

Gridworlds

What maps you used for testing. The provided maze is just for documentation. You should have at least two other maps to test and tune your algorithm. Explain how the maps guided your development. Please do not use more than 5 maps in your writeup.

Terminal States:

- Green - Positive Rewards
- Red - Negative Rewards
- Yellow - Start State

Map 1

1	0	0	0	0	0
-1	0	0	0	0	0
0	0	0	0	0	0
S	0	0	0	0	0

Map 2

2	0	0	0	-1	3
-1	0	0	0	-1	0
0	0	0	0	0	0
S	0	0	0	-1	0

Map 3

0	3	-1	-1	-2	5
0	0	0	0	0	0
0	-2	0	0	-2	-1
0	0	0	0	0	0
0	0	0	-1	2	0
S	0	0	0	0	0

Map 4

0	0	0	0	0	0	0
0	-1	-1	-1	-1	-1	0
0	-1	20	0	0	-1	0
0	-1	-1	-1	0	-1	0
0	0	0	0	0	-1	0
-2	-2	-2	-2	-2	-2	0
S	0	0	0	0	0	0

Maps 2 & 3 were designed to check if the algorithm has a good exploration policy. When the moves are probabilistic it makes sense to choose a positive reward far away from a negative reward to avoid the risk of falling into a negative reward. We tested if the algorithm would do the same. In the case where moves are deterministic, the algorithm should be able to reach the highest positive reward.


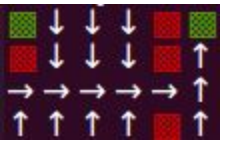
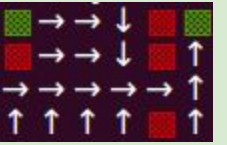
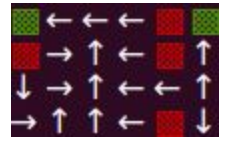
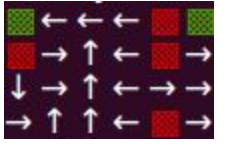

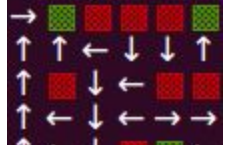
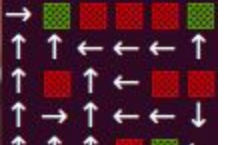


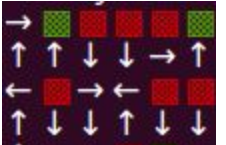
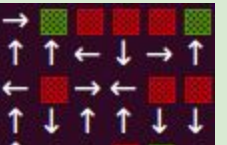
Map 4 was designed to see if the agent is able to find the solution to the maze even if it is too deeply hidden. It works when the move is deterministic but doesn't work well when the move is probabilistic since there is a lot of negative rewards where it could fall into. This map helped us understand how the size of the reward could affect the algorithm.

The move cost was -0.04.

How you came up with an exploration policy that does a good job on a variety of maps. How did you set the step size parameter?

Exploration policy

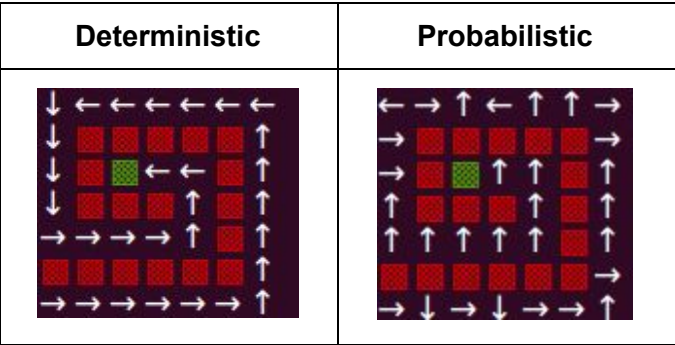
Initial exploration value(eps) is 1.0, it is decayed at a rate of 0.999995 at every iteration. If a random number chosen between 0 and 1 is lesser than eps our agent explores an action randomly else it follows the policy. Following table shows the results for different decay rates tested. The runtime here was 20 secs.

Decay Rate ->	0.9995	0.99995	0.999995
Map 2 (Deterministic moves)	 Preferred Goal : Left Top (+2)	 Preferred Goal : Right Top (+3)	 Preferred Goal : Right Top (+3)
Map 2 (Probabilistic moves) (Probability = 0.8)	 Preferred Goal : Left Top (+2)	 Preferred Goal : Left Top (+2)	 Preferred Goal : Left Top (+2)
Map 3 (Deterministic moves)	 Preferred Goal : Left Top (+3)	 Preferred Goal : Left Top (+3)	 Preferred Goal : Right Top (+5)
Map 3 (Probabilistic moves) (Probability = 0.8)	 Preferred Goal : Left Top (+3)	 Preferred Goal : Left Top (+3)	 Preferred Goal : Left Top (+3)

As explained in the previous question we used the maps 2 and 3 to tune our exploration parameter epsilon. With deterministic moves, a higher epsilon value of 0.999995 was needed to ensure that the world is sufficiently explored and chooses to go to a higher reward even if it is far away or hidden.

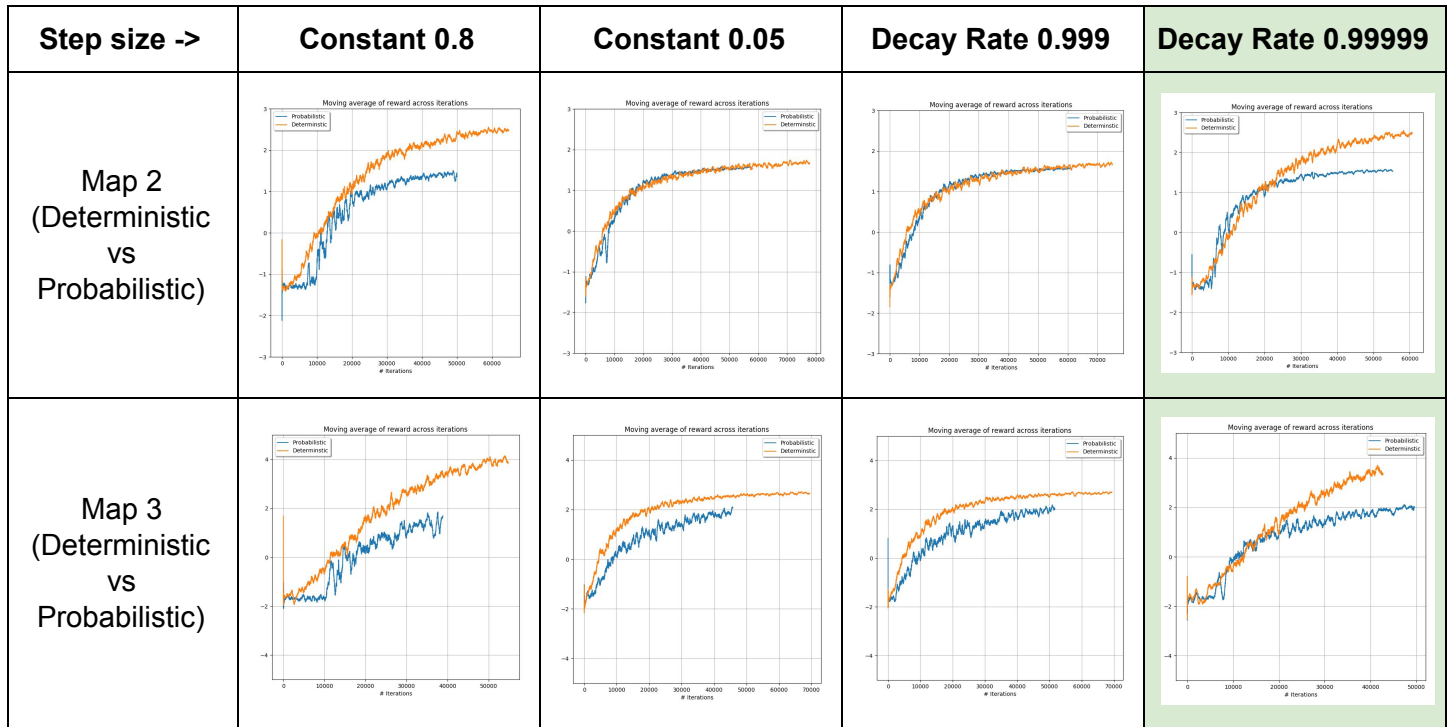
With probabilistic moves, the three parameters give almost the same result, and hence we went ahead with a decay rate of 0.999995. We can see that the algorithm has learnt to go to a reward with lesser risk rather than choosing the one with highest reward.

This was also able to find the reward in case of the Map 4 also, with deterministic moves. Following are the results for Map 4.



Step size parameter

The step size parameter is initialized with 0.8 and is decayed at a rate 0.99999 with a minimum value 0.05 to ensure that it learns at every step and doesn't converge due to the step size. Following graphs shows the performance of the agent with different step size parameters. The y-axis is the moving average of reward received over its last 500 iterations and x-axis is the number of iterations. The time limit for these runs were for 60 secs.



We can see that if the step size is too low(0.05) the agent is not able to find high rewards that are far away and tends to converge at smaller rewards. When the step size is too high(0.8) the agent tends to oscillate more for deterministic moves and is not able to converge to a solution but is able to find and propagate high rewards that are far away. We have combined the essence of good characteristics of both these step sizes and come up with a step size that is high in the beginning and decays as iteration increases and reaches a minimum value of 0.05 to ensure that it keeps learning till the end.

We have also compared the effect of a high decay rate(0.999) and the rate we chose(0.99999), we can see that if the step size decays fast the agent tends to converge at the local optimum.

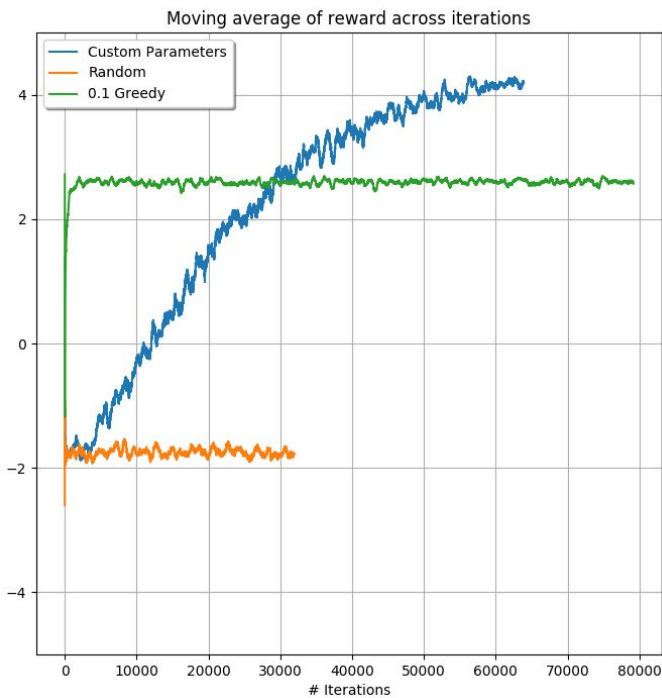
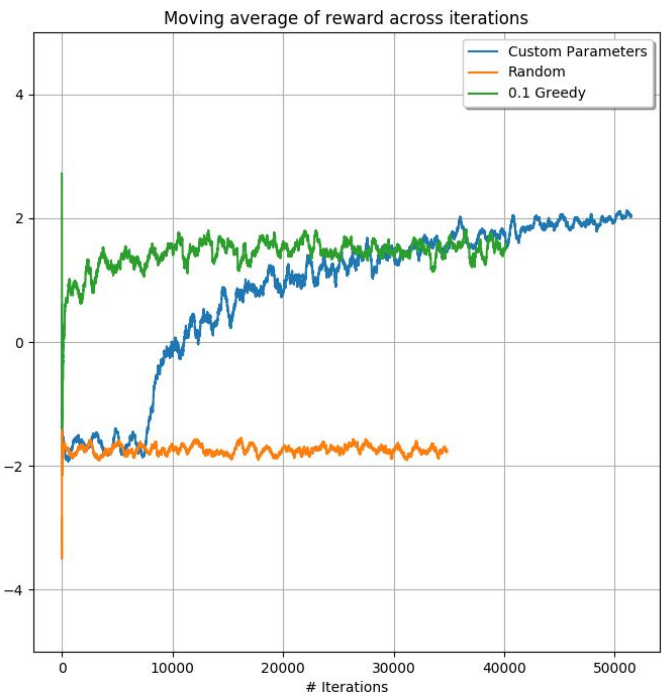
Plot a graph of performance on one of your maps. The x-axis should be the number of training runs, and the y-axis should be the average reward received.

The plot of performance for Map 3 is shown along with the plot for the next question.

Plot performance (on the same) graph of two simple policies and contrast their performance with your agent:

- **Random:** an agent that makes a random action at each timestep. Comparing performance with random is a good baseline. If your agent cannot beat random performance it is not doing very well.
- **ϵ -greedy with $\epsilon=0.1$.**

In the performance graph y-axis is the moving average of reward received over its last 500 iterations and x-axis is the number of iterations. The time limit for these runs were for 60 secs.

Map 3 - Deterministic Moves	Map 3 - Probabilistic Moves (Probability = 0.8)
 <p>The graph for Map 3 - Deterministic Moves shows the performance of three agents over 80,000 iterations. The y-axis represents the moving average of reward, ranging from -4 to 4. The x-axis represents the number of iterations. The 'Custom Parameters' agent (blue line) starts at a reward of approximately -1.5 and steadily increases, reaching a plateau of about 4.2 after 60,000 iterations. The 'Random' agent (orange line) remains flat at a reward of approximately -1.5 throughout the entire duration. The '0.1 Greedy' agent (green line) starts at a reward of approximately 2.5 and remains flat, indicating it is stuck in a local optimum.</p>	 <p>The graph for Map 3 - Probabilistic Moves (Probability = 0.8) shows the performance of three agents over 50,000 iterations. The y-axis represents the moving average of reward, ranging from -4 to 4. The x-axis represents the number of iterations. The 'Custom Parameters' agent (blue line) starts at a reward of approximately -1.5 and increases to a plateau of about 2.0 after 40,000 iterations. The 'Random' agent (orange line) remains flat at a reward of approximately -1.5 throughout the entire duration. The '0.1 Greedy' agent (green line) starts at a reward of approximately 1.0 and increases to a plateau of about 1.8 after 40,000 iterations.</p>
<p>The ϵ-greedy($\epsilon=0.1$) agent is stuck in a local optimum since it has very less exploration. But our agent was able to explore and find the highest reward.</p>	<p>Though the ϵ-greedy($\epsilon=0.1$) is stuck at a closer reward it is similar to our agent since our agent has decided to be satisfied with a lower reward than risking ending up in a negative terminal state since it is a probabilistic model.</p>

We can see that the random agent performs poorly since it hasn't learnt anything. It takes random moves. But the map is designed in such a way that the probability of reaching the high rewards are very low if random moves are taken.

Our agent performs the best amongst the three under consideration since we could see from the plot that it receives higher average reward with time.

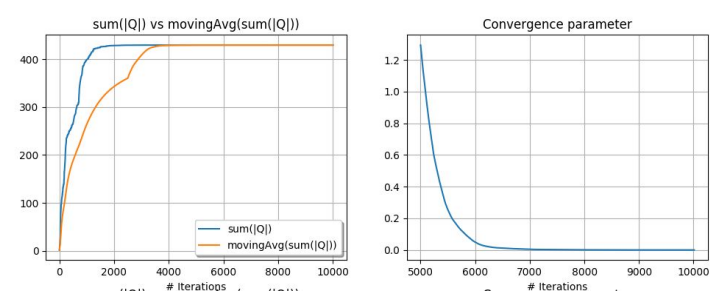
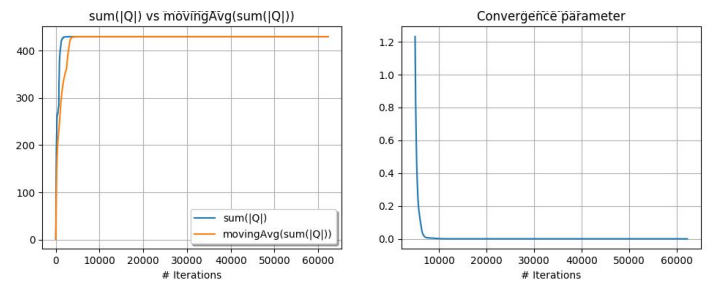
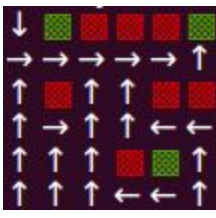
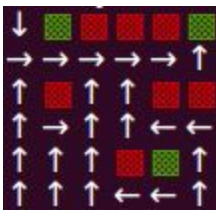
Extra Credit: Have your program stop if “performance” has converged. What that means, exactly, is up to you. Document how you instantiated performance, and explain how your program determines if it is done.

We stop our program if the values in the Q_table have converged. To do that we find the moving average of $\sum(|Q|)$ over its last 2500 iterations. When the moving average curve flattens out we can say that the model has converged i.e. there are no more significant updates to the Q_table.

We take the moving average value at 3 instances T (Current iteration), $T-625^{\text{th}}$ iteration, $T-1250^{\text{th}}$ iteration and we add the absolute difference between $(T, T-625)$ and $(T-625, T-1250)$. If their sum is less than a threshold (0.025) we say the model has converged. This condition is applied only after the 5000^{th} iteration. Also, we do not stop the program if the epsilon value is greater than 0.5, since the agent has to explore more before reaching a conclusion. Otherwise, it could be stuck in a local optimum.

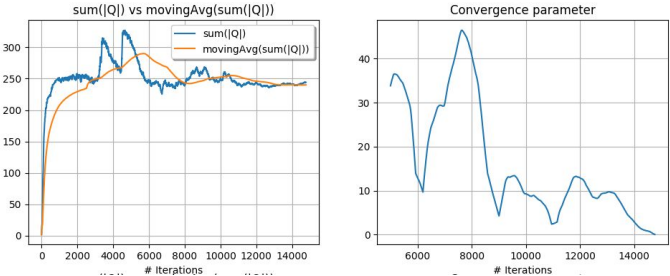
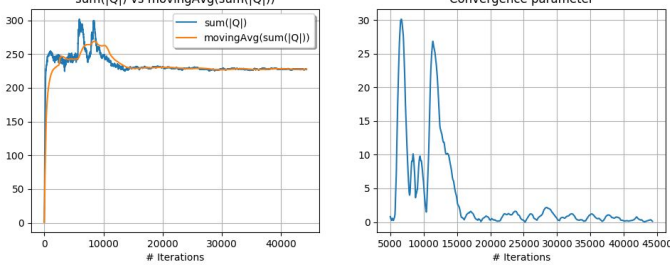
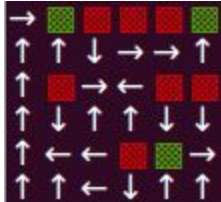
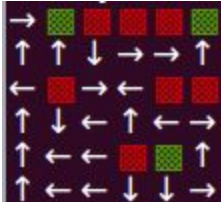
Our program stops if iterations > 5000 & convergence parameter ≤ 0.025 and epsilon ≤ 0.5 .

Following are the results showing the $\sum(|Q|)$, its moving average and the convergence parameter for Map 3 with deterministic moves.

Stopping model based on convergence parameter	Running model for 60.0 sec
	
	
Run time = 12.40 sec	Run time = 60.0 sec
Average Reward received from 1000 trials using this policy = 4.6	Average Reward received from 1000 trials using this policy = 4.6

We can clearly see that policy generated with the convergence criteria has the same performance as running for the full time in a deterministic setting proving the validity of our method.

Following are the results showing the $\sum(|Q|)$, its moving average and the convergence parameter for Map 3 with probabilistic moves with probability = 0.8.

Stopping model based on convergence parameter	Running model for 60.0 sec
	
	
Run time = 23.47 sec	Run time = 60.0 sec
Average Reward received from 1000 trials using this policy = 2.178	Average Reward received from 1000 trials using this policy = 2.240

From the above result we can see that stopping the program based on the convergence parameter has comparable performance as compared to the one running for the complete 60 sec.

