Introduction to Artificial Intelligence
198:440
Project 2

Larry Scanniello

July 2024

1 Preliminaries

Like project 1, I used NumPy arrays to model my ship. A detailed description of the workings of the ship can be found in the readme file.

Bot 1 works by setting course to the highest probability square, and only once arriving at this square does it sense and reevaluate its path. It sets its course via BFS. Bot 2 alternates sensing and moving. If at any point in its sensing, it finds that another square has a higher probability, then it changes course to that square. It also uses BFS.

All trials were run with the bots starting at same random spot, with a new randomly generated grid for each trial, and with mice placed in random squares, and a new random α within a certain range for each trial.

2 Implementation of the probabilistic knowledge base: One mouse

I will go along with the textbook and let a bold $\mathbf{P}$ denote a distribution of probabilities, and regular P denote a probability.

I used temporal modeling to model the ship as a Markov chain. This is valid because the location of the mice at each new time step in the ship depend only on the time step immediately before it. Each time step on the ship grid represents a new state X_t, where $P(X_t = x_t)$ is the probability that there is a mouse at location x_t at time t. For each time step t, the bots each gain new evidence e_t (each bot respectively gets their own evidence per time step).

I have a function in my code called **filtering**, which is a straightforward filtering function that calculates $\mathbf{P}(X_t|e_1, ..., e_t)$. In class/the textbook, it is shown that this distribution can be calculated recursively, via

$$\mathbf{P}(X_{t+1}|e_1, ..., e_{t+1}) = \alpha' \mathbf{P}(e_{t+1}|X_{t+1}) \sum_{x_t} \mathbf{P}(X_{t+1}|x_t) P(x_t|e_1, ..., e_t)$$

where α' normalizes the distribution. I stored probabilities by making each probability distribution a 40×40 array to match the main ship grid, and by storing probabilities at the index they are storing information for on the main grid. I will explain each of these terms and how I calculated them.

The calculation of $\sum_{x_t} \mathbf{P}(X_{t+1}|x_t) P(x_t|e_1, ..., e_t)$ can be thought of as the prediction stage of filtering, where we are essentially taking a weighted average of all of the possibilities for where the mouse can go. The weights, the numbers $P(x_t|e_1, ..., e_t)$, are just the numbers $Belief_t(x_t)$, which comes from the distribution $\mathbf{P}(x_t|e_1, ..., e_t)$, which is the result of the previous recursion step. We note here that $\mathbf{P}(X_0)$,

the base case, is the uniform distribution defined by $P(X_0 = x_0) = 1/(\text{num of open spaces at start})$ for all open spaces x_0, since at the start, the mouse is equally likely to be in any open space.

The distribution $\mathbf{P}(X_{t+1}|x_t)$ is the transition model, and in the case of our simulation, is determined by whether the mouse is stationary or stochastic. If the mouse is stationary, and x_t is an arbitrary space, then we can define $\mathbf{P}(X_{t+1}|x_t)$ by

$$P(X_{t+1} = y_{t+1}|x_t) = \begin{cases} 1 & \text{if } y_{t+1} = x_t \\ 0 & \text{otherwise} \end{cases}$$

since if a mouse is at space x_t at time t, then we can be sure it is in the same space at time $t+1$, and we can be sure it is not in any other space at time $t+1$. But for a space $z_t = z_{t+1}$, this implies that

$$\begin{aligned} \sum_{x_t} P(z_{t+1}|x_t)P(x_t|e_1,..,e_t) &= P(z_{t+1}|z_t)P(z_t|e_1,...,e_t) + \sum_{x_t \text{ not } z_t} P(z_{t+1}|x_t)P(x_t|e_1,...,e_t) \\ &= (1)P(z_t|e_1,...,e_t) + \sum_{x_t \text{ not } z_t} (0)P(x_t|e_1,...,e_t) \\ &= P(z_t|e_1,...,e_t) \end{aligned}$$

which tells us that in the stationary case, $\sum_{x_t} \mathbf{P}(X_{t+1}|x_t)P(x_t|e_1,...,e_t) = \mathbf{P}(X_t|e_1,...,e_t)$, which means that intuitively, our prediction of the location of a mouse that isn't moving is that it stays in place.

If the mouse is stochastic, then this requires more computations. Let x_t be a square on the board. We need to calculate $\mathbf{P}(X_{t+1}|x_t)$. If the mouse is at x_t, then the mouse is equally likely to make a move that is either move to any square that is adjacent to it or stay in place. Thus the nonzero probabilities in $\mathbf{P}(X_{t+1}|x_t)$ will be uniform, and more precisely, if N is the number of adjacent squares to x_t,

$$P(X_{t+1} = y_{t+1}|x_t) = \begin{cases} 1/(N+1) & \text{if } y_{t+1} \text{ is adjacent or equal to } x_t \\ 0 & \text{otherwise} \end{cases}.$$

For example, assume we have a 2×2 grid of all open squares, and let

$$\mathbf{P}(X_t|e_1,...,e_t) = \begin{pmatrix} .1 & .2 \\ .7 & 0 \end{pmatrix}$$

Then for the stochastic case, $\sum_{x_t} \mathbf{P}(X_{t+1}|x_t)P(x_t|e_1,...e_t)$ is

$$\begin{pmatrix} 1/3 & 1/3 \\ 1/3 & 0 \end{pmatrix}(.1) + \begin{pmatrix} 1/3 & 1/3 \\ 0 & 1/3 \end{pmatrix}(.2) + \begin{pmatrix} 1/3 & 0 \\ 1/3 & 1/3 \end{pmatrix}(.7) = \begin{pmatrix} .333 & .1 \\ .267 & .3 \end{pmatrix}$$

There are two things to note in my example. One, the space with the highest probability of a mouse at time t is not the most likely to have a mouse at time $t+1$. Second, the probabilities become less spread, since at each prediction step the predicted state gets closer to the stationary distribution.

Now that we have examined the prediction stage of filtering, we need to look at the stage of taking into account new evidence. If the bot sensor gets a beep at the most recent time step, then it is $\mathbf{P}(e_{t+1}|X_{t+1})$ which will encode this information.

If e_{t+1} is a positive sense when the bot was located at z_{t+1}, and x_{t+1} is a space that is d (Manhattan distance) away from z_{t+1}, then $P(e_{t+1} = +|x_{t+1})$, the probability of a positive beep at z_{t+1} given the mouse is at x_{t+1}, is defined to be $e^{-\alpha(d-1)}$. If we do not receive a sense, then instead we have $P(e_{t+1} =$

$-|x_{t+1}) = 1 - e^{-\alpha(d-1)}$. For example, if we have a 3×3 grid of all open squares, and we receive a positive sense at $(0, 0)$, then

$$\mathbf{P}(e_{t+1} = +|X_{t+1}) = \begin{pmatrix} 0 & 1 & e^{-\alpha} \\ 1 & e^{-\alpha} & e^{-2\alpha} \\ e^{-\alpha} & e^{-2\alpha} & e^{-3\alpha} \end{pmatrix}$$

Alternatively, if we do not get a sense, then

$$\mathbf{P}(e_{t+1} = -|X_{t+1}) = \begin{pmatrix} 0 & 0 & 1 - e^{-\alpha} \\ 0 & 1 - e^{-\alpha} & 1 - e^{-2\alpha} \\ 1 - e^{-\alpha} & 1 - e^{-2\alpha} & 1 - e^{-3\alpha} \end{pmatrix}$$

On the other hand, if we get a piece of evidence of the form $\neg(i, j)_t$, which means the bot stepped into the space (i, j) at time t and there was no mouse there, then we can define

$$P(e_{t+1} = \neg(i, j)_t|x_{t+1}) = \begin{cases} 1 & \text{if } x_{t+1} \text{ is not the space stepped into} \\ 0 & \text{if } x_{t+1} \text{ is the space stepped into} \end{cases}$$

This form of evidence zeros out the space that was stepped into and normalizes the other probabilities accordingly.

In my code, I represented these distributions as arrays, which are the same size as the main ship's array, with probabilities in a space on an array corresponding to the main grid. Storing these distributions as NumPy arrays has provided ease in computations, with NumPy performing fast pointwise operations. Updating the knowledge base via filtering is performed for each bot at each time step with each new piece of evidence, using the formula

$$\mathbf{P}(X_{t+1}|e_1, ..., e_{t+1}) = \alpha' \mathbf{P}(e_{t+1}|X_{t+1}) \sum_{x_t} \mathbf{P}(X_{t+1}|x_t) P(x_t|e_1, ..., e_t)$$

where the multiplication of distributions is defined by pointwise array multiplication.

3 Bot 3

Bots 1 and 2 do not take care in what path they choose to get to the destination. This is not efficient. Suppose a bot needs to move 10 squares up and 10 squares right, and for simplicity assume there are no closed spaces on the grid. There are a total of $\frac{20!}{10!10!} = 184756$ possible shortest paths that this journey can take. Why take a random path given by BFS when we can choose a path more deliberately? My bot 3 used UFCS and prediction to make better decisions on which path to take. Like the other bots, I had bot 3 set out to the square with the highest probability of having a mouse in it. Once the destination was set, I initiated a search.

The first step of the search was to create a graph. The first vertex of the graph is the current bot index. To get the next level of the graph, I ran my probability distribution through **predicting**, my prediction algorithm. This is just filtering without the new evidence step. (If the mouse is stationary, then this prediction function just returns the same distribution). The weights of the vertices that the initial node points come from this predicted distribution. The weights of the vertices that these vertices point to are on the next predicted distribution, and so on until the goal node is reached. In this way, my bot accounts for the predicted probability of there being a mouse in the space *at the time the bot will be there.*

For stochastic mice, this prediction strategy is most effective when the bot is nearest to the destination, because if the bot needs to take many steps to get to the destination, the predicted grids many time steps from the bot's present will approach the stationary distribution. However, this approach will still be better

than just taking whatever path BFS gives, because at the very least the squares nearest to the bot can be chosen effectively.

If $x_1, ..., x_n$ are the potential squares the bot can step in, the goal is to find find $\underset{x_1,...,x_n}{\text{argmax}}\ P(x_1 \vee ... \vee x_n)$. But with some mathematical manipulations we can restate this:

$$\begin{aligned}
\underset{x_1,...,x_n}{\text{argmax}}\ P(x_1 \vee ... \vee x_n) &= \underset{x_1,...,x_n}{\text{argmax}}\ 1 - P(\neg x_1 \wedge ... \wedge \neg x_n) && (1)\\
&= \underset{x_1,...,x_n}{\text{argmin}}\ P(\neg x_1 \wedge ... \wedge \neg x_n) && (2)\\
&= \underset{x_1,...,x_n}{\text{argmin}}\ P(\neg x_1)P(\neg x_2)...P(\neg x_n) && (3)\\
&= \underset{x_1,...,x_n}{\text{argmin}}\ (1 - P(x_1))(1 - P(x_2))...(1 - P(x_n)) && (4)\\
&= \underset{x_1,...,x_n}{\text{argmin}}\ \ln(1 - P(x_1)) + \ln(1 - P(x_2)) + ... + \ln(1 - P(x_n)) && (5)
\end{aligned}$$

For step 3, we have assumed the independence of the mice. This is an approximation, because in my simulations, I did not allow the possibility of two mice stepping into the same space, but since it is very rare for the mice to step into each other's spaces on a 40×40 board, this is a justified approximation. To make this work easily with a UFCS algorithm, I added 1 to each weight to make the weights positive.

In addition to the UFCS/prediction, my bot alternates sensing and moving, and reevaluates its goal every 5 steps. If the mouse is stochastic, and if my bot's goal is within four squares (Manhattan distance) away, my bot does not alternate, and only moves.

4 Results: One mouse

It was not obvious at first how to come up with values for α. I ultimately decided to let α take on the following values:

$$\left\{ \frac{-\ln 0.5}{k-1} : k = 4, 5, 6, ..., (\text{upper bound}) \right\}$$

where I will refer to k as a new index. The reason I chose this set of values for α is that if the mouse is k away from the bot, the probability of a beep is 0.5. In other words, the value of k determines the distance at which a beep is a coin flip. I did most of my testing between $k = 4$ and $k = 100$, which corresponds to $\alpha = .231$ and $\alpha = .00700$. I found that if α was any higher (k any lower) then bot 2 would fail, getting stuck in an indecisive alternation between two points.

The summary statistics of my data can be seen in figures 1 and 2. The numbers observed are the number of time steps taken until a bot's completion. I ran hypothesis tests at 90% confidence where the null

	Bot 1	Bot 2	Bot 3
Mean	401.65	309.024	275.55
Median	337.5	270	238
Max	1871	5000	1030
Std	278.7653918	234.6616319	160.8623704
Z-score	12.38975681	3.72064765	
Reject null	Yes	Yes	

Figure 1: Time steps to completion for one stationary mouse, k random from 4 to 100, $n = 1000$

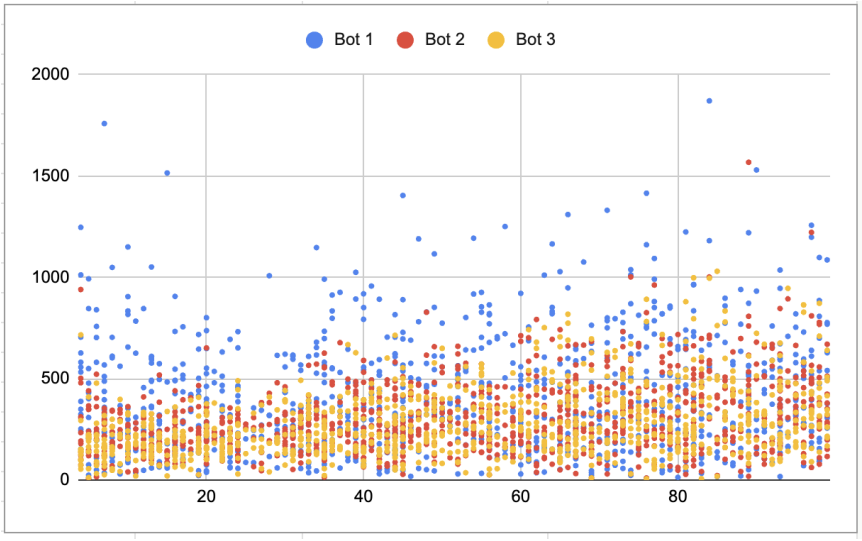

Figure 2: One stationary mouse, time steps to completion as a function of k, k random from 4 to 100, $n = 1000$

	Bot 1	Bot 2	Bot 3
Mean	550.667	488.503	430.523
Median	415.5	373	321.5
Max	3895	4688	3141
Std	486.9902603	422.293227	374.319268
Z-score	6.185479933	3.249076837	
Reject null	Yes	Yes	

Figure 3: One stochastic mouse, time steps to completion as a function of k, k random from 4 to 100, $n = 1000$

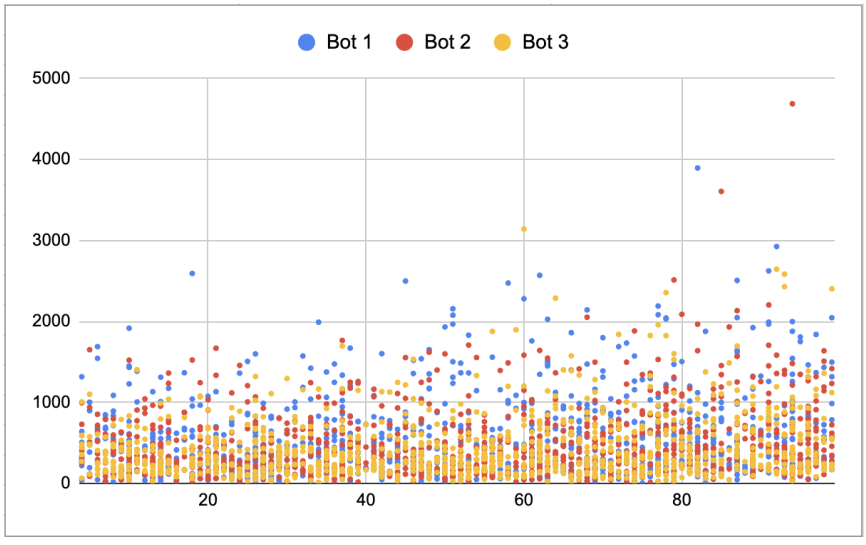

Figure 4: One stochastic mouse, time steps to completion as a function of k, k random from 4 to 100, $n = 1000$

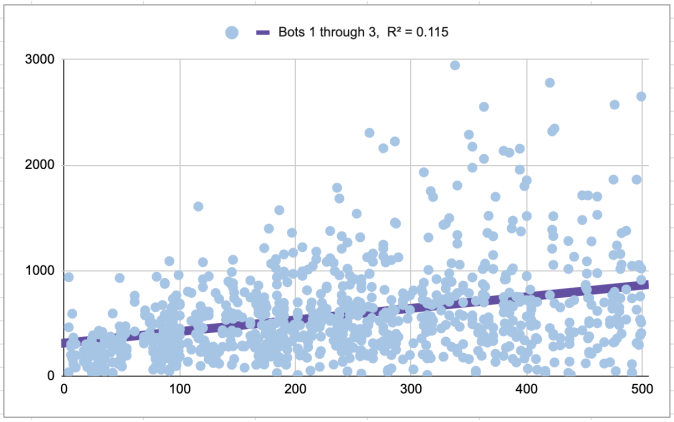

Figure 5: All bots, k random from 4 to 500, stationary mouse, $n = 300$

hypotheses were that $\mu_1 \leq \mu_3$ and $\mu_2 \leq \mu_3$ (with μ_i being the true mean time steps taken of bot i), and the alternate hypothesis were that $\mu_1 > \mu_3$ and $\mu_2 > \mu_3$. The z-statistic reported for bot i is

$$\frac{\bar{x}_i - \bar{x}_3}{\sqrt{s_i^2/n + s_3^2/n}}$$

where $\bar{x}_i$ is the reported mean for bot i, s_i is the reported standard deviation, and n is the number of trials.

For the one mouse case, for both stationary and stochastic mice, my data is strong evidence that bot 3 is better than bot 1 and bot 2 on average, and we can strongly reject both null hypotheses.

I think my bot 3 is not as wasteful as the other two bots. Rather than taking any path to the destination, it considers many paths to the destination and chooses the best one. Like bot 2, it moves slowly and takes in a lot of information via frequent sensing. But it is not as indecisive as bot 2. Nor is it as hasty as bot 1. In the stochastic case, I think the fact that it can move quickly if it is near a mouse makes a difference, when being too slow can mean the mouse gets away.

The evidence for a relationship between α and steps to completion is not as strong as I thought it would be. In figures 2 and 3, visually, there does not seem to be a very strong relationship between k and steps

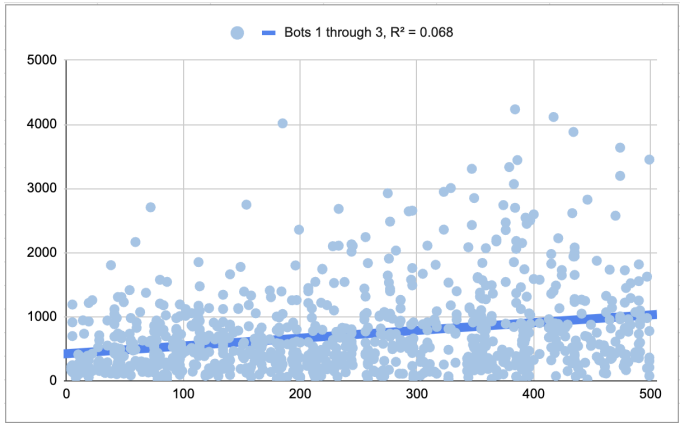

Figure 6: All bots, k random from 4 to 500, stochastic mouse, $n = 300$

to completion for the bots. I will say here that I did not test on any values for α higher (k lower) than when $k = 4$. At these values of α, bot 2 would completely fail, getting stuck forever traversing one area of the grid, not decisively choosing a destination to go to. So I do not have data from trials that the bot fails at this level of α, but I didn't think the data would be necessary.

I also did tests where k was chosen randomly from $\{4, 5, ..., 500\}$. For reference, at $k = 500$, $\alpha = .0014$, and the probability of getting a beep when 10 away from a mouse is .98. When we let k be this large, we see in figures 5 and 6 (where a point on the graph can be any of bots $1, 2$, or 3) we see trendlines indicate both a stationary and stochastic mouse that there is a positive relationship between k and time steps for a bot until completion. Furthermore, the slope of these trendlines are approximately 1, which means on average, if we increase k by 1, we should expect one more trial until completion. However, the R^2 of these regressions are both low, which means that changes in k account for little of the variance at play. However, that does not mean that increasing k/decreasing α isn't doing anything. It may be a reflection that in this project in general, there is a *lot* of variance at play, and given the fuzzy picture of the current state that the bots get from their evidence, any bot strategy will have varied success.

5 Two mice: Stationary

In the one mouse case, we were interested in $\mathbf{P}(X_t|e_1, ..., e_t)$. The variable X_t had on the order of 1000 possible values it could take on, one corresponding to each open space on the grid. In this new context, we will redefine X_t to be a random variable that takes on a value for every *pair* of points on the grid. In other words, X_t may take on the value $(0, 0) \wedge (15, 16)$, or $(39, 34) \wedge (14, 13)$. One simplifying assumption for the purpose of computation is that we will store probabilities for pairs like $(2, 9) \wedge (2, 9)$ or $(12, 13) \wedge (12, 13)$, even though the mice can't be in the same space. Also, we will not distinguish the mice, instead letting these probabilities be probabilities that *there is a mouse in both spaces*, so $(2, 9) \wedge (12, 13)$ and $(12, 13) \wedge (2, 9)$ are the same value.

Working with a random variable that takes on on the order of millions of values was very computationally expensive.

In terms of filtering, much remains the same. In the new evidence step, we need to alter our probabilities. A sense is positive if it detects either mouse. Thus if x_t is d_1 away from the bot, and y_t is d_2 away,

$$\begin{aligned} P(e_t = +|x_t \wedge y_t) &:= P(\text{beep from } x_t \text{ or beep from } y_t) \\ &= 1 - P(\neg\text{beep from } x_t \wedge \neg\text{beep from } y_t) \\ &\quad \text{By the independence of the beeps,} \\ &= 1 - P(e_t = -|x_t)P(e_t = -|y_t) \\ &= 1 - (1 - e^{-\alpha(d_1 - 1)})(1 - e^{-\alpha(d_2 - 1)}) \end{aligned}$$

	Bot 1 mouse 1	Bot.1 mouse 2	Bot 2 mouse 1	Bot 2 mouse 2	Bot 3 mouse 1	Bot 3 mouse 2
Mean	455.24	854.67	519.18	800.72	465.16	719.22
Median	350	761.5	479.5	763.5	354	627
Max	1360	5000	1741	2180	1856	1972
Std	354.9182043	555.3062865	368.9576467	397.6496626	399.7734894	434.7256787
Z-score	-0.003471131485	1.920645879	0.9929935872	1.3833211		
Reject null	No	Yes	No	No		

Figure 7: Time steps to completion, two stationary mice, k up to 100, $n = 100$

	Bot 1	Bot 2	Bot 3
Mean	1309.91	1319.9	1184.38
Median	1177.5	1209	1017
Max	5503	3764	3800
Std	797.4254671	750.403548	825.7908701
Z-score	1.093502604	1.2145417	
Reject null	No	No	

Figure 8: Total damage, two stationary mice, k up to 100, $n = 100$

This defines $\mathbf{P}(e_t = +|X_t)$, and $\mathbf{P}(e_t = -|X_t)$ is defined similarly (with 1 minus the same quantity instead), and $\mathbf{P}(e_t = \neg(i,j)_t)|X_t)$ is the same as before, except it eliminates all pairs with (i,j). The prediction step is the same as before, but instead of looking at every possibility of where the mouse is and where it can go, we now need to look at every possible pair of spaces where the mouse in, and look at all possible pair of spaces the mice can be in at the next time step. This makes it computationally infeasible to run many simulations, since this means that each filtering step, we need to perform tens of millions of operations. So I only tried this approach of storing probabilities for pairs for stationary mice. Although the stationary mice simulations were slow to run, the case of stochastic mice with the full joint distribution were infeasible to run.

Once I had the distribution of all pairs, I calculated the distribution of probabilities that there is a single mouse at each point. This was done by making a pass through the pairs, and if D is the set of open spaces on the main grid, we find for each space x_t the sum

$$P(x_t|e_1,..,e_t) = \sum_{y_t \in D} P(x_t \wedge y_t|e_1,...,e_t)$$

via marginalization. Then the bots make decisions the same as before, going to the destination with the highest probability given by this distribution.

The only other detail we need to look at is what happens when a mouse is caught. Given the set of all probabilities of the form $P(x_t \wedge y_t|e_1,...,e_t)$, and given that we found a mouse at y_t, for each space we are interested in $P(x_t|y_t,e_1,...,e_t)$. But since we can assume there being mice in x_t and y_t are independent, this is just the probability $P(x_t|e_1,...,e_t)$, the value of which we can find by marginalization.

I ran 100 tests with two stationary mice, k up to 100. Our null hypotheses in figure 7 were

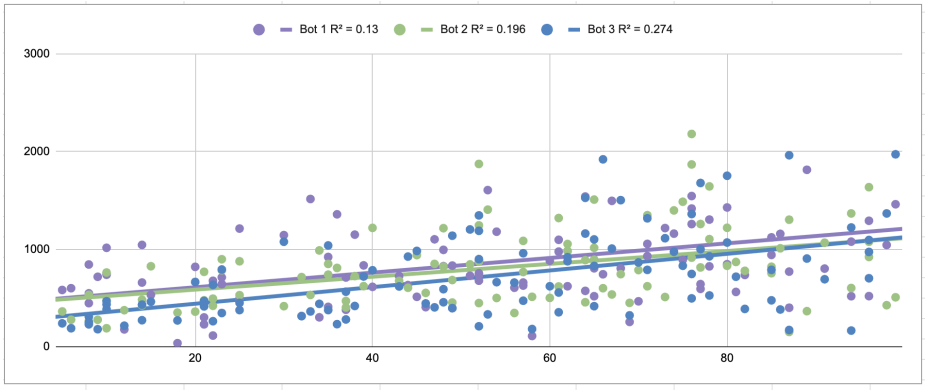

Figure 9: Two stationary mice, time steps to find both mice as a function of k, $n = 100$

bot i average time to catch mouse $j \leq$ bot 3 average time to catch mouse j
denoted by $\mu_{i,j} \leq \mu_{3,j}$

Of these, the only that our data allows us to reject is that $\mu_{1,2} \leq \mu_{3,2}$, and we can accept the alternate hypothesis that $\mu_{1,2} > \mu_{3,2}$. Given the computation time of completing these simulations (about 24 hours for 100 simulations), it is much more difficult to get statistically significant data than for the one mouse case, when I could run 1000 simulations in a night. Although we cannot reject the null hypotheses for difference of means outright, the data still represents evidence that points towards that bot 3 catches the second mouse faster than the other bots. To reject the nulls relating to mouse 2 times, if we devote more time and computational resources towards simulations, we will probably see that the overall difference in means is what we want (although there is not such a big difference in the time it takes for the bots to catch mouse 1).

I am not completely sure why bot 1 has a lower mean mouse 1 time than bots 2 and 3. We had strong evidence that it performs worse than the other bots on average in catching one mouse. I think bot 1 is highly mobile. If it doesn't get a sense at the start, it will make a journey all of the way to the other side of the ship without sensing at all. I think if there are two mice, it is less likely that a journey like this will be wasteful, and will bring it far from a mouse, and its mobility can help it in catching the first mouse. However, once it catches the first mouse, it doesn't have as thorough of information for catching the second mouse as the other bots do when they catch their first mouse, due to not sensing, which may explain the higher mouse 2 mean time. Also, due to the high variance at play in all of these results, we cannot rule out that bot 1 actually does have a higher true mean, and that our result was not due to random chance, especially since its observed mean was pretty close to bot 3's observed mean.

In figure 8, I have displayed summary data for what I will argue is the most important statistic in determining the quality of the bots: Total mouse destruction. This statistic is calculated by

$$2\cdot(\text{total time with two mice})+(\text{total time with one mouse})$$

We can see from the data that in our sample, bot 3 had both lower average and median mouse destruction. This has to be qualified with the fact that there is considerable variance here: Based on the standard deviations, there are probably many trials where bot 1 or bot 2's total destruction is lower than bot 3's. We also cannot reject the null hypotheses in this case (where the null hypotheses are difference in means with bot 3), although the z-scores are not incredibly far off from rejecting the null. This data is evidence that the mean total destruction for bot 3 is lower than the other bots on average, and if we had more time to denote to running tests, we could probably say this with more confidence. If we were sure that the results of the data were not due to random chance, then based on this statistic, bot 3 would have good claim to being the best choice of bot.

	Bot 1 mouse 1	Bot 1 mouse 2	Bot 2 mouse 1	Bot 2 mouse 2	Bot 3 mouse 1	Bot 3 mouse 2
Mean	431.5740741	939.5740741	425.2336449	926.8611111	445.5233645	795.7037037
Median	286	790	336	786	330	654
Max	2359	2500	1988	2500	1803	2500
Std	413.3116586	548.7415714	371.7950485	576.6397579	383.4759186	518.6232901
Z-score		1.980215735		1.757489877		
Reject null	No	Yes	No	Yes		

Figure 10: Time steps to completion, particle filtering with 100000 samples, k from 4 to 100, two stochastic mice, $n = 108$

	Bot 1	Bot 2	Bot 3
Mean	1371.148148	1371.305556	1260.25
Median	1164	1123	970.5
Max	4771	5000	5000
Std	884.0437023	918.2438588	912.8001196
Z-score	0.9069539353	0.8913873567	
Reject null	No	No	

Figure 11: Total damage by mice, particle filtering with 100000 samples, k from 4 to 100, two stochastic mice, $n = 108$

6 Two mice: Stochastic

To get this to work, I searched through our textbook to find a solution, and I landed on something we never covered in class but seemed like a fun experiment to try: Particle filtering.

Let me explain how I implemented particle filtering. First, I sampled 100000 pairs from all possible open pairs on the grid, where each pair was equally likely to be sampled. I stored them in a list, which looked like

$$[(24, 25, 34, 35), (0, 2, 1, 1), (26, 21, 30, 33), (11, 9, 22, 8), ...].$$

(The trick in making these tuples was to make each tuple such that the tuple formed by the first two indices was always less than or equal to the tuple formed by the third and fourth indices.) Once there is a piece of evidence, the next phase, analogous to normal filtering, is to take into account this new evidence. Each sample $x_t \wedge y_t$ in the list above is given a weight $P(e_t|x_t \wedge y_t)$. Then from this list of samples, we resample, where the probability of picking a sample is proportional to its weight. I used NumPy's built-in sampler for this, which sped up computations a lot. After resampling, we have an approximation of $\mathbf{P}(X_t|e_1, ..., e_t)$, where $P(X_t = x_t|e_1, ..., e_t) \approx \frac{\text{Frequency of } x_t \text{ is in the sample}}{N}$, where $N = 100000$. The textbook shows that the approximations approach the true probability as $N \to \infty$. We then repeat this process indefinitely, but instead of initially sampling from a uniform distribution of open spaces, we replace each tuple in the list with a random tuple from a list of possible adjacent pairs. (This is also how bot 3's prediction works.)

In terms of computational efficiency, this algorithm was an improvement over my previous one. I ran it with 100000 samples. I could have ran it with fewer, and the speed would have increased significantly, but I wanted the fidelity that many samples brings. Although it wasn't fast, I could knock out 100 simulations in a 24-hour period, which is not amazing, but was not possible with the previous algorithm. In terms of performance, it was not perfect. I capped out each simulation at 2500 total time steps, which may make certain means artificially lower, but I found that the bots hit this max maximum pretty uniformly.

Some of the data obtained is pretty surprising. I am not sure why I would be getting lower means and medians for stochastic mice than for stationary mice, when in the one-mouse case, stochastic pretty soundly took longer than stochastic. We don't have enough trials to rule out random chance, and considering the variance, random chance is completely plausible at explaining the data. Once again, bot 1 does a really good job of finding the first mouse - probably for the same reasons stated for the stationary case. Bot 3 wins again in catching the second mouse.

In figure 11, I have the summary data again for total mice destruction. While once again the computation time has prohibited us from getting conclusive evidence, we do have evidence nevertheless here which shows that bot 3 is the most effective in reducing the overall damage done by the mice, whether you look at mean or median, and hence given the evidence, is the best bot out of the three.

There is definitely a way to get this to work with a distribution with $2n$ values as opposed to n^2, but it was fun to try out particle filtering and use it successfully to get the bots to catch mice.

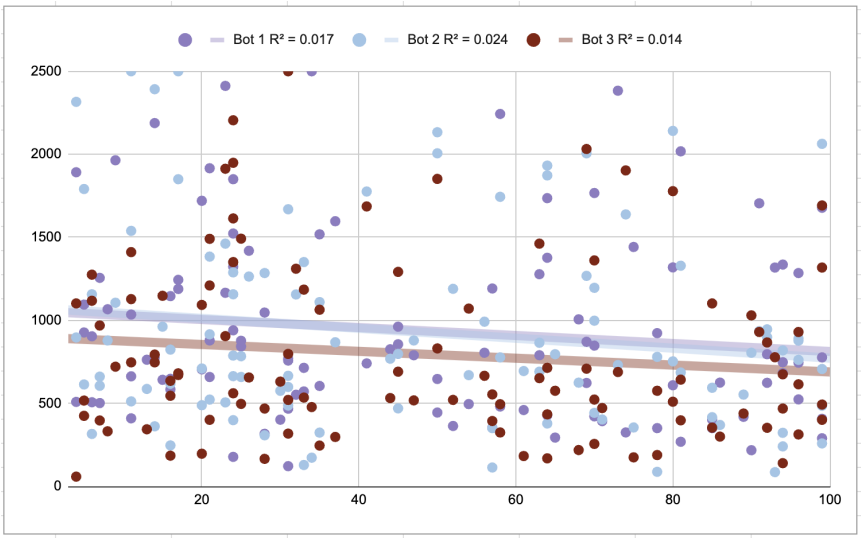

Figure 12: Trials to completion as a function of k, with k random from 4 to 100, two stochastic mice, particle filtering, $n = 108$

7 Question not yet answered

2) Suppose the ship had four rooms, A, B, C, D, and the probability the mouse was in each of them was $0.4, 0.3, 0.2, 0.1$. If you looked in room B, and the mouse was not there, what is the probability it is in room A?

We are looking for $P(A|\neg B)$. Using the definition of conditional probability,

$$P(A|\neg B) = \frac{P(A \wedge \neg B)}{P(\neg B)}$$

Since $A \iff A \wedge \neg B$, we have $P(A \wedge \neg B) = P(A) = 0.4$. We also have $P(\neg B) = P(A \vee C \vee D)$. Since A, C, and D are mutually exclusive, $P(\neg B) = P(A) + P(C) + P(D) = 0.7$. This gives us our final answer,

$$P(A|\neg B) = \frac{4}{7} \approx .5714.$$

To see how an update like this would work using filtering, assume

$$\mathbf{P}(X_t|e_1, .., e_t) = \begin{pmatrix} .4 & .3 \\ .2 & .1 \end{pmatrix}$$

Assuming the mouse doesn't move, so our predicted distribution is the same, we need to factor in $\mathbf{P}(\neg B_{t+1}|X_{t+1})$, which equals

$$\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$$

Thus

$$\begin{aligned} \mathbf{P}(X_{t+1}|e_1, ..., e_{t+1}) &= \alpha' \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} .4 & .3 \\ .2 & .1 \end{pmatrix} \\ &= \alpha' \begin{pmatrix} .4 & 0 \\ .2 & .1 \end{pmatrix} \end{aligned}$$

$$= \begin{pmatrix} 4/7 & 0 \\ 2/7 & 1/7 \end{pmatrix}$$

which agrees with our initial result, because it is the same calculation.