_moves.py

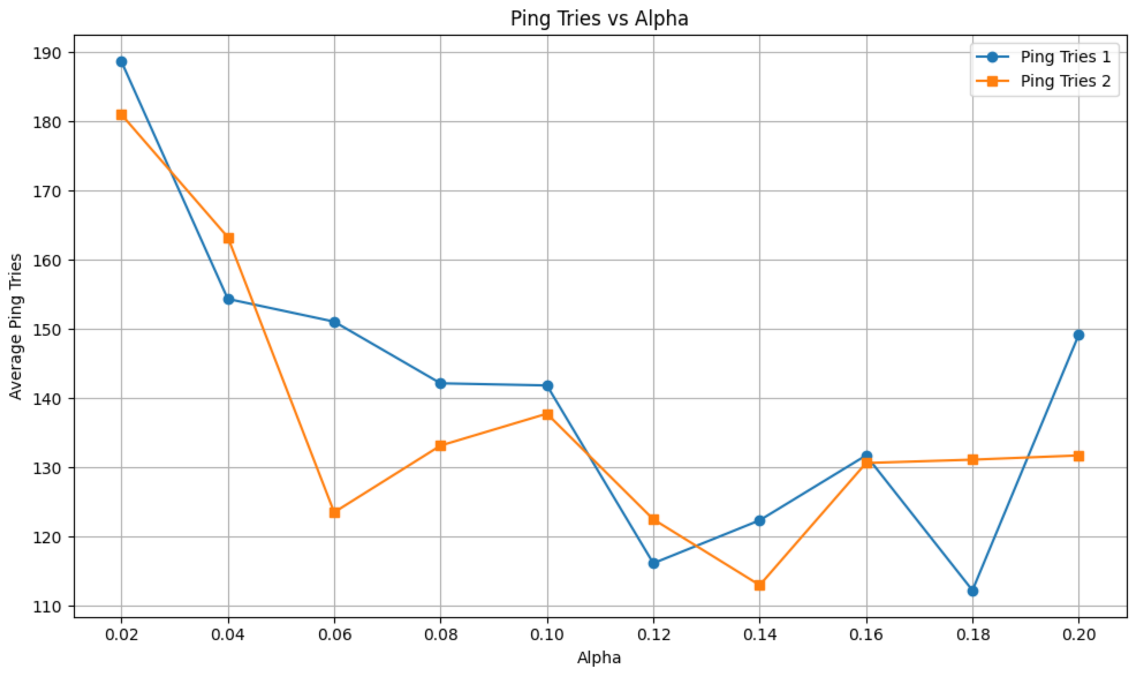
We now assume that in every timestep, after the bot takes an action, the space rat moves in a random open direction. This is implemented via **move_rat()** which is called after every action taken by the bots.

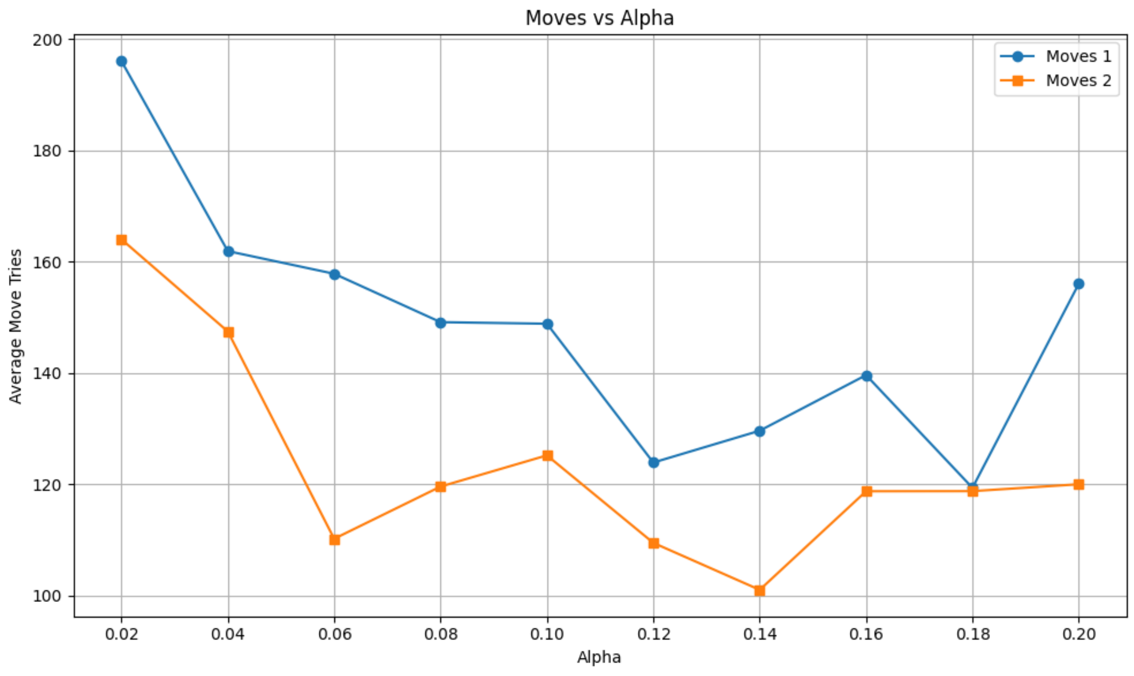
During phase 2, each time move_rat() is called, **update_Ps_after_rat_move()** is called to properly update the knowledge base. The function basically distributes each location's own probability value equally to all valid non blocked adjacent cells. By doing this step for each non blocked location, we correctly distribute the probabilities for all cells during each update. It works as follows –

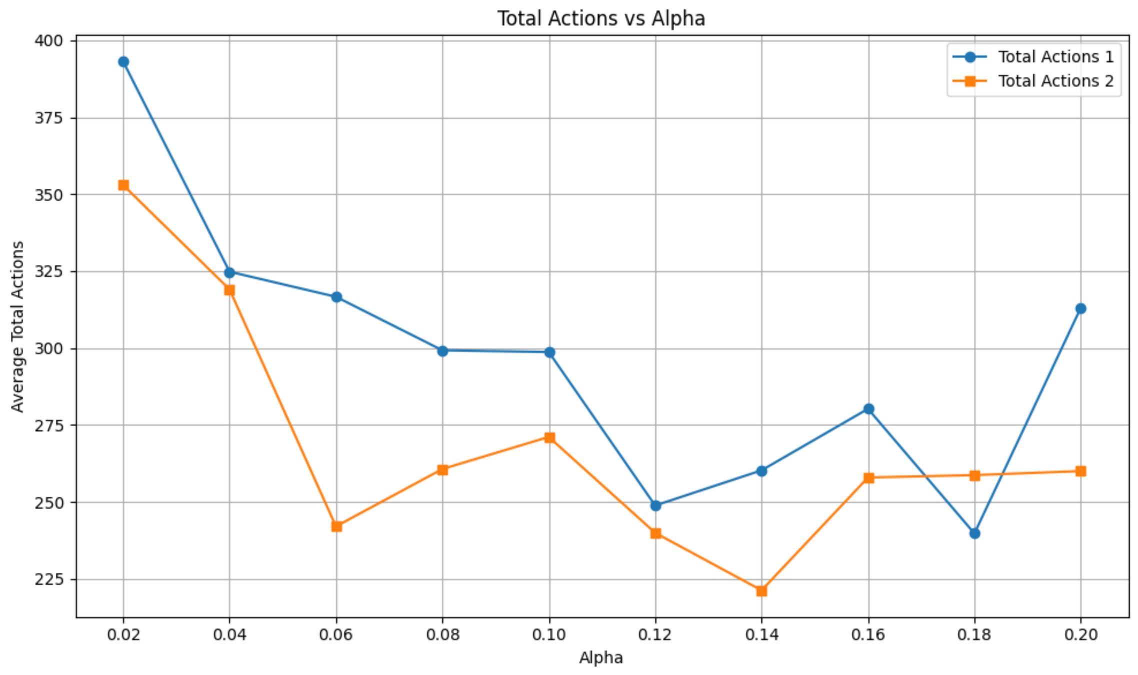
1. Create a new set **new_p_rat** which is initialised to zero for each location in p_rat (which is each non blocked location)
2. For each location in p_rat –
   a. Create **candidates** to store the location of each non blocked adjacent cell
   b. Now for each candidate –
      i. Add **p_rat**[**location**] / **length**(**candidates**) to new_p_rat[candidate]
3. Return new_p_rat

We plot the same values for this rat moving case. The values on the bar plot are - 140.2, 136.5, 147.5, 123.1, 296.0, 267.8. **Total actions of bot2 is 90% of total actions of bot1, i.e. a 10% improvement.**




Ping Tries vs Alpha
Ping Tries 1
Ping Tries 2
Average Ping Tries
Alpha
190
180
170
160
150
140
130
120
110
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0.16
0.18
0.20


Moves vs Alpha
Moves 1
Moves 2
Average Move Tries
Alpha
200
180
160
140
120
100
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0.16
0.18
0.20



Observations –

- Values as a function of alpha are noisier this time. But we again see the expected decrease in actions with increasing alpha for this range.
- There is a **41% increase** in total actions of bot2 in this case from the rat not moving case.
- For this case both the number of pings and moves taken is lower for bot2 than bot1.

I tried some more bot2 algorithms, but they were unable to improve on this bot for this case, so I kept bot2 the same as rat not moving case.

Potential improvements I was unable to address–

- Currently I did not plot the path taken / cells visited by the bot on the ship. This would help to understand when and how the bot goes astray.
- Also plotting grid probabilities at appropriate times could be used to see exactly how the properly fuzzy radii are forming and improve the bot strategy.
- Noisier plots with alpha imply more simultaneous could help clear the picture for the rat moving case.