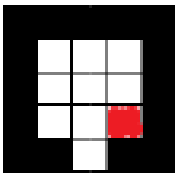
Question 1 – Localization

a) You are somewhere in the above maze, with no prior knowledge. What is the probability you are at G?

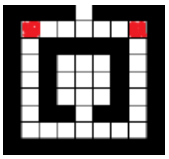
There are 694 white cells in the maze which means that, when we are in the maze, we can be in one of these 694 cells. And probability of being in each one of them are same without prior knowledge. So the probability that we are at G is 1/694.

b) You can try to move UP, DOWN, LEFT, RIGHT. If you are not blocked in that direction, you successfully move. If you are blocked, you stay in the same place. You receive no feedback. Give a (short) sequence of moves that will with probability at least 1/2 end with you at G.

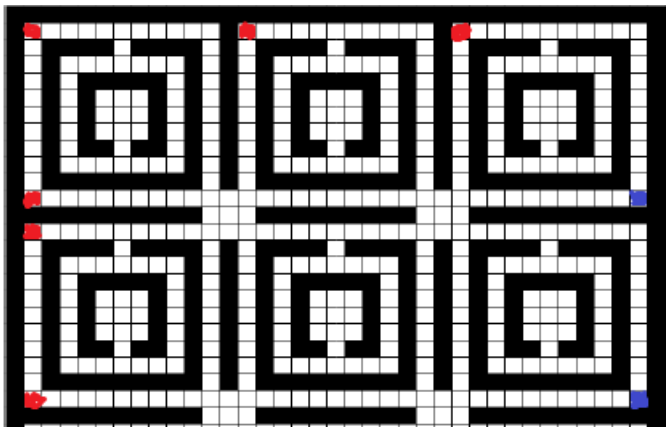
To do this, we need to operate some extra move to make ourselves blocked in that direction and make more starting cells gather in same cell after several move. Original maze can be separate to similar 9 parts. Try to find a sequence to gather all starting cells in the upper 6 parts. We can think from small part to large part.



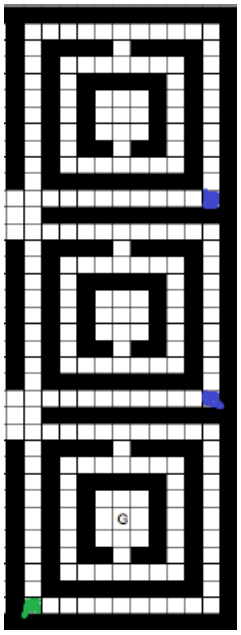
Step 1: Start from this smallest part, if we operate 2 right and 2 down, we will end up at the cell at the lower right corner (red cell above). Then we operate 1 left and 2 down, we will end up at the exit cell in the next larger part.



Step 2: Now we think about this larger part, all starting cells in the smaller part end at the exit cell in the outer circle. Then we operate 6 left and 6 up, all starting cells in this part gather at the upper left and upper right cells (red cells above). Then we operate 3 left, 1 up, 3 right and 2 up. All starting cells in this part will gather at the exit cell in the next larger part.



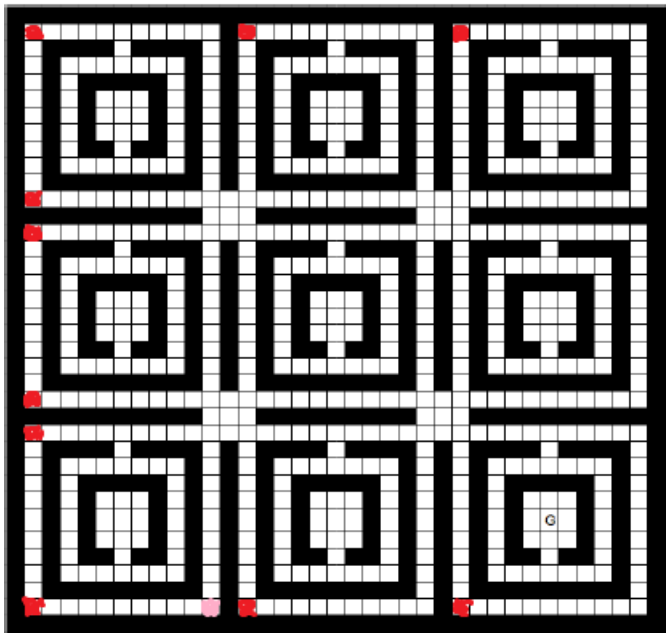
Step 3: Then we think about the largest part, now we need to gather cells in the outer circle of last part. Then we operate 22 up and 34 left, all starting cells in this part will gather at several cells (red cells above). Next, we operate 10 down and 34 right, all cells in this part will end up at two cells (blue cells above). Finally, find a sequence to gather these two cells at G.



Step 4: Then we operate 10 left and 24 down, two blue cells will gather at one cell (green cell above). Our final task is to put green cell at G. We need to operate 10 up, 5 right, 2 down, 3 left, 6 down, 3 right and 3 up.

In conclusion, total sequence is {2*right, 2*down, 1*left, 2*down, 6*left, 6*up, 3*left, 1*up, 3*right, 2*up, 22*up, 34*left, 10*down, 34 * right, 10*left, 24*down, 10*up, 5*right, 2*down, 3*left, 6*down, 3*right and 3*up}.

c) Find the shortest possible sequence of moves that guarantees you will end up at G regardless of where you started.



To finish this task, we can still take Step 1 and Step 2 in part b. After that, we operate 34 up and 34 left, all starting cells in the maze will end up at several cells (red cells above). Then we operate 10 down, 24 left to gather cells in the upper two level. And operate 10 up and 24 left to gather lowest level. Next, we operate 10 right and 34 down, all cells will end up at one cell (pink cell above). Finally, we need to find a sequence to make this cell end up at G. We operate 10 up, 19 right, 2 down, 3 left, 6 down, 3 right and 3 up.

Total sequence is {2*right, 2*down, 1*left, 2*down, 6*left, 6*up, 3*left, 1*up, 3*right, 2*up, 34*up, 34*left, 10*down, 24*left, 10*up, 24*left, 10*right, 34*down, 10*up, 19*right, 2*down, 3*left, 6*down, 3*right and 3*up}.

d) Suppose that after each move, you receive an observation / feedback of the form Yt = the number of blocked cells surrounding your location. Let Y0 be the number of blocked cells surrounding your starting location. Again, you get no feedback if the move was successful or not, simply the number of blocked cells surrounding your current location.

d.1) You initially observe that you are surrounded by 5 blocked cells. You attempt to move LEFT. You are surrounded by 5 blocked cells. You attempt to move LEFT. You are surrounded by 5 blocked cells. Indicate, for each cell, the final probability of you being in that cell.

d.2) Write an algorithm to take a sequence of observations {Y0; Y1; : : : ; Yn}and a sequence of actions {A0;A1; : : : ;An□1} and returns the cell you are most likely to be in.

d.1**)** To do this task, firstly I scan all white cells and make a dictionary of number of each white cell's neighbor blocked cells. Then I look at Y_0, and test all values in the dictionary and record all keys whose value equals to Y_0 to get a set CELLS. Next, I look at A_0. Record length of CELLS as l, then drop first of set and get the testing cell, move it according to A_0. If it's not blocked, update the position. Otherwise, keep the

position. Then test this new position according to Y_1 and dictionary. If it satisfies, insert it to the end of CELLS. Repeat this l times. Then I look at A_1 and Y_2, and continue.

After all operations, I get a CELLS with no repeating positions, which means that probabilities of being in each cell in CELLS are same. The probability of being in (1, 5) or (1, 17) or (1, 29) or (2, 1) or (2, 11) or (2, 13) or (2, 23) or (2, 25) or (2, 35) or (3, 4) or (3, 5) or (3, 6) or (3, 16) or (3, 17) or (3, 18) or (3, 28) or (3, 29) or (3, 30) or (4, 3) or (4, 9) or (4, 15) or (4, 21) or (4, 27) or (4, 33) or (5, 5) or (5, 17) or (5, 29) or (8, 3) or (8, 9) or (8, 15) or (8, 21) or (8, 27) or (8, 33) or (9, 4) or (9, 5) or (9, 6) or (9, 16) or (9, 17) or (9, 18) or (9, 28) or (9, 29) or (9, 30) or (10, 1) or (10, 35) or (13, 5) or (13, 17) or (13, 29) or (14, 1) or (14, 35) or (15, 4) or (15, 5) or (15, 6) or (15, 16) or (15, 17) or (15, 18) or (15, 28) or (15, 29) or (15, 30) or (16, 3) or (16, 9) or (16, 15) or (16, 21) or (16, 27) or (16, 33) or (17, 5) or (17, 17) or (17, 29) or (20, 3) or (20, 9) or (20, 15) or (20, 21) or (20, 27) or (20, 33) or (21, 4) or (21, 5) or (21, 6) or (21, 16) or (21, 17) or (21, 18) or (21, 28) or (21, 29) or (21, 30) or (22, 1) or (22, 35) or (25, 5) or (25, 17) or (25, 29) or (26, 1) or (26, 35) or (27, 4) or (27, 5) or (27, 6) or (27, 16) or (27, 17) or (27, 18) or (27, 28) or (27, 29) or (27, 30) or (28, 3) or (28, 9) or (28, 15) or (28, 21) or (28, 27) or (28, 33) or (29, 5) or (29, 17) or (29, 29) or (32, 3) or (32, 9) or (32, 15) or (32, 21) or (32, 27) or (32, 33) or (33, 4) or (33, 5) or (33, 6) or (33, 16) or (33, 17) or (33, 18) or (33, 28) or (33, 29) or (33, 30) or (34, 1) or (34, 11) or (34, 13) or (34, 23) or (34, 25) or (34, 35) , is 1/128. The probability of being in other white cells is zero.

d.2) Using the same method to get the set of final possible cells. Count the appearance of each different cells, and sort them, then return the position of one of cells appearing mostly.

Question 2 - Markov Decision Processes

a) For each of the 10 states, what is the optimal utility (long term expected discounted value) available in that state (i.e., U^*(state))?

There are three types of states—New, $Used_i$ and Dead.

For state New, the formula of its utility is

$$U^*(New)=100+ \beta*(1*U^*(Used_1)).$$

For state $Used_i$ (i-th state Used), the formula of its utility is

$$U^*(Used_i)=\max\{100-10*i+\beta*[(1-0.1*i)* U^*(Used_i)+0.1*i* U^*(Used_{i+1})], -250+\beta* U^*(New)\}.$$

For state Dead, the formula of its utility is

$$U^*(Dead)= -250+\beta* U^*(New).$$

Assume A is the set of utility of all states which is [U^*(New), $U^*(Used_1)$, $U^*(Used_2)$, $U^*(Used_3)$, $U^*(Used_4)$, $U^*(Used_5)$, $U^*(Used_6)$, $U^*(Used_7)$, $U^*(Used_8)$, U^*(Dead)], and initialize A as [100,90,80,70,60,50,40,30,20,10]. After many times iterations, the optimal value of A is [800.53, 778.37, 643.22, 556.12, 502.84, 475.85, 470.48, 470.48, 470.48, 470.48].

b) What is the optimal policy that gives you this optimal utility - i.e., in each state, what is the best action to take in that state?

The optimal policy is that USE the machine when the machine is in the state **NEW** or **Used1** or **Used2** or **Used3** or **Used4** or **Used5**. And REPLACE it when the machine is in the state **Used6** or **Used7** or **Used8** or **Dead**.

c) New machines are expensive. Suppose you wanted to open a store selling used machines for less – suppose each used machine you sold had an equal chance of being in Used1 and Used2. If you gave these machines away for free, no one would buy New machines and everyone would come to you instead. If you sold them at 250, no one would buy them. What is highest price you could sell your used machines for, while ensuring that (rational) users would buy them?

Assume that the price we sell used machines for is K. Then according to question, the highest value of K is the highest value that makes $(K/250)* U^*(New) < 0.5*(U^*(Used_1) + U^*(Used_2))$. Make $U^*(New)$, $U^*(Used_1)$ and $U^*(Used_2)$ equal to their optimal values, and highest values of K is 221.

d) For different values of β (such that 0 < β < 1), the utility or value of being in certain states will change. However, the optimal policy may not. Compare the optimal policy for β = 0.1, 0.3, 0.5, 0.7, 0.9, 0.99, etc. Is there a policy that is optimal for all sufficiently large β? What do you make of it?

Assume that POLICY is the set of actions taken in each state, POLICY=[$Action_{New}$, $Action_{Used1}$, $Action_{Used2}$, $Action_{Used3}$, $Action_{Used4}$, $Action_{Used5}$, $Action_{Used6}$, $Action_{Used7}$, $Action_{Used8}$, $Action_{Dead}$]. When we choose to use machine, we mark that action as 1. When we choose replace machine, we mark that action as 0.

When β = 0.1, POLICY=[1, 1, 1, 1, 1, 1, 1, 1, 1, 0].

Whenβ = 0.3, POLICY=[1, 1, 1, 1, 1, 1, 1, 1, 1, 0].

Whenβ = 0.5, POLICY=[1, 1, 1, 1, 1, 1, 1, 1, 1, 0].

Whenβ = 0.7, POLICY=[1, 1, 1, 1, 1, 1, 1, 1, 1, 0].

Whenβ = 0.9, POLICY=[1, 1, 1, 1, 1, 1, 0, 0, 0, 0].

Whenβ = 0.99, POLICY=[1, 1, 1, 1, 0, 0, 0, 0, 0, 0].

When β is smaller than 0.8, the optimal policy is same, [1, 1, 1, 1, 1, 1, 1, 1, 1, 0]. The optimal policies are different when β is from 0.8 to 0.99. There is no optimal policy for all sufficiently large β. According to this, I know that, when we calculate the optimal utility, we shouldn't set β very large, otherwise we will get a very specific optimal policy with high probability of incorrect when we actually apply it.

Question 3 – Classification

a) Construct a model to classify images as Class A or B, and train it on the indicated data. What does your model predict for each of the unlabeled images? Give the details of your model, its training, and the final result. Do the predictions make sense, to you?

Our goal is to classify unlabeled images as Class A or B, so we can use some simple classifier like KNN.

Firstly, we need to preprocess training data and testing data. Original image data is a 5*5 matrix, transfer them to 1*25 vectors. Then calculate all Euclidean distances between vector of testing data and vector of each training data, and get a set of Euclidean distances. After that, we sort this set and choose smallest k values, k is an odd number. Then we find labels of the training data represented by the Euclidean distances we picked. And compare the appearance of 'Class A' and 'Class B'. If 'Class A' appears more, we predict unlabeled image as Class A. If 'Class B' appears more, we predict unlabeled image as Class B.

My model predict unlabeled images as Class B, Class A, Class B, Class A, and Class B. These predictions make sense to me. Because when we look at unlabeled images, we can find that black cells in image 1 and 3 gather at the lower right corner, black cells in image 2 and 4 gather at the upper left corner, and number of black cells in same positions in image 5 and Class B is higher than number of black cells in same positions in image 5 and Class B.

b) The data provided is quite small, and overfitting is a serious risk. What steps can you take to avoid it?

We can separately drop one image in Class A and Class B, and don't train my model on the data of these two images.

c) Construct and train a second type of model. Specify its details. How do its predictions compare to the first model?

I choose Naïve Bayes classifier as second type of model. Considering that there are some values not appearing in Class A or Class B, I use Laplacian smoothing when I calculate the probability.

When I train model with data from Class A, I calculate probabilities of each item equals to 1 in the training data vector with Laplacian smoothing, like $p(cell_i=1)$=((number of images in Class A when $cell_i$ is 1)+1)/(number of images in Class A +2)

Then I use a set to record all these probabilities called px, px=[$p(cell_1=1)$, $p(cell_2=1)$,…, $p(cell_{25}=1)$]. And do the same thing to train model with data from Class B and get a set py.

When we predict an unlabeled image, firstly we initialize two variable resultx and resulty, numbers of images in Class A and Class B are same. So resultx=0.5 and resulty=0.5. Then we test i-th item in the testing data vector from the beginning to the end, if it equals to 1 then resultx = resultx * px[i] and resulty = resulty * py[i] , if it equals to 0 then resultx = resultx * (1-px[i]) and resulty = resulty * (1-py[i]). After testing all 25 items in the testing data vector, we compare resultx and resulty. If resultx is larger than resulty, then we classify the testing data as Class A. If resulty is larger than resultx, then we classify the testing data as Class B.

The result of prediction is also Class B, Class A, Class B, Class A, and Class B.

Question 4 - The Big Picture

SheepdogBot

1) Find shortest path to home

When Bot are taking sheep out, they need to find shortest path to come back home which will save most energy. In reality, it is hard to apply BFS or DFS because we should minimize the step we take. But we can apply A* to find shortest path, direction and distance to home can be recorded easily. So, when we decide to go home. We can find optimal nearest next step according to direction to home and distance that we have travelled.

2) Find ill sheep

We can give some probabilities of some observations and whether sheep is ill to Bot, and it finds ill sheep. Observations are like how much sheep eats, how fast sheep moves, how fat sheep is. And give some probability like P(sheep eats less than 4 pounds| sheep is ill)=0.8, and according to these probabilities and actual observation of sheep, apply Bayes' theorem to decide whether sheep is ill.

3) Decide whether taking sheep out or not today

We can according season, temperature, sunshine and so on to create a Bayesian Networks to guess the probability of that sheep can get enough grass to eat and will not face dangers today. If the probability is high enough, bot decides to take sheep out, otherwise it won't to take sheep out.

4) According to environment, choose where it takes sheep to

When bot is taking sheep out, it can choose where it takes sheep to. We can use different symbols in the environment to find best choice based on Logical Inference. Like if it finds tracks of wolves toward left, it will know left is dangerous and take sheep right.

5) Action of whether forcing sheep to keep together

When sheep are moving, we can define different states of sheep based on sparseness of sheep. And there are probabilities of sheep becoming sparser and transferring to next state. At state of sparser, speed of sheep moving will decrease. When sheep is totally dispersed, bot will force sheep to keep together and transfer to state which is very tight. Then at every state, bot needs to choose to force sheep to keep together or let sheep go based on optimal utility.