Robotics: Path Planning and Reinforcement Learning Assignment

----------Grid World----------

I have defined each grid world model with an excel file. Each excel file includes a 10 by 10 matrix information which a cell with 0 value indicates an unoccupied position, while a wall is defined by a -1 . Also, I have put 100 for the goal cell.
4-square connectivity (Up, Right, Left and Down) is defined for the grid world, and for the stochastic world, the probability values are 0.6 for the ordered movement direction, and 0.1 for other other 3 actions or remaining in the same cell.

----------Q-Learning----------

- the transition and reward function

The same grid world matrix is used as a reward function, so reward value for directions towards the obstacle is selected as -1, and for the other valid movement is defined as 0. The reward for the goal sate is also 100. The transition method is implemented in the "nextStep.m" function. As the world is stochastic, next step will be in the same direction of the next action just in 0.6 of the cases.

- a description of the exploration policy

I applied the epsilon greedy method as exploration policy. In the " nextAction.m" function, I will generate a random number between 0 and 1, if this number is lower than epsilon next action will be selected randomly and if it is greater next action will be selected based on the maximum Q in the cell. Increasing the epsilon force the next action to be selected more randomly, then each episodes will take more time to be finished, because more pixels will be explored in each episode.

- description of the Q-Learning rule implementation

4 Q values (Up, Right, Down, and left) has been updated in each step based on the following formula:

Q_Up (current sate)= Q_Up(current state) + alpha*(reward(nextStep) + gamma*max(4 Qs of the nextStep) – Q_Up(current sate))

The same formula is also applied to update the Q_Right, Q_Down, and Q_Left.

- a heatmap of the final Q-table (post-learning)

Figure 1 shows a sample heatmap of the Q_table after applying the Q-Learning.

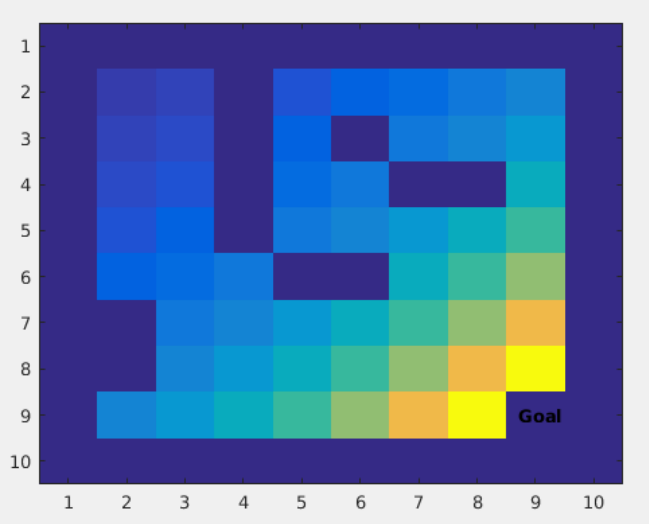

Figure 1

As figure 1 shows, for the pixels which are closer to the goal the Q value is higher.

- the learning and training parameters (number of episodes, alpha, and gamma)

Parameters which are used in this implementation are:
number of episodes = 10000
alpha = 0.9
gamma = 0.8
Increasing the number of the iterations will not change the result after a point, as the program is converged with that number of episodes, and the difference between the Q values and the updated ones would be negligible after that point. Small number of iterations may be not enough for the algorithm to be converged and not give the good result.
Alpha is the learning rate or step size. Setting alpha to zero means that algorithm learns nothing.
Gamma determines how much we want the reward to affect the learning process. Increasing the gamma means that reward is less important in training.

- two example action sequences generated

Figure 2 and 3 shows two different paths for the same grid world and the same Qtable, but different start points.

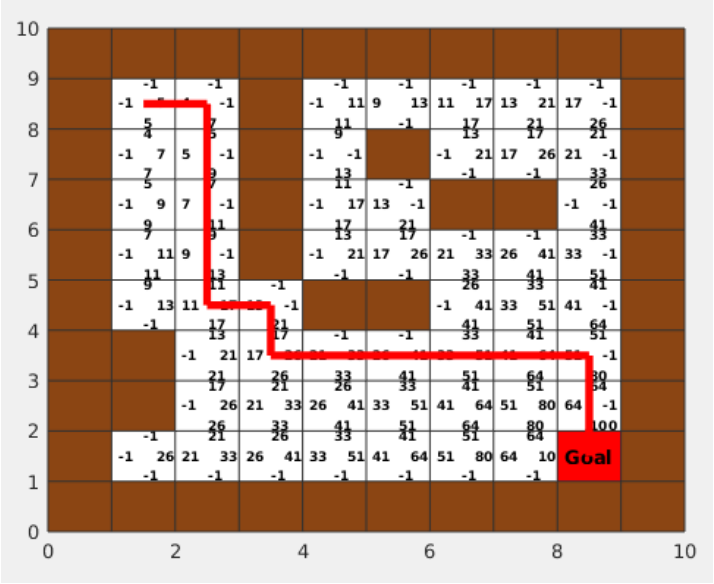

Figure 2

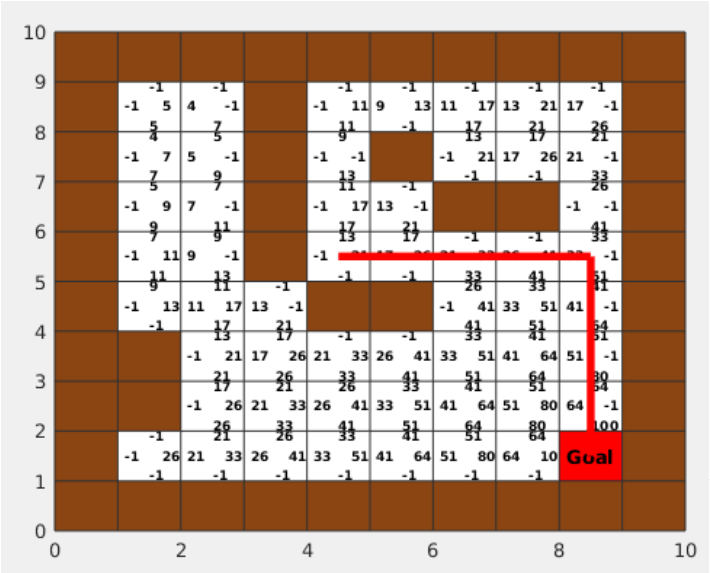

Figure 3

----------A* Planning----------

- pseudocode

- 1 - Set the default g value for each cell as 1000 (a large number compared to final g)
- 2 - Set the default parent value for each cell as 0.
- 3 - Find h for each cell based on the Euclidean distance between the cell and the goal point
- 4 - Put the start point as the current cell

- 5 - check all the neighbors of the current state, and calculate the new g for that cell (add one to g of the current cell) if this new g is smaller than the previous g then change g value of that cell and set the current cell as a parent for that cell.
- 6 - Put the current cell in the closed set
- 7 - Put all the neighbors of the current cell as open set.
- 8 - Find the cell with minimum f value in open set. And consider it as the current cell.
- 9 - Go to step 5

- f, g, and h values for one example map

Figure 4 shows a sample for f, g and h in each cell.

Figure 4

The program has been implemented in Matlab2016b, and the demo is based on Matlab App Designer.