Google colab actual code- project3_AI.ipynb
Google colab test code - test_project3_ai.ipynb

Have tested my code using the example given in the markov decision process slides for value iteration algorithm, in test_project3_ai.ipynb file, where all the answers were proved to be accurate and the code for project3 is in project3_AI.ipynb file.

Code logic flow

- We can divide the whole code into two parts
    1. For running iteration value algorithm
    2. For visualising using both table and direction policy

Data preparation

- We assign values to the variables for R1, R2, r and Hazard H, destination D, obstacle, utility U.
- Utility U, reward are dictionaries with keys as the positions on the grid and for obstacles, it is assigned None

Iterative value algorithm

- We iterate the logic till we finish the iterations or till we get the difference in the current utility value and previous utility value less than threshold value(0.01) for all states.
- The main logic here is to get the visiting order of the neighbours starting from the destination till we complete all the non-obstacle and non-hazard positions. To get the visiting order I used the “pattern_order()” function.
- After getting this order now, we run the iterations,where we update the utility values for all the positions.
- To update the utility values, we use the function

$$U_{k+1}(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s,a) U_k(s')$$

- To get the product of maximum probability and utility, I used the “Transprob()” function, where probability is 0.9 if it goes in the intended direction, 0 for opposite direction, 0.1/(number of non-block positions surrounding it) otherwise.
- This “valueIteration()” function returns the utility values after iterating the algorithm for the number of iterations mentioned.

Visualising table and directions

- We take the output from “valueIteration()” function which is a dictionary with keys as the positions on the grid and values as the utility values, we convert this into a list of lists such that we can use python's inbuilt tabulate function to visualise the table using function “visualise_table()”
- To get the directions we assign the directions such that the arrow is pointing to the max value of the up, down, left and right possible directions using the function “get_arrow” and we iterate this for all positions in the grid using the “visualise_policy” function.

Have run all the policy for 100 iterations and stopped if the difference in the current and previous utility value is less than 0.01.

T1. a,b ->
The utility values after 100 iterations of the value iteration algorithm are shown below along with the directions of the policy. R1, R2, r = 10, -5, -5

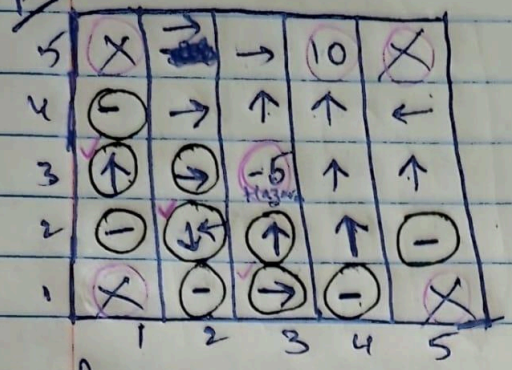
X	-2.96	3.12	10	X
-6.26	-7.76	-2.69	2.84	-3.06
-11.01	-9.9	-5	-3.28	-7.94
-6.59	-11.08	-9.93	-8.39	-6.32
X	-6.59	-11.07	-6.32	X

X	→	→	10	X
←↑	→	↑	↑	←
↑	→	−5	↑	↑
↓←	↓←	↑	↑	→↓
X	↓←	→	→↓	X

T2.
Total possible paths are 5X5 = 25
Out of which 4 are obstacles, 1 hazard and 1 is destination
So, 25 - 6 = 19 possible start positions.
From them there are blocked positions which are highlighted with black colour circle in the figure below. They are (2,1), (1,2), (1,4), (4,1), (5,2). So, count of blocked is 5
And there are cases where the direction is going towards the blocked positions mentioned above. They are (2,2), (3,1), (1,3). So, count of blocked is 5+3 = 8
And there are cases where the direction is going towards hazard. They are (2,3), (3,2). So, the total count of blocked is 8+2 = 10.

Now, **total paths that reach destination are 19-10 = 9 paths**
Fraction of shortest paths that reach destination are 9/9 = 1

P1



In general

Total = 5×5 = 25 start positions

25 − 4 − 2 (Destination; Hazard) = 19 start positions possible.
(obstacles)

For P1

Blocked positions are 5 (represented as ⊖)
+ (2,2) position + (1,3) Pos + (3,1) Pos

Reason: It moves towards blocked positions.

So, 5+3=8.

Plus we see (3,2), (2,3) positions going towards Hazard; which is a deadend.

So, 8+2=10.

1) ∴ Total paths are 19−10 = 9 paths to reach destination.

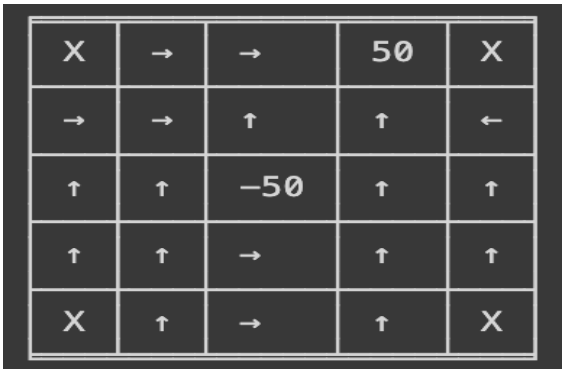
2) All are shortest paths.

So, $\frac{9}{9} = 1$

T3.

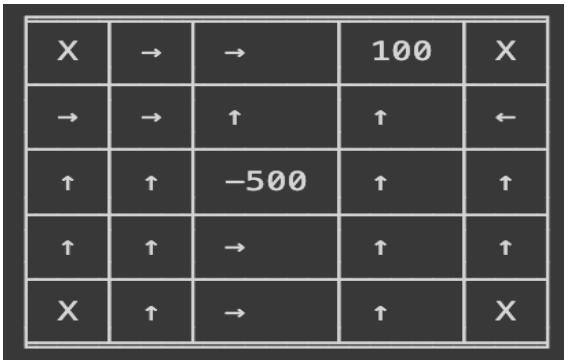
1. Policy 2 - R1=50, R2=-50 and r=-5

X	28.39	38.53	50	X
11.45	20.04	28.82	37.99	26.6
4.91	9.2	-50	24.35	18.47
-0.95	2.65	5.35	15.47	11.16
X	-2.79	1.46	7.6	X



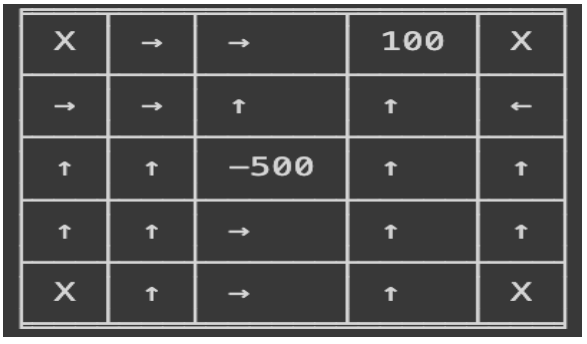
2. Policy 3 - R1=100, R2=-500, r=-5

X	67.56	82.79	100	X
40.6	54.62	68.2	81.93	63.64
30.27	22.63	-500	41.14	50.68
21.13	14.59	6.9	30.4	39.18
X	7.37	12.2	20.17	X



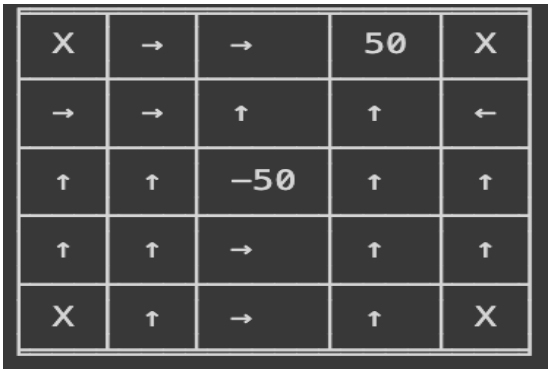
3. Policy 4_1 by using R1 and R2 from policy 3 - R1=100, R2=-500, r=-1

X	76.21	87.38	100	X
55.14	66.66	76.68	86.69	72.04
47.71	42.29	-500	55.84	62.69
41.14	36.46	30.2	48.04	54.39
X	30.05	33.82	39.43	X



Policy 4_2 by using R1 and R2 from policy 2 - R1=50, R2=-50 and r=-1

X	37.03	43.12	50	X
25.85	31.93	37.29	42.75	34.98
21.99	23.6	-50	32.73	30.16
18.54	19.85	19.92	27.57	25.83
X	15.92	18.7	22.17	X



In General we have 19 possible paths to the destination shown above and we need to subtract the blocked positions and those that are moving towards hazard, but here for all policy 2, 3 and 4 we don't have any blocked positions.So, we have a total of 19 paths that reach the destination. And these paths are the shortest paths possible so the fraction is 19/19 = 1.

1. Compare policy 2 with policy 1.
   Policy 2 performs better than policy 1.
   Reason -
   1. Policy 2 has more paths reaching the destination i.e 19 paths than policy 1, which has 9 paths.
   2. Both the policy's produced the paths that are shortest hence both have a fraction of 1.
2. Compare policy 3 with policy 1 and policy 2
   1. Policy 3 is the same as policy 2. The directions are all the same only the values vary that is because the values are relative to the rewards R1, R2 and r.
   2. Hence Policy 3 = Policy 2 and policy 3 performs the same as policy 2 w.r.t policy 1 i.e policy 3 is better than policy 1
3. Compare policy 4 with best from policy 1,2,3
   We know the best policy is 2 and 3 they both are the same but with different R1, R2 values so I tried using both these values. Policy_4_1 is from policy 3(R1,R2) values and Policy_4_2 is from policy 2(R1,R2) values.
   1. Here as well both policy_4_1 and policy_4_2 show the same policy; only the values change(because of different R1 and R2 values) but the directions remain the same. So, let's consider these both policies' as policy 4
   2. Policy 4 is the same as policy 2 and policy 3. Hence they both perform the same.
   3. Here, for r=-1 and for r=-5 we got the same policies. Then how can we say which is better? This depends on the R1 and R2 values. The ratio of these values w.r.t r value. If the ratio of R1/R2 and R2/r is less than and nearer to 1, then the live-in reward is the same as hazard reward; this will lead to less accurate paths to destination because now our robot tries to jump out as early as possible. But if the ratio is greater than 1 and far away from 1(as seen in the cases in policy2,3,4 where the ratio is 10 and 100); then the robot can explore the possible paths before exiting and can get the optimal paths to reach its destination. So, the ratio is

(R1/R2) X (R2/r) = R1/r. This ratio is what matters for deciding the performance. Now considering this logic, for policy 1, we have 10, -5, -5 as R1,R2 and r and the ratio of R2/r is 1;R1/R2=-2 i.e < 1 and here we got 9 paths that reach destination out of 19 possible paths for policy 1, because remaining paths are either blocked or leading to exits (hazard).
But when we consider policy 2 and 3 where R1=50, R2=-50, r=-5 and R1=100, R2=-500, r=-5 , here clearly the ratio of R2/r > 1 and far from 1 i.e 10 and 100 times respectively and R1/R2=-1,-0.2 < 1. And in this case we got the optimal policy for both of these. Thus confirming the logic. Now in policy 4, we took r=-1 which further increased this ratio to 50 and 500 respectively making it more prone to giving optimal solutions but since we already have optimal policy with r=-5 for these R1 and R2 values with r=-1 we got the same.

Now, when we use r=-1 for R1=10, R2=-5 we will get a better policy than r=-5(policy 1). The results are

X	5.68	7.71	10	X
2.3	4.02	5.77	7.6	5.32
1.0	2.08	-5	5.1	3.71
-0.16	0.74	1.47	3.3	2.25
X	-0.38	0.46	1.69	X

X	→	→	10	X
→	→	↑	↑	←
↑	↑	−5	↑	↑
↑	↑	→	↑	↑
X	↑	→	↑	X

Here, we have a total of 19 possible paths and all are shortest paths hence this(r=-1) performs better than policy 1 with r=-5 in this case. But we need to note that ratio is what determines which one performs better.

T4.
The usual formula for calculating utility value is

$$U_{k+1}(s) = r(s) + \gamma * \max_a \sum_{s'} TransProb(s, a) * U_k(s')$$

Now, when we have robots moving we have a certain probability of those robots being present in the given location. And the hint is given to assume we have a formula to calculate this probability of finding a robot in the given grid position.
Now, if the probability value is showing the chances of seeing a robot in a particular location then (1 - probability value) gives the chance of not seeing a robot in that location of the grid position. This probability needs to be multiplied with the probability of the intended direction such that we get the probability of going in the intended direction/other directions along with the absence of robots present in that location.

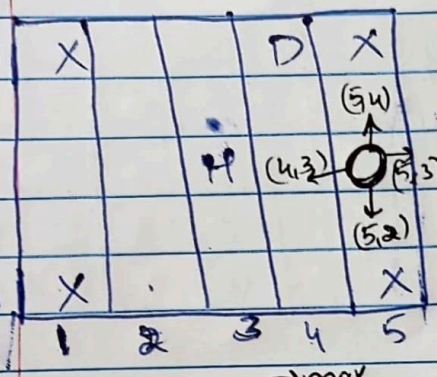
Why are we multiplying these both probabilities?
Because the event of going in a certain direction and a robot being present in a particular grid position are completely independent of each other. Hence we can multiply by due to these independent events.

Now, where in the value iteration function do we need to make changes?
We need to make changes to the utility function such that we include the probability of not finding a robot in the function. So, I have modified the given equation to the below equation.

$$U_{k+1}(s) = r(s) + \gamma * \max_a \sum_{s'} (1 - Prob_{robot}(s')) * TransProb(s, a) * U_k(s')$$

Here, we have introduced the probability of the robot of the next state(s') in the utility value iteration function.

A working example is attached in the next page.



$$U(5,3) = R(5,3) + \gamma \max_a \left[(1 - \text{prob\_robot}(5,3)) \times \text{TransProb}(5,3)_a \times \text{Utility Value} \right]$$

up, down, left, right

intended up (1)

$$\left(1 - \text{Prob\_robot}(5,4)\right) \times 0.9 \times U(5,4) + \left(1 - \text{prob\_robot}(4,3)\right) \times \frac{0.1}{2} \times U(4,3)$$

(0.9: up; 0.1/2: left)

$$+ \left(1 - \text{Prob\_robot}(5,3)\right) \times \frac{0.1}{2} \times U(5,3)$$

(0.1/2: right)

intended down (2)

$$\left(1 - \text{Prob\_robot}(5,2)\right) \times 0.9 \times U(5,2) + \left(1 - \text{Prob\_robot}(4,3)\right) \times \frac{0.1}{2} \times U(4,3)$$

(0.9: down; 0.1/2: left)

$$+ \left(1 - \text{Prob\_robot}(5,3)\right) \times \frac{0.1}{2} \times U(5,3)$$

(0.1/2: right)

intended left (3)

$$\left(1 - \text{Prob\_robot}(4,3)\right) \times 0.9 \times U(4,3) + \left(1 - \text{Prob\_robot}(5,4)\right) \times \frac{0.1}{2} \times U(5,4)$$

(0.9: left; 0.1/2: up)

$$+ \left(1 - \text{Prob\_robot}(5,2)\right) \times \frac{0.1}{2} \times U(5,2)$$

(0.1/2: down)

intended (4)

right $(1 - \text{Prob\_robot}(5,3)) \times 0.9 \times U(5,3)$ (right)

$+ (1 - \text{Prob\_robot}(5,4)) \times \frac{0.1}{2} \times U(5,4)$ (up)

$+ (1 - \text{Prob\_robot}(5,2)) \times \frac{0.1}{2} \times U(5,2)$ (down)

from (1), (2), (3), (4) → we take max.
let it be max_value.

So,

$$U(5,3) = R(5,3) + \gamma \times \text{max\_value}$$

Now, the max_value will be different from the original one.