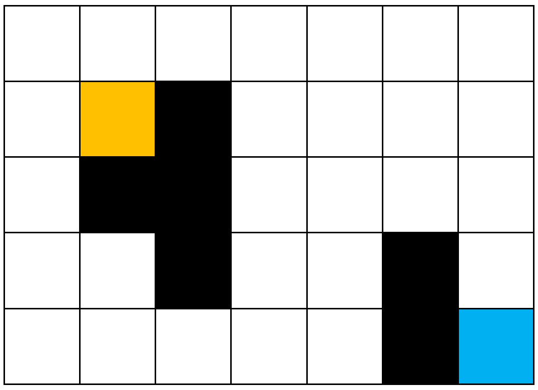
Intelligent Systems (IS Fall 2013)

Assignment 3: Reinforcement Learning

Student: Fang–Lin He



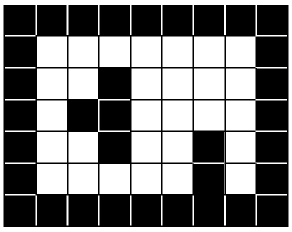
Start: 🟨 Goal: 🟦

Figure 1: **30-State Maze**. For all questions refer to this figure. The agent has four possible actions: North, South, East, West. When the agent takes an action, it goes to the adjacent state in the chosen direction with probability 0.70, and in one of the other directions with probability 0.30. For example, if the agent chooses North, then there is a 70% chance that it actually goes North, a 10% chance it will go South, 10% it will go West, and 10% it will go East. If the agent goes in a direction that will take it outside the maze (e.g. going South in S), it stays in the same state. The reward r is 0 for all state transitions, except that when entering the goal state G the reward is 10.0. The discount factor is set to 0.9. The agent cannot leave the goal state. You may number the states any way you want.

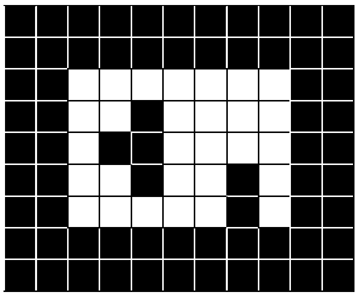
Description of my programs

- States:

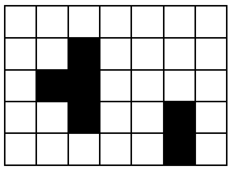
  According to the question requirements, I set my states as a 2-D matrix, so the indices of the states are two dimensional. To fit all the questions, for each question, I used a map (read from a *.txt file) to get its walls / obstacles (value = 0), correct areas (value = 1), start point (value = 2), and goal point (value = 3). For questions 1, 2, and 3, I set outside of the maze also *walls* (black cells), like:



  for Q1 and Q3;

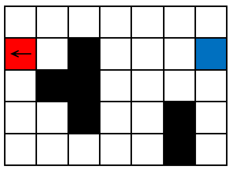


for Q2; and



for Q4 (torus).

Only white cells are correct areas, so only the values / Q-values of white cells are recorded. If the agent moves out of the boundary, it moves to the other side. For example, in the figure below, if the current state is at the point with red background, and the action is to go west, then the next state is at the point with blue background:



- Actions:
  For actions, I recorded their amounts of displacement in rows and columns. For example, to move toward the west, the action is [0, -1]. To move 2 steps toward the west and 1 step toward the north is [-1, -2].
- Rewards:
  I load a text file (*.txt) to obtain the rewards for each state. The size of matrix of rewards should be the same as of states.
- Parameters in the program:
  A. When the agent takes an action, it goes to the adjacent state in the chosen direction with probability <u>0.70</u> → para.prob_follow_act
  B. After the evaluation has converged → $\Delta < \theta$ (See slides IS-RLpart2-2013.pdf, page 9) → θ = para.stop_thres
  C. Discount factor γ → para.discount_factor
  D. Learning rate α → para.learning_rate
  E. Each episode ends after 100 actions → para.max_actions
  F. How many episode to run → para.max_episode
  G. ε -greedy → para.epsilon

My answers to questions

- Question 1

A. (35 points) Implement Policy Evaluation. Starting with $V(s) = 0$, $\forall\, s$, and assuming a random policy ($\pi(s, a) = 1/4$, $\forall$ s, a), what are the final values, $V(s)$, after the evaluation has converged?

- See Q1.m; map file: Q1_map.txt; reward file: Q1_reward.txt; Δ values in each iteration: Q1_A_delta.txt
- The evaluation converges after 89 iterations.
- $V(s)$ after convergence (round up to the 4th digit):

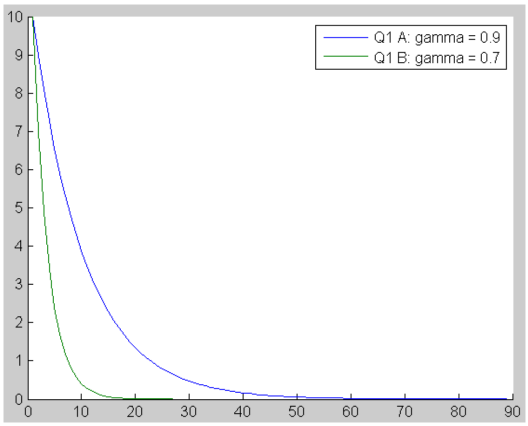
0	0	0	0	0	0	0	0	0
0	0.1808	0.3118	0.7130	1.4319	2.3989	3.8655	5.3553	0
0	0.1305	0.1808	0	1.8211	2.9665	5.5615	9.2268	0
0	0.0885	0	0	1.8752	3.4042	8.6603	20.8658	0
0	0.0863	0.1046	0	1.2348	1.6290	0	53.9859	0
0	0.1046	0.1697	0.3759	0.7498	0.9728	0	99.9915	0
0	0	0	0	0	0	0	0	0

B. (10 points) How and why do the values change if the discount factor, γ is changed to 0.7 (again starting with $V(s) = 0$, $\forall\, s$)?

- See Q1.m; map file: Q1_map.txt; reward file: Q1_reward.txt; Δ values in each iteration: Q1_B_delta.txt
- The evaluation converges after 27 iterations.
- $V(s)$ after convergence (round up to the 4th digit):

0	0	0	0	0	0	0	0	0
0	0.0004	0.0014	0.0058	0.0204	0.0543	0.1335	0.2538	0
0	0.0002	0.0004	0	0.0363	0.1023	0.3217	0.8101	0
0	0.0001	0	0	0.0488	0.1728	0.7929	3.2442	0
0	0.0001	0.0002	0	0.0214	0.0441	0	13.6927	0
0	0.0002	0.0005	0.0023	0.0079	0.0140	0	33.3311	0
0	0	0	0	0	0	0	0	0

- Comparison of delta values (x-axis: iteration time; y-axis: delta value)



From this plot, we can clearly see that, with smaller discount factor, it is more *shortsighted*, and the evaluation converges faster. The reason that it converges faster with smaller discount factor γ is that, the values are smaller (because: $V_{k+1}(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} P^a_{ss'}\left[R^a_{ss'} + \gamma V_k(s')\right]$); with smaller values, the difference between updated values and original values (Δ) are smaller, so of course it converges faster ($\Delta < \theta$).

- Question 2

A. (40 points) Implement Q-learning with learning rate $\alpha = 0.4$. Initialize $Q(s, a) = 0$, $\forall$ s, a. Starting each episode in state *S*, run Q-learning until it converges, using an ε-greedy policy. Each episode ends after 100 actions or once the goal, G, has been reached, whichever happens first.

◆ See Q2.m; map file: Q2_map.txt; reward file: Q2_reward.txt

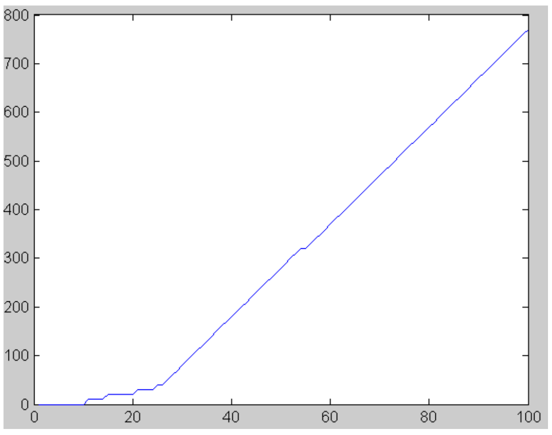
◆ Run Q2.m, and see Q2_path.txt. It records how the agent moves for each episode.

◆ The following table is the best action according to the final *Q*(*s*, *a*). (It's not always the same.)

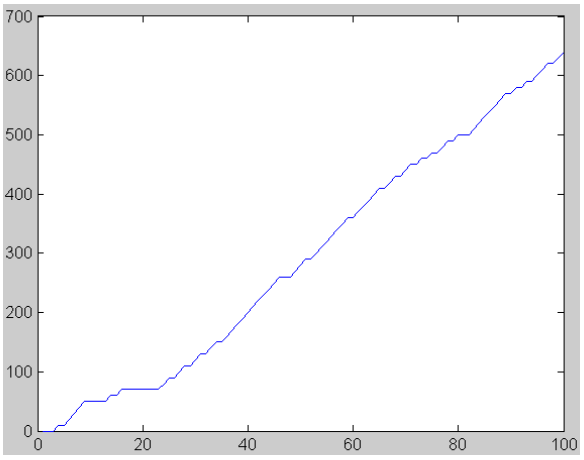
	1	2	3	4	5	6	7	8	9
1	■	■	■	■	■	■	■	■	■
2	■	>	>	>	V	V	V	V	■
3	■	>	^	■	>	V	>	V	■
4	■	^	■	■	>	>	>	V	■
5	■	^	<	■	>	^	■	V	■
6	■	>	>	>	^	^	■	^	■
7	■	■	■	■	■	■	■	■	■

B. (10 points) Plot the accumulated reward for the run, i.e. plot the total amount of reward received so far against the number of episodes, and show the greedy policy with respect to the value function.

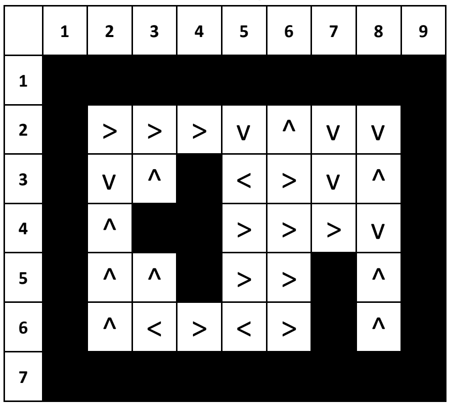
◆ The accumulated reward (x-axis: episode, y-axis: accumulated reward) (It's not always the same):



- From the plot, we can see that, in the first 10 episodes, the agent cannot find a good solution, and after 100 actions it still cannot find the goal. But after the 10th episodes, once the agent finds the goal, it gets reward, and in next few iterations, it can find the goal much faster. Finally, after around 25 episodes, the agent can find the goal almost every time within 100 actions. Sometime if the agent is lucky enough, it can find the goal in the first few iterations, and the accumulated reward is very high, since it can find the goal faster and faster.
- If I set the ε a small value, let's say, 0.001, the accumulated reward is like the following plot:



And its best solution (according to the final Q values):



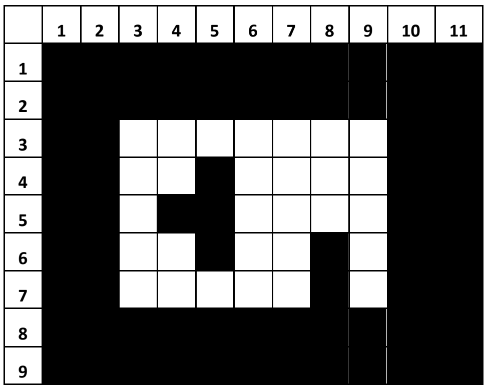
	1	2	3	4	5	6	7	8	9
1	■	■	■	■	■	■	■	■	■
2	■	>	>	>	V	^	V	V	■
3	■	V	^	■	<	>	V	^	■
4	■	^	■	■	>	>	>	V	■
5	■	^	^	■	>	>	■	^	■
6	■	^	<	>	<	>	■	^	■
7	■	■	■	■	■	■	■	■	■

- It is not quite stable, and the final solution is not a good one, since we cannot reach the goal using this path. The reason is that, when the ε is small, the agent rarely explores but only exploits, so the agent didn't try to find a better solution, but only follow the path it firstly find.

- Question 3

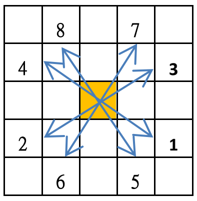
A. (15 points) If instead of moving N,S,E,W, the agent moves like a knight in chess (for example, one step North, then two steps East, in one move), how would the value of the states change (run Q-learning with this new action set)? So the agent now has 8 possible moves. If an action would take it outside the maze, it stays where it is. The probability for moving according to the chosen action is 72% and 4% for each of the other 7 actions.

- See Q4_torus.m; map file: Q2_map.txt; reward file: Q2_reward.txt
- The map is now:

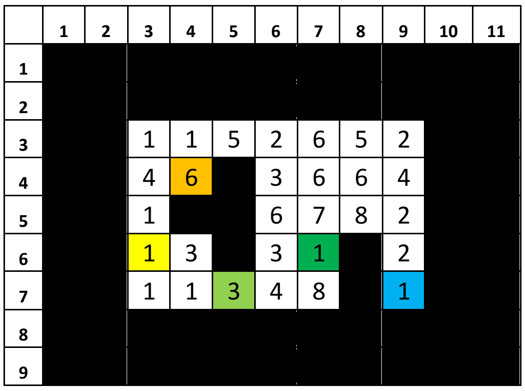


So the agent cannot move outside of the wall.

- Actions:



- Final Q and best path (numbers are the best Q values; colors are the best path: 🟧 → 🟨 → 🟩 → 🟩 → 🟦)

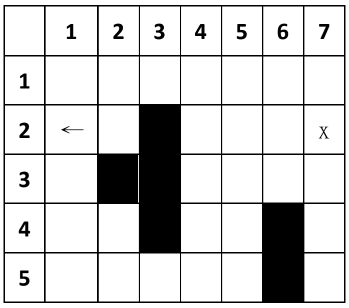


	1	2	3	4	5	6	7	8	9	10	11
1											
2											
3			1	1	5	2	6	5	2		
4			4	6		3	6	6	4		
5			1			6	7	8	2		
6			1	3		3	1		2		
7			1	1	3	4	8		1		
8											
9											

- Question 4:

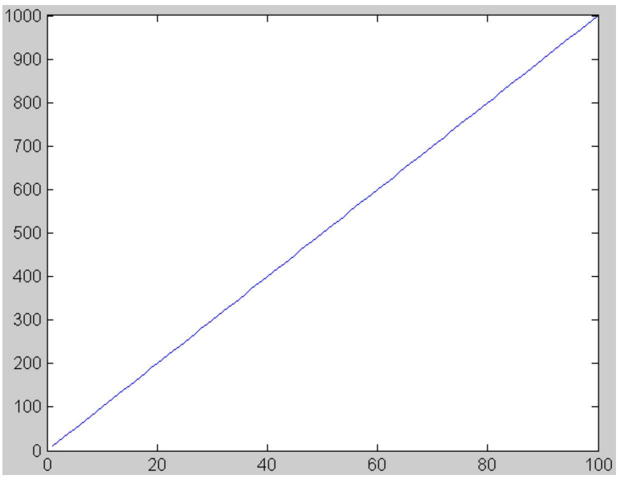
A. Torus with four actions (NESW):

- This question I simply set the map as:



So when the agent we out of the boundary (e.g. State [2, 1] & action West), it will move to another side (e.g. move to State [2, 7]).

- Accumulated reward:



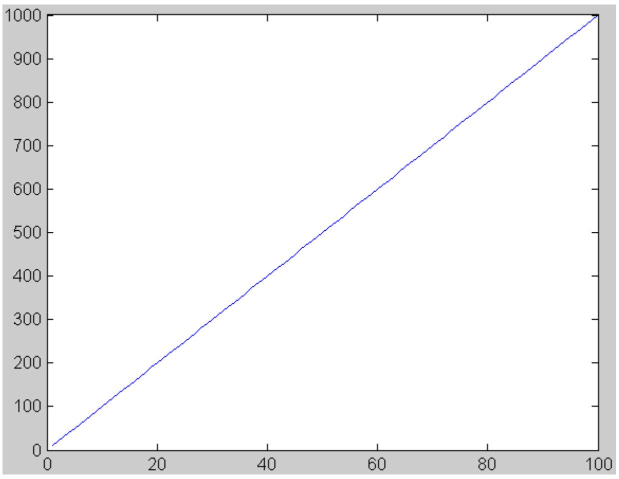
The agent can find the goal within 100 actions in every episode (It's not always the same. Sometimes the final accumulated reward is around 980.)

- Final path (🟧 → 🟨 → 🟩 → 🟩 → 🟦):

	1	2	3	4	5	6	7
1	^	^	^	^	>	<	>
2	^	<	■	^	>	v	>
3	v	■	■	^	^	>	^
4	<	v	■	^	^	■	v
5	<	<	<	<	<	■	^

B. Torus with eight actions (knight):

- Accumulated reward:



The agent can find the goal within 100 actions in every episode.

- Final Q and best path (🟧 → 🟨 → 🟩 → 🟦):

	1	2	3	4	5	6	7
1	3	2	2	4	3	3	1
2	2	1	1	1	1	4	4
3	4	1	1	7	3	5	3
4	3	1	1	5	1	1	1
5	8	2	1	2	8	1	1