Aliens - Rematch

Jan Marzan

April 1, 2024

Abstract

This project investigates probabilistic decision-making strategies for an autonomous agent tasked with rescuing crew members while avoiding aliens in a hostile environment. The agent operates within a simulated spaceship and utilizes both probabilistic and deterministic sensor data to make informed decisions about navigation and rescue operations. Three scenarios of increasing complexity are explored: one alien and one crew member, one alien and two crew members, and two aliens and two crew members. For each scenario, different bot strategies are developed and compared, focusing on their ability to update and use their beliefs about the locations of the crew members based on sensor information, and to make decisions that balance the goals of rescue and survival. The performance of the bot strategies is evaluated through simulations, considering metrics such as the average number of moves required to rescue all crew members, the probability of success, and the average number of crew members saved. This project provides valuable insights into the design and implementation of probabilistic decision-making strategies for autonomous agents operating in uncertain, multi-agent environments.

1 Code

The repository for this project can be found here. The code relevant to all the bots can be found in the `game/agents/` directory. In this directory you can also find the `probability_engines` directory, where modules pertaining to vectorized probability computations take place. These engines are separated from the bot logic, allowing for easy interchangeability and code reusability.

The `game/` directory contains the files that run the simulation(s) and handle data collection. To run the project yourself, first run `python3 -r requirements.txt`. Second, specify the settings in `config.json`, particularly the last 6 fields. You can specify a suite if you want, but suite 0 is reserved for visualizations (and is decided by the last 3 fields). Third, run `python3 main.py visualize`. Adding the `visualize` option in the command allows you to see the agents operating in the ship, the bot's crewmate belief map, and the bot's alien belief map.

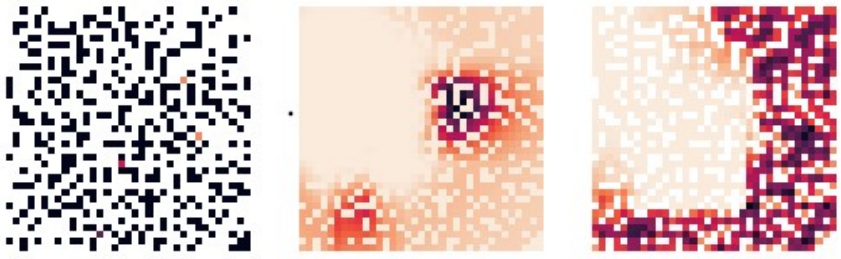

Figure 1: In the leftmost image, we see the bot in red, the crewmates in orange, and the alien in purple. The middle image shows how strongly the bot believes where the crewmates are. The right image shows where it thinks the alien might be.

2 Probability Modeling (Q2)

In order to correctly inform the bot's decisions, it is imperative that we properly use the data the bot receives from its sensors to model the bot's view of the ship, as the bot cannot directly observe the state of the ship. To achieve this, we employ a temporal probabilistic model that represent's the bot's belief about the state of the ship at each time step. This model enables the bot to maintain a probability distribution over the possible locations of the alien(s) and crew member(s), which is updated using a technique called "filtering" as we continue to collect sensor data.

Filtering is a fundamental concept in probabilistic reasoning that allows an agent to update its beliefs about the state of the world based on observations. This makes it the perfect method to use for the purposes of this project. By applying Bayesian inference principles, the bot can systematically update its belief using the likelihood of observed sensor data, resulting in a posterior probability distribution that reflects the bot's updated knowledge about the ship's state.

To formalize the filtering process, let us model the ship as a sequential Bayesian network or a Markov chain. Let the random variable X_0 be the initial state of the ship (the positions of the crew members, aliens, and the bot itself), and $X_1, X_2, ..., X_t, ...$ be the succeeding states. By the Markov chain assumption, we have that any state X_n depends only on its previous state, X_{n-1}. This makes sense when we consider the alien or the bot's position, as its current position is *primarily* decided by its previous position. Next, let the random variable Y_t be our observation of the state of the ship at time t. In our case, this is the sensor data the bot receives and the position of the bot, which we necessarily have perfect knowledge of.

With these definitions, we can now reason about two certain probabilities. The first one is the transition model:

$$P(X_{t+1} = x \mid X_t)$$

which is the probability that the ship transitions to some other state x given its current state. And the second is the observation model:

$$P(Y_t = y \mid X_t)$$

which is the probability that we make some certain observation at time t given the current state of the ship. Note that these are probabilities that we can determine from the context of the situation (to be discussed later). For the purpose of this project, for any of our alien or crew sensors, $Y_t = 1$ means the sensor was activated, and $Y_t = 0$ means it was not. Now, what we are primarily interested in is figuring out this probability:

$$\text{Blf}_{t+1}(x) := P(X_{t+1} = x \mid Y_0, Y_1, ..., Y_{t+1})$$

which is the probability that the ship is in some state x at time t given all of our previous observations that we made. This is the probability that the bots should operate on when making decisions. From class, this is derived to be

$$\text{Blf}_{t+1}(x) = \beta_{t+1}\ P(Y_{t+1} \mid X_{t+1}) \sum_{x' \in S} \text{Blf}_t(x')\ P(X_{t+1} = x \mid X_t = x')$$

where β_{t+1} is some normalization constant, and S the space of all possible states our ship can be in. From here on out we can define $\text{blf}_t(x)$ to be the "denormalized" version of $\text{Blf}_t(x)$ by omitting the normalization constant. Then, by the law of total probability

$$\beta_t = \frac{1}{\sum_{c \in S} \text{blf}_t(c)}$$

This recursive formula for Blf_t is important, as it gives us a way to update our belief of the current state of the ship using the ship's previous state and what we know about the environment the bot is operating in (i.e. transition and observation models).

Now that we have a framework for filtering, we can go over the calculations I made to properly update the Bot's belief about the state of the ship in each of the various scenarios.

2.1 One Crew Member

Let C be the random variable representing the crewmate's position on the ship, Y_t be the random variable representing our crewmate sensor data at time t, and S the set of all open cells on the ship. Note that C does not have a dependence on time, as the crewmate cannot move from one position to the other. So its position remains the same at any time t. The bot should ideally move towards positions on the ship that are highly likely to contain the crewmate. Thus, we can use our formula above and let $\text{Blf}_t(c)$ be how strongly we should believe, given our observations, that the crewmate is in cell c for all $c \in S$. As per the project requirements, at time 0 we initially set the belief of all open cells excluding the one the bot is on to be equally likely. Then we can update our belief the next time step as per our formula

$$\text{blf}_{t+1}(c) = P(Y_{t+1} \mid C = c) \sum_{c' \in S} \text{Blf}_t(c')\ P(C = c \mid C = c')$$

Notice that the observation model, $P(Y_{t+1} \mid C = c)$, is something that we already know, as it is based on whether or not we receive a beep from the crewmate. If d is the distance between the cell c and the bot's current position, then the probability we receive a beep is

$$P(Y_t = 1 \mid C = c) = e^{-\alpha(d-1)}$$

If we didn't receive a beep, we can simply take 1 minus this value.

There's a peculiar expression within the summation, namely $P(C = c \mid C = c')$, which is the probability that the crewmate is in cell c given that it was in cell c' previously. But the crewmate cannot move, so this evaluates to 0 for $c \neq c'$ and 1 for $c = c'$.

Furthermore, since the bot has perfect knowledge of the cell that it's currently in, say c_b, we can make the assumption that the crewmate is not there. Otherwise, the simulation would have already ended, or the crewmate teleported away.

With all this in mind, we can simplify to obtain the following formula

$$\text{blf}_{t+1}(c) = \begin{cases} 0, & \text{if } c = c_b \\ \text{blf}_t(c)\ e^{-\alpha(d-1)}, & \text{if received beep} \\ \text{blf}_t(c)\ (1 - e^{-\alpha(d-1)}), & \text{otherwise} \end{cases}$$

We normalize to obtain $\text{Blf}_{t+1}(c)$.

2.2 One Alien

Let A_t be the random variable representing the alien's position on the ship at time t, Y_t be the random variable representing the bot's alien sensor data at time t, and S the set of all open cells on the ship. Unlike the crewmate, the alien can move, so there exists some dependence on time. The bot should ideally avoid cells that are highly likely to contain the alien. As before, we want to know how likely the alien is in a cell given our sensor data, i.e. $P(A_t = c \mid Y_0, Y_1, ..., Y_t) = \text{Blf}_t(c)$. At time 0, all cells outside the Bot's alien sensor range are equally likely to contain the alien. Then we update our belief as per our formula

$$\text{blf}_{t+1}(c) = P(Y_{t+1} \mid A_{t+1} = c) \sum_{c' \in S} \text{Blf}_t(c')\ P(A_{t+1} = c \mid A_t = c')$$

The observation model, $P(Y_{t+1} \mid A_{t+1} = c)$ depends on whether or not the bot received a beep from its alien sensors and where c is relative to the bot. Let me denote $S_a(c_b)$ to be the set of cells within the bot's alien sensor range decided by the bot's position c_b, and clarify that $Y_t = 1$ to mean the alien sensors have went off (we detect an alien in $S_a(k)$). Thus

$$P(Y_{t+1} \mid A_{t+1} = c) = \begin{cases} 1, & \text{if } c \in S_a(c_b) \text{ and } Y_t = 1 \\ 0, & \text{if } c \in S_a(c_b) \text{ and } Y_t = 0 \\ 1, & \text{if } c \notin S_a(c_b) \text{ and } Y_t = 0 \\ 0, & \text{if } c \notin S_a(c_b) \text{ and } Y_t = 1 \end{cases}$$

The next thing to consider is the probability expression within the summation, $P(A_{t+1} = c \mid A_t = c')$. This is the probability that the alien is in cell c given that the alien was in c' in the previous time step. Note that at any position, the alien chooses at random to move to any adjacent cell that is not closed. Thus, this probability is nonzero only if c is adjacent to c', since c becomes one of the options to move to from c'. Furthermore, the probability that the alien moves from c' to any adjacent open cell is equally likely. Let me denote $n(c)$ to be the number of adjacent open neighbors that a cell c has and $S_n(c)$ the set of open cells adjacent to c, *excluding* c itself. Then we can formulate this probability as

$$P(A_{t+1} = c \mid A_t = c') = \begin{cases} 0, & \text{if } c \notin S_n(c') \\ \frac{1}{n(c')}, & \text{if } c \in S_n(c') \end{cases}$$

Thus, we have formulated all probabilities necessary to compute $\text{blf}_{t+1}(c)$. All that is left is to normalize to obtain $\text{Blf}_{t+1}(c)$. One would notice that $\text{Blf}_{t+1}(c)$ doesn't actually depend on $\text{Blf}_t(c)$, but rather on the belief of the cells adjacent to c. This is because the alien does not consider staying in the same cell an option. When visualizing the bot's alien belief on a heatmap, this creates an interesting blinking checkerboard pattern.

2.3 Two Crew Members

Let C_1 be the random variable representing the the first crewmate's position on the ship, C_2 the second crewmate's position on the ship, Y_t the random variable representing our crewmate sensor data at time t, and S the set of all open cells on the ship. As in the one crewmate case, there is no dependence on time. In this scenario, the bot is interested in determining the most likely pair of positions that the crewmates are in. In other words, for any two cells $i, j \in S$, we want $P(C_1 = i, C_2 = j \mid Y_0, Y_1, ..., Y_t) = \text{Blf}_t(i, j)$. From above

$$\text{blf}_{t+1}(i,j) = P(Y_{t+1} \mid C_1 = i,\, C_2 = j) \sum_{i',j' \in S} \text{Blf}_t(i',j')\; P(C_1 = i,\, C_2 = j \mid C_1 = i',\, C_2 = j')$$

First, we determine the observation model, $P(Y_{t+1} \mid C_1 = i,\, C_2 = j)$. Let B_1 be the event that crewmate 1 emits a beep, and B_2 the event crewmate 2 emits a beep. Also let d_1, d_2 be the distances between the bot and crewmate 1 and crewmate 2 respectively. Note that, per time step, the bot only receives a beep and only one beep if at least one crewmate emits a beep. If both crewmates emit a beep, the bot's sensor only registers this as one beep. The sensor's detection is equivalent to a logical or between B_1 and B_2. It should be mentioned that the the crewmates emit beeps *independently* of each other. Thus, the probability the bot receives a beep is given by

$$\begin{aligned} P(B_1 \vee B_2) &= 1 - P(\neg(B_1 \vee B_2)) \\ &= 1 - P(\neg B_1 \wedge \neg B_2) \\ &= 1 - P(\neg B_1)\, P(\neg B_2) \\ &= 1 - (1 - e^{-\alpha(d_1 - 1)})(1 - e^{-\alpha(d_2 - 1)}) \end{aligned}$$

I realize now that I could've just used the inclusion-exclusion principle to compute the same thing, but that's not how I did it in my code. Regardless, this gives us

$$P(Y_{t+1} \mid C_1 = i,\, C_2 = j) = \begin{cases} 1 - (1 - e^{-\alpha(d_1 - 1)})(1 - e^{-\alpha(d_2 - 1)}), & \text{if the bot received a beep} \\ (1 - e^{-\alpha(d_1 - 1)})(1 - e^{-\alpha(d_2 - 1)}), & \text{otherwise} \end{cases}$$

Now, we determine the observation model $P(C_1 = i,\, C_2 = j \mid C_1 = i',\, C_2 = j')$. This is the probability that the crewmates are in cells i and j given they were previously in cells i' and j' respectively. As in the one crewmate case, this mainly serves to be an indicator function, as the crewmates cannot move. This probability is 1 if $i = i'$ and $j = j'$, 0 else. So we can remove the summation and deal only with the case where $i = i'$ and $j = j'$.

We also have the additional fact that for the bot's position, c_b, we know that the crewmate is not there as it should have been teleported away or the simulation completed. We should update the Blf_t map to account for this fact, so that for all future updates it remains 0.

Thus, we can formulate the belief update as such.

$$\text{blf}_{t+1}(i,j) = \begin{cases} 0, & \text{if } i = c_b \text{ or } j = c_b \\ \text{Blf}_t(i,j)\ [1 - (1 - e^{-\alpha(d_1-1)})(1 - e^{-\alpha(d_2-1)})\ ], & \text{if the bot received a beep} \\ \text{Blf}_t(i,j)(1 - e^{-\alpha(d_1-1)})(1 - e^{-\alpha(d_2-1)}), & \text{if not} \end{cases}$$

As usual, we normalize to obtain Blf_{t+1}.

2.4 Two Aliens

Let $A1_t$ be the random variable representing the first alien's position on the ship at time t, $A2_t$ the second alien's position, Y_t the random variable representing the bot's alien sensor data at time t, and S the set of all open cells on the ship. As in the one alien case, both aliens will randomly move to any of the cells adjacent to the cell it's currently on. So there exists some dependence on time. The bot is interested in which cells are most likely to contain either alien, $P(A1_t = i \vee A2_t = i \mid Y_0, ..., Y_t)$, so that it may avoid them. But observe that

$$P(A1_t = i \vee A2_t = i \mid Y_0, ..., Y_t) =$$
$$P(A1_t = i \mid Y_0, ..., Y_t) + P(A2_t = i \mid Y_0, ..., Y_t) - P(A1_t = i \wedge A2_t = i \mid Y_0, ..., Y_t)$$

by the inclusion-exclusion principle. However, the aliens cannot share the same position, so the last term is actually 0. Then, by law of total probability, this gives us

$$\begin{aligned} & P(A1_t = i \vee A2_t = i \mid Y_0, ..., Y_t) \\ &= P(A1_t = i \mid Y_0, ..., Y_t) + P(A2_t = i \mid Y_0, ..., Y_t) \\ &= \sum_{j \in S} P(A1_t = i \wedge A2_t = j \mid Y_0, ..., Y_t) + \sum_{j \in S} P(A1_t = j \wedge A2_t = i \mid Y_0, ..., Y_t) \\ &= \sum_{j \in S} \text{Blf}_t(i,j) + \sum_{j \in S} \text{Blf}_t(j,i) \end{aligned}$$

where $\text{Blf}_t(i,j) = P(A1_t = i \wedge A2_t = j \mid Y_0, ..., Y_t)$. We have formulated this prior, as in the two crewmate case.

$$\text{blf}_{t+1}(i,j) = P(Y_{t+1} \mid A1_{t+1} = i, A2_{t+1} = j) \sum_{i',j' \in S} \text{Blf}_t(i',j')\ P(A1_{t+1} = i, A2_{t+1} = j \mid A1_t = i', A2_t = j')$$

Now we must determine our observation and transition models.

First, let us consider the observation model $P(Y_{t+1} \mid A1_t = i,\ A2_t = j)$, which is the probability of our alien sensor data given the positions of each alien. This is similar to the one alien case, where the probability is 0 or 1 depending on where the alien is and whether or not we receive the beep. However, there is the added caveat that we cannot rule out positions outside of the alien sensor range when we receive a beep, as the second alien may be outside of it. Let $S_a(c_b)$ to be the set of cells within the bot's alien sensor range decided by the bot's position c_b, and clarify that $Y_t = 1$ to mean the alien sensors have went off (we detect an alien in $S_a(c_b)$), and 0 otherwise. Thus

$$P(Y_{t+1} \mid A1_{t+1} = i,\ A2_{t+1} = j) = \begin{cases} 0, & \text{if } Y_{t+1} = 0 \text{ and } i \text{ or } j \in S_a(c_b) \\ 1, & \text{otherwise} \end{cases}$$

Next, we now consider the transition model $P(A1_{t+1} = i,\ A2_{t+1} = j \mid A1_t = i',\ A2_t = j')$, which is the probability that alien 1 is in cell i and alien 2 is in cell j given that they were just previously in cells i' and j' respectively. This one is a little trickier to determine, as we need to consider all possible combinations of i' and j' adjacent to i and j.

I make the assumption that the aliens move independently of each other. This isn't generally true in practice, since each alien will not move to a cell the other alien is in. However, this assumption allows me to more easily vectorize the computation in the future, so we make this assumption to favor speed over accuracy. Nonetheless, we can now express the transition model as follows:

$$\begin{aligned} & P(A1_{t+1} = i,\ A2_{t+1} = j \mid C1_t = i',\ C2_t = j') \\ &= P(A1_{t+1} = i \mid A1_t = i',\ A2_t = j') \cdot P(A2_{t+1} = j \mid A1_t = i',\ A2_t = j') \\ &= P(A1_{t+1} = i \mid A1_t = i') \cdot P(A2_{t+1} = j \mid A2_t = j') \end{aligned}$$

This is product of the same expression that we have derived before in the one alien case. Also notice that this product is 0 when i not adjacent to i' or j not adjacent to j'. As a reminder, $n(c)$ is the number of adjacent open neighbors that a cell c has and $S_n(c)$ the set of open cells adjacent to c, *excluding* c itself. Therefore, we have

$$P(A1_{t+1} = i,\ A2_{t+1} = j \mid A1_t = i',\ A2_t = j') = \begin{cases} 0, & \text{if } i \notin S_n(i') \text{ or } j \notin S_n(j') \\ \frac{1}{n(i') \cdot n(j')} & \text{otherwise} \end{cases}$$

So, in the summation we only consider i', j' if both are adjacent to i, j respectively.

Thus, we have formulated all probabilities necessary to compute $\text{blf}_{t+1}(i, j)$. All that is left is to normalize to obtain $\text{Blf}_{t+1}(i, j)$. The bot can then use this information as is, or use it compute $P(A1_t = i \vee A2_t = i \mid Y_0, ..., Y_t)$ to judge alien presence probabilities on a cell by cell basis. In the strategies I've implemented, I opted for the latter.

3 Bot Designs (Q1)

Here, I go over the strategies implemented for each bot.

3.1 Bot 1

Bot 1 uses a non-vectorized approach to computing probabilities, as I implemented and ran this bot prior to creating the probability engine modules. The algorithm to compute probabilities iterates through every cell, multiplying each one by the probability of the received sensor data and the cell's current belief.

Regardless, this bot selects the most likely cell to contain the crewmate, and uses the A* pathfinding algorithm to create a path to that cell. At each time step, the bot executes the next step in that path. Then, in the next time step it recomputes the most likely cell as well as the path to that cell.

In its pathfinding algorithm, the bot ensures that, at the very least, the first cell in its path does not contain the alien. After that, all other cells to be considered in the path are allowed to have a threshold of a 10% probability or less of containing the alien.

3.2 Bot 2

As with Bot 1, this bot also uses non-vectorized algorithms to compute probabilities. For this bot's strategy, I opted for a utilitarian approach towards finding the crewmate. That is, at any time step, we consider the utility of cells within some local search depth, then execute the next step in the path to that cell. Let i be the cell of interest. To determine the utility of the cell let me define some functions and constants (as well as the values I chose for those constants):

- $\text{Blf}_t^c(i)$: the normalized belief that the crewmate is in cell i at time t
- $\text{Blf}_t^a(i)$: the normalized belief that the alien is in cell i at time t
- $f(i)$: a function that indicates whether or not the bot has explored the cell i
- x: the utility of a cell the bot has explored (-1)
- u: the utility of a cell that the bot hasn't explored (1)
- c: the utility of a cell containing an crewmate (10)
- a: the utility of a cell containing an alien (-5)
- depth: while not used in the utility function, I want to note it is set to (7)

Thus, we can compute the utility U of any cell i as such

$$U(i) = c\,\text{Blf}_t^c(i) + a\,\text{Blf}_t^a(i) + (x + f(i)(-x + u))$$

Then, the bot searches the cells within a set search depth from its current position, finding the one that has the highest utility and constructs the path to that cell. For the implementation of its search, the bot simply performs a BFS from its current position with a maximum depth of 7.

With these specific values set for the constants, the bot has a higher preference for going towards cells that are more likely to contain the crewmate than avoiding cells with the alien. Furthermore, we have an added preference towards cells that the bot hasn't seen yet, as it knows for sure that the cells it has explored does not contain the crewmate. However, the bonus for entering unexplored cells is slight compared to the penalty for cells likely to contain the alien, allowing the bot to flee to explored cells if need be. Nonetheless, this added exploration bonus compensates for the bot's short-sightedness from performing a local search. This, in a sense, guides the future decisions of the bot, as it will always choose to avoid the areas it has already explored. Furthermore, it is better to listen for crewmate beeps from various positions throughout the ship. The exploration bonus allows the bot to reposition itself in various areas and collect sensor data.

3.3 Bot 3

This bot utilizes the same strategy as Bot 1, but with the upgraded probability engine modules that use vectorized operations to compute belief. However, this bot operates in an environment containing two crewmates yet only considers the existence of one.

3.4 Bot 4

This bot utilizes the same strategy as Bot 1 with upgraded probability engine modules, but operates under the assumption that two crewmates exist. As such, it uses the `twoCrewmate.py` probability engine. To select the target to path towards, the bot takes the pair of cells most likely to contain the crewmates, and then paths to the closest one. However, once it finds the first crewmate, the bot "collapses" the belief map generated by the probability engine into 2 dimensions given the location of the first crewmate.

Suppose we found the first crewmate at cell i. Then, we want to compute $P(C_2 = j \mid Y_0, ..., Y_t, C_1 = i)$ for all j. By definition of conditional probability

$$P(C_2 = j \mid Y_0, ..., Y_t, C_1 = i) = \frac{P(C_2 = j, C_1 = i \mid Y_0, ..., Y_t)}{P(C_1 = i \mid Y_0, ..., Y_t)}$$

Notice that the term in the numerator is just $\text{Blf}_t(i, j)$ for some fixed i. The term of the denominator can be considered some normalization constant such that $\sum_{j \in S} P(C_2 = j \mid Y_0, ..., Y_t, C_1 = i) = 1$. Thus, the bot can correctly update its strategy once it has found the first crewmate.

3.5 Bot 5

This bot utilizes the same strategy as Bot 2 with upgraded probability engine modules, but operates under the assumption that two crewmates exist. Unlike Bot 4, this bot considers the utility of cells on a per cell basis, as the probability that a pair of cells contains the crewmates is a little hard to work with. Thus, as done in section 2.4, we use the belief map to compute $P(C_1 = i \ \vee \ C_2 = i \mid Y_0, ..., Y_t)$ for each cell i.

$$P(C_1 = i \ \vee \ C_2 = i \mid Y_0, ..., Y_t) = \sum_{j \in S} \text{Blf}_t(i, j) + \sum_{j \in S} \text{Blf}_t(j, i)$$

Bot 5 utilizes this value in place of $\text{Blf}_t^c(i)$ when computing the utility of each cell. This bot also uses a similar strategy as Bot 4 when the first crewmate is found, "collapsing" the map using the position of the found crewmate, and reverts to single crewmate behavior.

3.6 Bot 6

This bot utilizes the same stategy as Bot 1 with upgraded probability engine modules, but with an additional flag set in the `oneAlien.py` module. This flag ensures that, while using the same computations as in the one alien case, the alien sensor going off does not rule out the possibility of aliens being outside of the sensor range.

3.7 Bot 7

This bot utilizes the same strategy as Bot 4 with the upgraded probability engine modules, but accounts for the fact that there are two aliens and two crewmates. As mentioned in section 2.4, we use the probability $P(A1_t = i \vee A2_t = i \mid Y_0, ..., Y_t)$ to judge the presence of aliens on a cell by cell basis, and use that to determine which cells to avoid in its path to the most likely crewmate. As with Bot 4, we pick the pair of cells with the highest probability of containing the crew mates and path to the closest one. Furthermore, finding one crewmate collapses the map back into two dimensions using the position of the found crewmate, and reverts to single crewmate behavior.

3.8 Bot 8

This bot utilizes the same strategy as Bot 5 with the upgraded probability engine modules, but accounts for the fact that there are two aliens as well as two crewmates. As mentioned in section 2.4, we use the probability $P(A1_t = i \vee A2_t = i \mid Y_0, ..., Y_t)$ to judge the presence of aliens on a cell by cell basis, and use that in the calculation of utility in place of $Blf_t^a(i)$. The bot uses a similar calculation to judge the presence of crewmates for its utility, as does bot 4. This bot also collapses the map once a crewmate is found.

4 Performance Analysis (Q3)

To evaluate the performance of each set of bots, I set in place the testing regiment recommended by the TA. This involved generating 10 unique ship configurations. Then, for each ship configuration, generating 10 different starting positions. Then for each starting position, testing the bots 10 times. This results in a total of 1000 tests for each bot. It should be noted that bots operating in the same environment (e.g. bot 1 and bot 2) were tested using the same ship configurations and starting positions.

I chose the following values for k and α and ran the same sets of tests for each combination.

- k: 3, 7, 10
- α: 0.01, 0.05, 0.1, 0.5, 1

My choices are a little arbitrary, but from experimentation I found these numbers to create a diverse set of data.

For the sake of organization, you may view the charts at the end of this paper. You can also view them separately in the `report/charts` directory.

4.1 Bot 1 vs. Bot 2

There is a clear distinction in the results between low values of α and high values of α. So, let us consider first the lower values of α. For these values, we see the bots performing consistently amidst themselves (for various k), and similarly against each other. Bot 1 sees better results for lower values of k while Bot 2 sees better results for the higher values of k. This could be because a higher value of k can ensure that the cells within Bot 2's search depth do not contain aliens. This encourages Bot 2 to explore more and collect more meaningful data from its sensors.

When it comes to the higher values of α, however, we see a drastic drop in the performance of Bot 1, while Bot 2 maintains consistent results. As both bots can no longer rely on receiving beeps from crewmates, Bot 2 has an advantage in that it remembers where it was before and is actively encouraged to go to places it hasn't been before. Thus, Bot 2 can just look for the crewmate without needing their beeps. This is reflected in the higher number of steps on average Bot 2 takes to save the crewmate compared to its performance in lower α values. Bot 1 on the other hand, sees many cells to be equally likely to contain the crewmate, which causes it to randomly wander the ship as it traverses to these cells. It is generally understood that randomly wandering around is not an ideal strategy for search and rescue missions.

4.2 Bot 3 vs Bot 4 vs Bot 5

Here, we see Bots 4 and 5 consistently outperforming Bot 3 across all metrics (for the most part, see $k = 3$). As before, Bot 5, equipped with the ability to remember where it was before (and act on that fact), is able to endure less sensitive sensors and achieve consistent success. In fact, it appears that Bot 5's memory is advantageous in general across various configurations. Of course, with the slight exception for $k = 3$, where it feels less confident to explore in the face of greater uncertainty about the alien's position. For $k = 7$, however, we see Bot 5 require a little more steps on average to save the crew member than Bot 4. This is explained by Bot 5's local search limitation, requiring it to move around a little more to pinpoint where the crewmate is.

It should be noted that the number of steps required by Bot 3 to save all crewmates is drastically higher than both Bot 4 and Bot 5. This is most likely because Bot 3's one crewmate assumption causes it to mislead itself as to where the crewmates truly are.

As for the anomaly at $k = 3$, where Bot 3 outperforms Bots 4 and 5, it is difficult to understand why. Visualizing simulations with these settings don't reveal any glaring issues, and I'm led to believe the most probable explanation may be some error in how I organized and entered the data. I would retest, but I do not have the time unfortunately.

4.3 Bot 6 vs Bot 7 vs Bot 8

Due to time constraints, I was only able to run these tests for $k = 7$. Nonetheless, we see results consistent with before. The bot making incorrect assumptions about the environment (Bot 6) generally performs the worse, the global search bot (Bot 7) making the correct assumptions about the environment performs much better than the previous bot, and the local search bot with the ability to remember where it was previously performs the best (Bot 8). Though, it is intriguing to understand why Bot 7, while making the correct assumptions about the environment and performing the correct updates on its belief, performs worse. Investigating this further, I found that when an alien first enters Bot 7's sensor range, the bot becomes surrounded by cells that have high likelihoods to contain the alien. When coupled with the fact that the first step in the path Bot 7 computes to the crewmate must not contain the alien, the bot finds itself stuck as probabilities seep in deeper into the bot's alien sensor range. This is true for Bots 1 and 4 as well. I have observed them remaining still as they wait for the alien to leave its sensor range. Bots 2, 5 and 8 don't suffer from this paralaysis as it satisfies its desire to explore in the face of alien presence.

Coming back to the data, we see a generally lower success rate and number of crewmates saved than previous tests. As is expected when introducing an additional adversary to the simulation. Another thing of interest is how close the number of average crewmates saved is between Bot 6 and bot 7 for $\alpha = 0.1$. This difference is not shared in the corresponding probability of success for each bot. This suggests that Bot 6 is consistently capable at finding one crewmate, but cannot properly adjust its beliefs as it searches for the second one. Bot 7 can properly pivot its belief once it finds the first crewmate as it collapses its previous belief map using the position of the crewmate it found. However, the similarity in the number of crewmates found suggests that Bot 7 is not as consistent at finding the first crew mate.

5 Conclusion

5.1 Speculation (Q4)

The ideal bot should take into account *all* available information, including sensor data (alien detection and crew member beeps), probability distributions of alien and crew member locations, and the history of explored cells. That means a local search will not suffice. It should update beliefs without making assumptions about the independence of alien movement and crewmate positioning as I did. It should also utilize a more nuanced approach towards calculating utilities of cells, which may incorporate the expected amount of information that the bot may gain from entering a particular cell. This could possibly be computed by "simulating" certain sensor events in each cell, and seeing which one narrows down crewmate and alien probabilities the most (i.e. reduces variance). If the ideal bot were to use a utilitarian approach towards judging cells, this may mean a more refined choice of the utility function

constants. The bot could also consider greater time horizons, rather than a single time step, via prediction, and discount future utilities of cells by the correct amount.

I am tempted to say that the bot should use some sort of learning algorithm to figure out some of these values and strategies, like the formulation of constants. But the "ideal" bot should already know what these values are, as well as what actions to take when presented with certain information. A bot that learns is also subject to tending towards a local minima, as opposed to the global minima, which isn't necessarily ideal.

5.2 Bonus

With crewmate movement, two things must change from the model defined above. First, there now exists a dependence on time, as where the crewmate is at some time t now dictates where it can be at time $t+1$. Thus the random variable C becomes C_t. Second, we must update the transition models $P(C_{t+1} = c \mid C_t = c')$ and $P(C1_{t+1} = i,\ C2_{t+1} = j \mid C1_t = i',\ C2_t = j')$. Assuming the crewmate's movement matches that of the alien, we may base these calculations off of our previous findings. As before, let me denote $n(c)$ to be the number of adjacent open neighbors that a cell c has and $S_n(c)$ the set of open cells adjacent to c, *excluding* c itself. Then, for the one crewmate case

$$P(C_{t+1} = c \mid C_t = c') = \begin{cases} 0, & \text{if } c \notin S_n(c') \\ \frac{1}{n(c')}, & \text{if } c \in S_n(c') \end{cases}$$

For the two crewmate case

$$P(C1_{t+1} = i,\ C2_{t+1} = j \mid C1_{t+1} = i',\ C2_{t+1} = j') = \begin{cases} 0, & \text{if } i \notin S_n(i') \text{ or } j \notin S_n(j') \\ \frac{1}{n(i') \cdot n(j')} & \text{otherwise} \end{cases}$$

As a result, we no longer only consider the current belief of a cell to determine the next belief. Rather, we consider the belief of the cells adjacent to it, as those are the cells that the crewmate must be in to move into this cell. Thus, the summation returns instead of being simplified out.

6 Charts

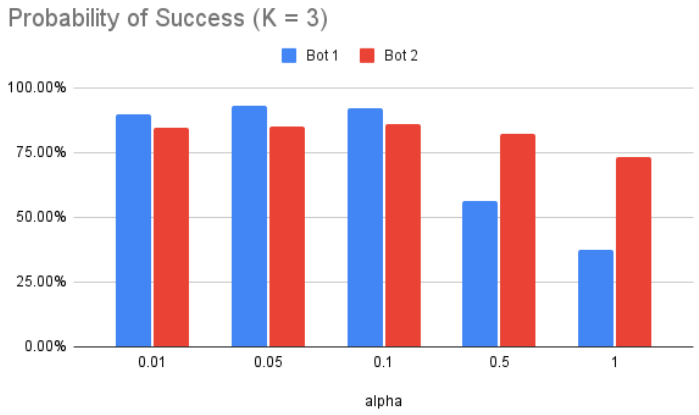

Figure 2: Bot 1 vs Bot 2 - Probability of Success and Avg. Number of Crewmates Saved for K = 3

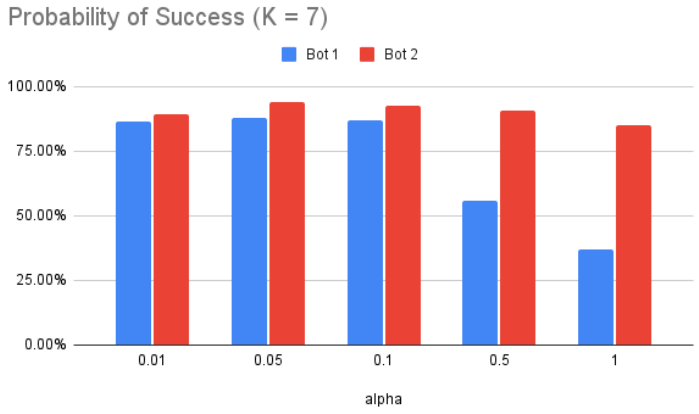

Figure 3: Bot 1 vs Bot 2 - Probability of Success and Avg. Number of Crewmates Saved for K = 7

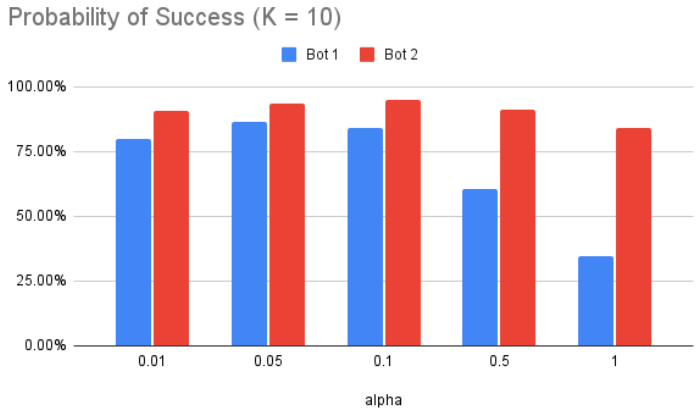

Figure 4: Bot 1 vs Bot 2 - Probability of Success and Avg. Number of Crewmates Saved for K = 10

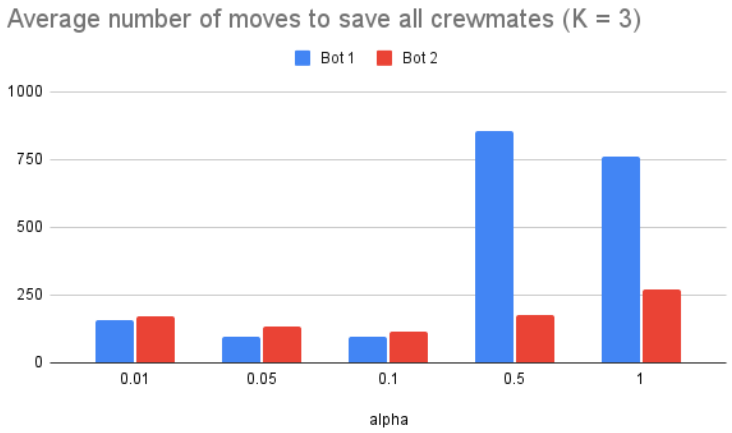

Figure 5: Bot 1 vs Bot 2 - Average number of moves to save all crewmates for K = 3

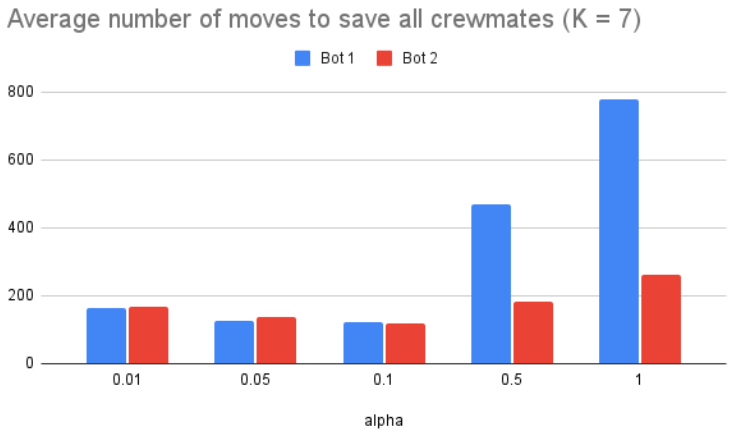

Figure 6: Bot 1 vs Bot 2 - Average number of moves to save all crewmates for K = 7

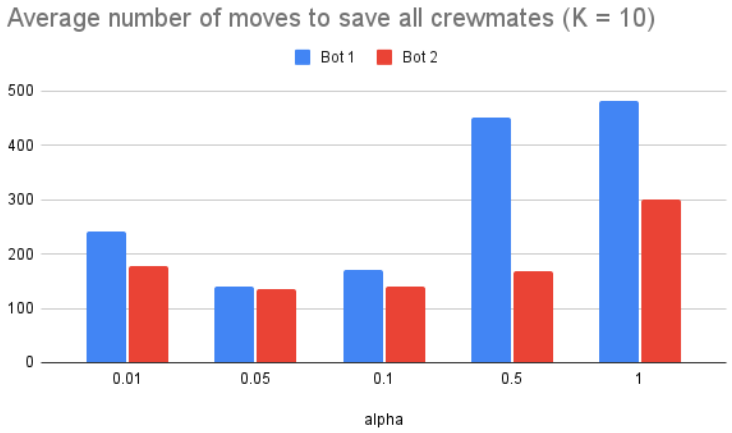

Figure 7: Bot 1 vs Bot 2 - Average number of moves to save all crewmates for K = 10

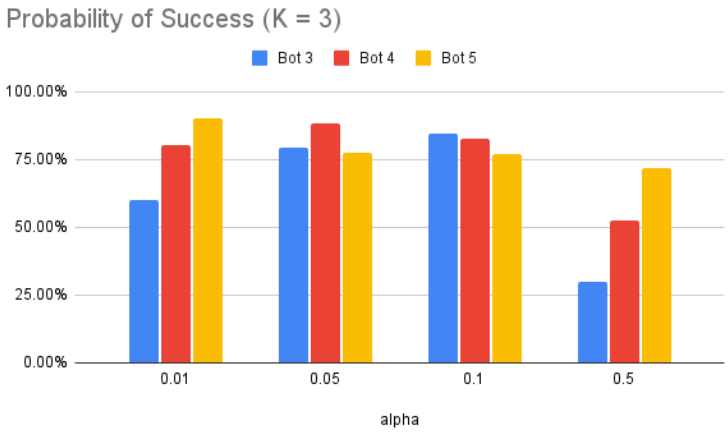

Figure 8: Bot 3 vs Bot 4 vs Bot 5 - Probability of Success for K = 3

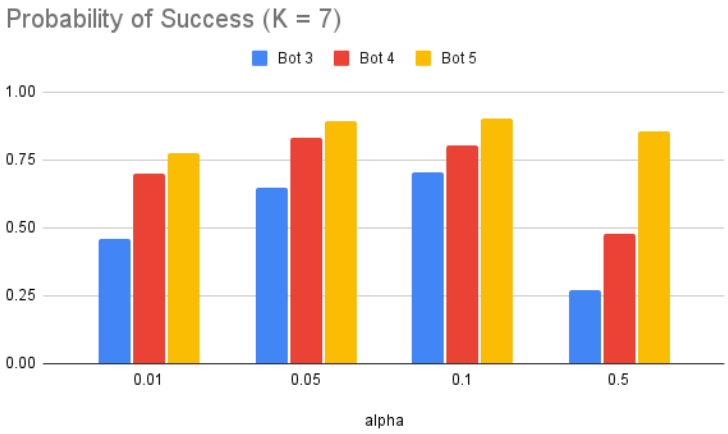

Figure 9: Bot 3 vs Bot 4 vs Bot 5 - Probability of Success for K = 7

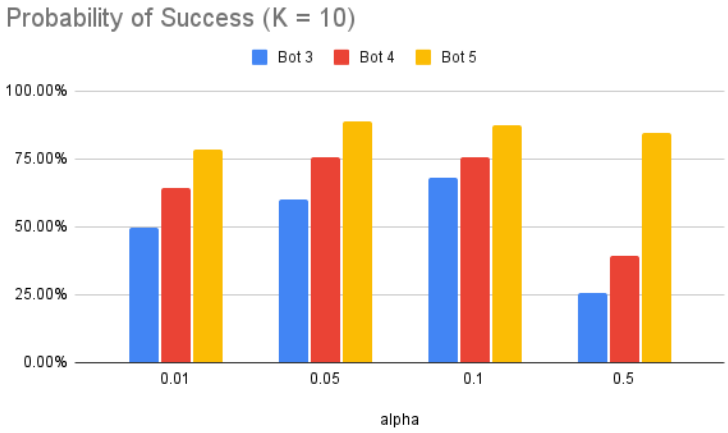

Figure 10: Bot 3 vs Bot 4 vs Bot 5 - Probability of Success for K = 10

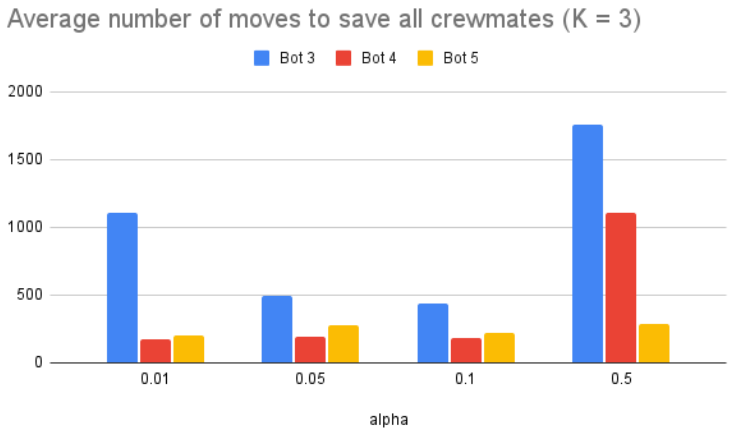

Figure 11: Bot 3 vs Bot 4 vs Bot 5 - Average number of moves to save all crewmates for K = 3

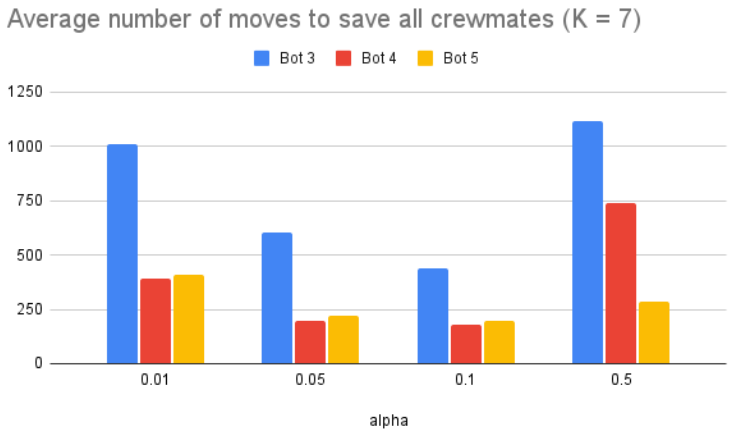

Figure 12: Bot 3 vs Bot 4 vs Bot 5 - Average number of moves to save all crewmates for K = 7

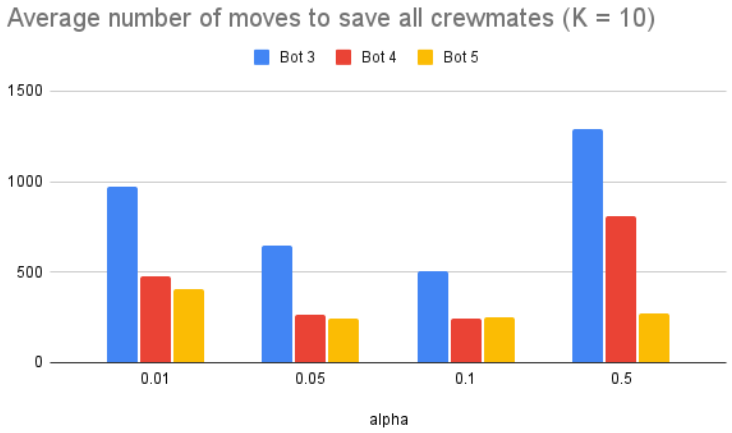

Figure 13: Bot 3 vs Bot 4 vs Bot 5 - Average number of moves to save all crewmates for K = 10

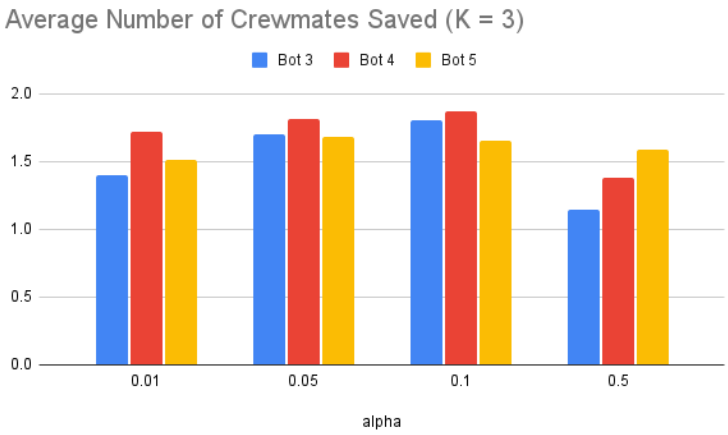

Figure 14: Bot 3 vs Bot 4 vs Bot 5 - Average Number of Crewmates Saved for K = 3

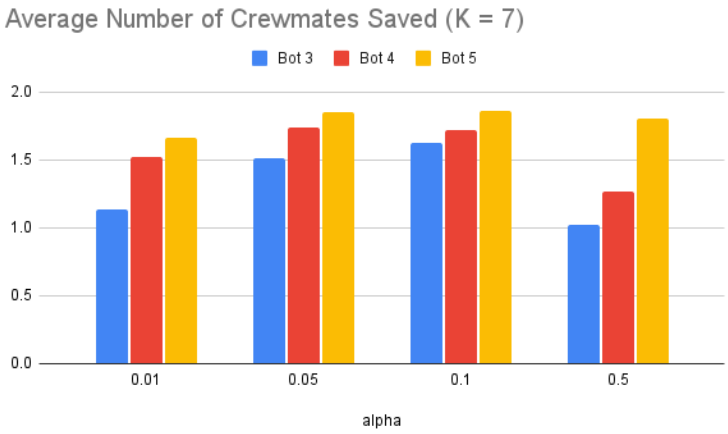

Figure 15: Bot 3 vs Bot 4 vs Bot 5 - Average Number of Crewmates Saved for K = 7

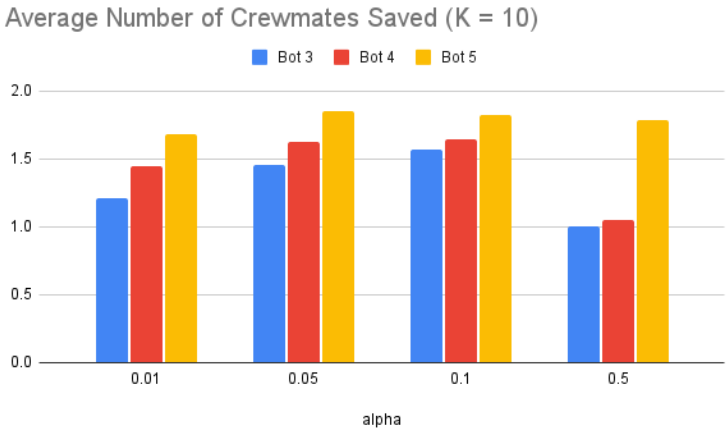

Figure 16: Bot 3 vs Bot 4 vs Bot 5 - Average Number of Crewmates Saved for K = 10

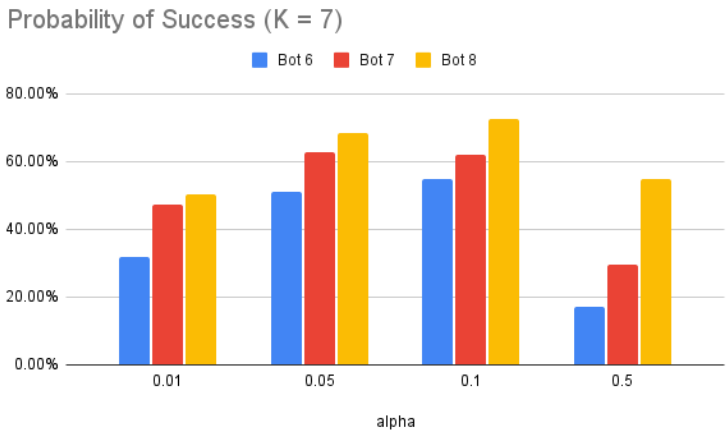

Figure 17: Bot 6 vs Bot 7 vs Bot 8 - Probability of Success for K = 7

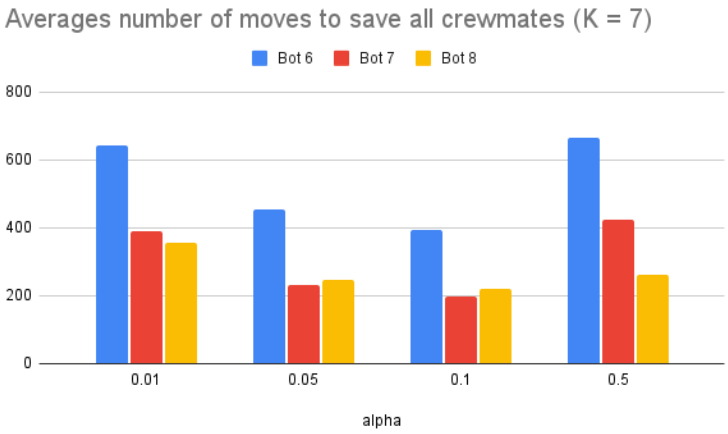

Figure 18: Bot 6 vs Bot 7 vs Bot 8 - Averages number of moves to save all crewmates for K = 7

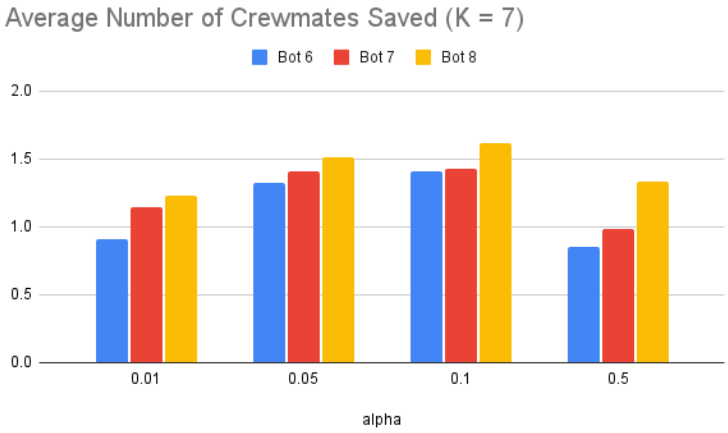

Figure 19: Bot 6 vs Bot 7 vs Bot 8 - Average Number of Crewmates Saved for K = 7