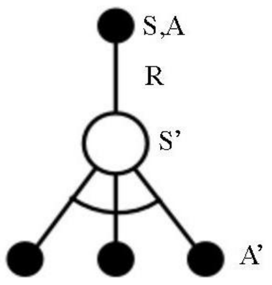
Q-learning Algorithm and Maze Navigation Problems

Q-learning is an off-policy TD control algorithm, which updates the value function based on the equation:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max Q(S_{t+1}, a) - Q(S_t, A_t)]$$

In this case, the learned action-value function Q, directly approximates q*,the optimal action-value function, independent of the policy being followed. The backup diagram for Q-learning is:



The Q-learning algorithm is shown in procedural form is as following:

```
Initialize Q(s; a); 8s 2 S; a 2 A(s); arbitrarily, and Q(terminal-state; ·) = 0
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        Choose A from S using policy derived from Q (e.g., -greedy)
        Take action A, observe R, S0
        Q(S; A) Q(S; A) + αR + γ max a Q(S0, a) - Q(S, A)
        S S0;
    until S is terminal
```

To solve the maze navigation problems, we need construct the maze environment first. The gym library is a collection of test problems — environments — that we can use to work out your reinforcement learning algorithms. These environments have a shared interface, allowing us to write general algorithms. We choose gym toolkit to construct our own maze environment. In gym, individuals interact primarily through the environment in the following ways: step, reset, render, close, seed. Step is the core method that defines the dynamics of the environment, determines the individual's next state, reward information, whether or not the episode is terminated, and some additional information that is not allowed to be used to train the individual. Reset is called before starting an individual's interaction with the environment, this method determines the individual's initial state and possibly some other initialization settings. Seed can set the seed of some random numbers. Override the render method if the interaction between the individual and the environment needs to be animated. Simple UI design can be done using gym's wrapped pyglet methods, which is defined in the rendering.py file.

The gym provides the environment that user can customize the size of the grid, the number of horizontal and vertical grids, the distribution of internal obstacles, and the immediate bonus value of each grid. In the generic grid world environment class UI interface, I use different color settings to indicate different meanings: a grid with a blue border indicates a starting state; a grid with a gold border indicates an ending state, which can be more than one; a black grid indicates an obstacle grid, which an individual cannot normally enter; and other different colors indicate different immediate bonus values for the grid, with bonus Grids with a value of 0 are colored gray, with a bonus value of negative color display bias with red, the smaller the value, the darker the red; grids with a bonus value greater than 0 are biased to display green, the larger the value, the fuller the green; individuals are represented using yellow circles.

The maze is shown as following:

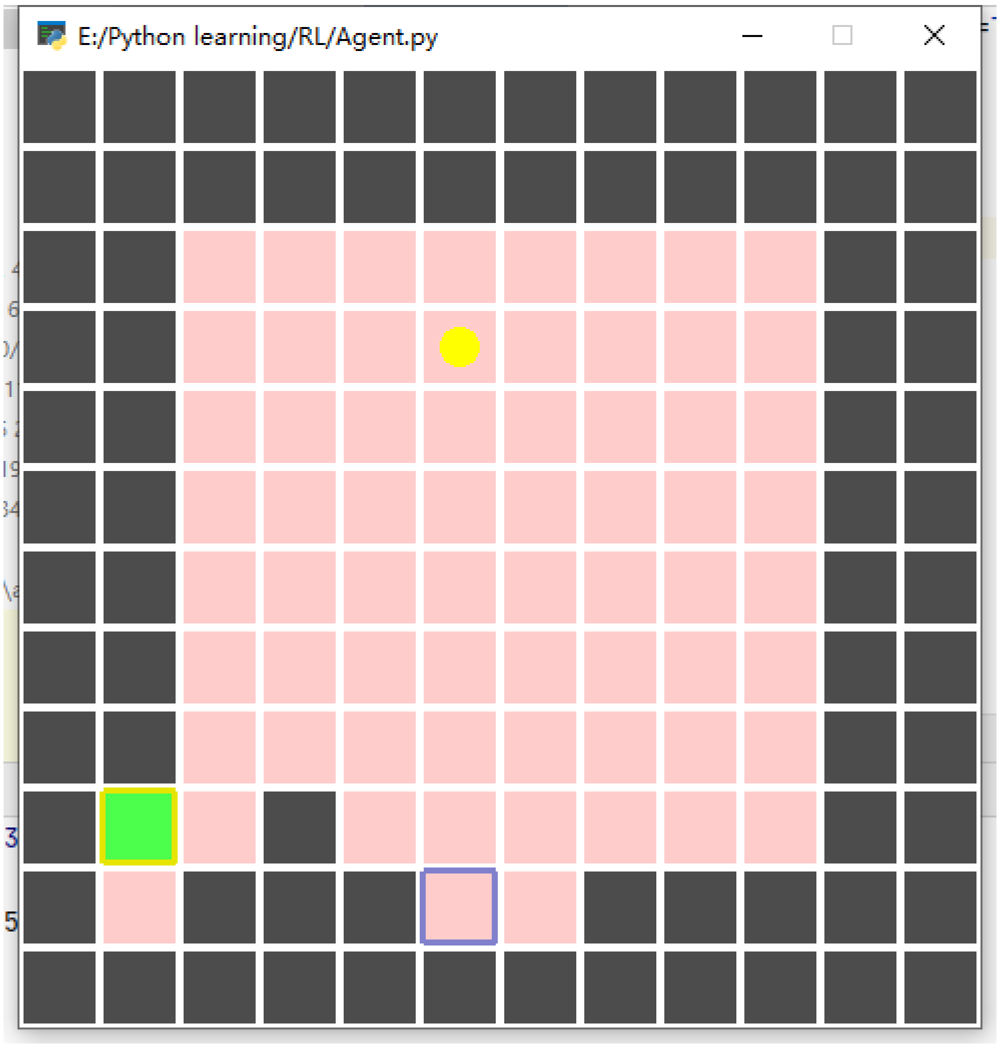


Figure 1. 12* 12 maze environment (blue indicate start point, yellow indicate the goal)

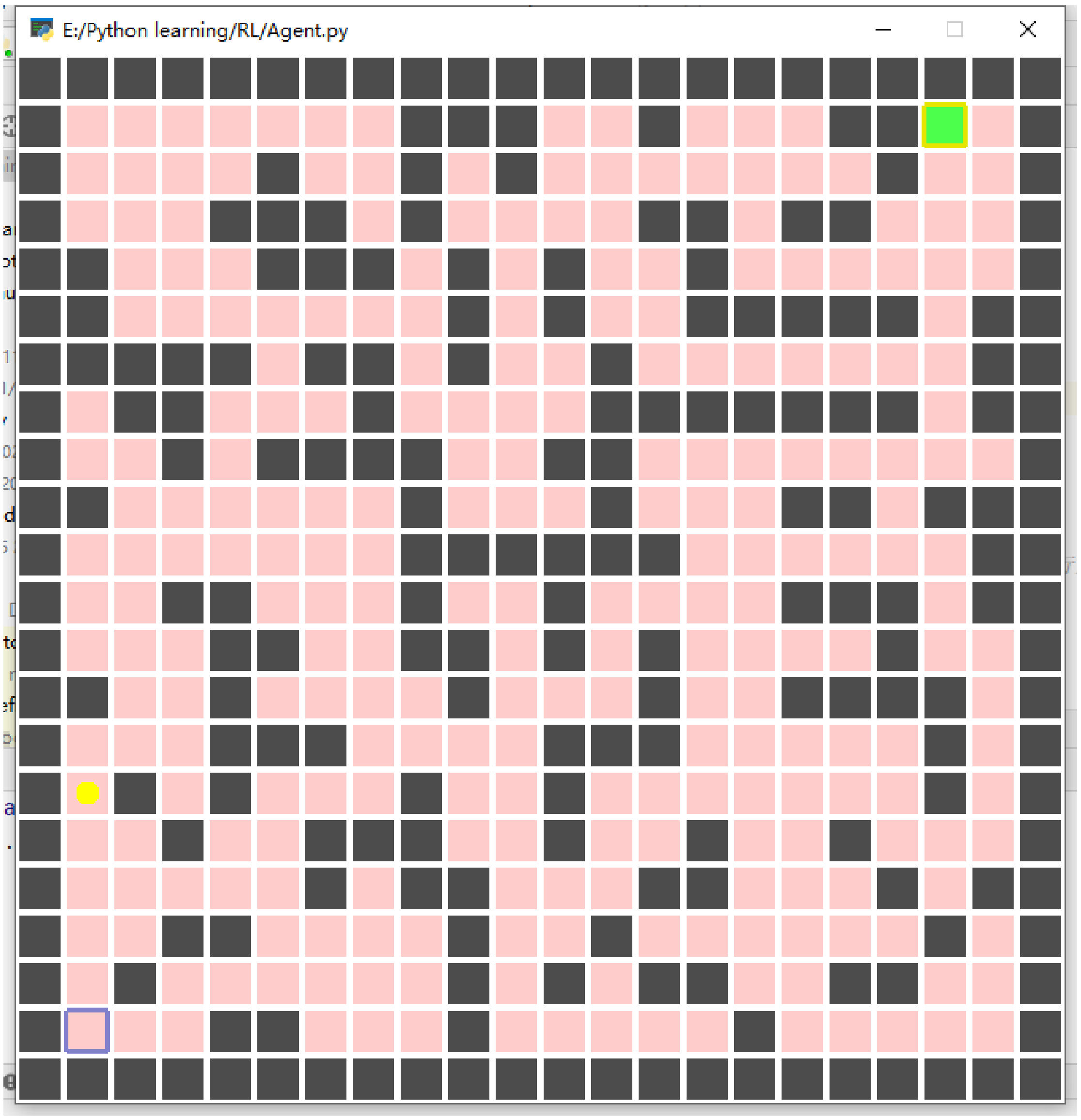


Figure 2. 22* 22 maze environment (blue indicate start point, yellow indicate the goal)

After constructing the environment, it's time for define an agent class which contains the basic parameters and methods. The parameter agent has the environment, action value graph which needing updated during the loop and the state that tell the current position. The Q-learning algorithm is defined in QLearning method that input parameter are discount ratio, learning rate and the max episode number. There is other method that enable us check whether the cycle is terminated and judge the current state.

To balance exploration and exploitation we use epsilon-Greedy to choose between exploration and exploitation randomly. The epsilon-greedy, where epsilon refers to the probability of choosing to explore, exploits most of the time with a small chance of exploring.

The final step of the algorithm is to show the result and visualizations. I record the steps in every episode and draw the steps over episode. From the relationship between episode and steps we can find that as the training goes on, the agent is supposed to be more and more 'smarter' as the steps is getting smaller and finally find the optimal path. The result is as following.

In 12*12 maze navigate case, agent start at point (6,2) and end at (2,3), the result is as following:

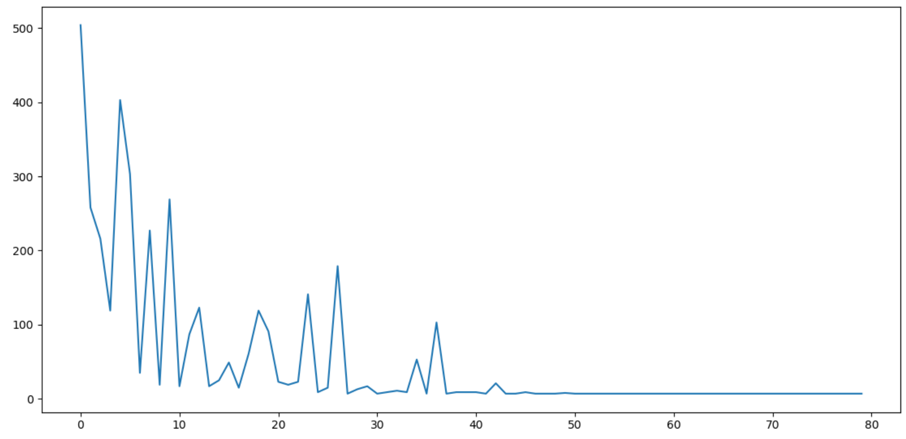


Figure 3. Steps over episodes for 12*12 maze

Best policy is shown in the following table, the goal state has 10 points reward, while other state has -1 reward, and I turn the two dimensional coordinates into one dimension. There are at least 7 steps before the agent arrive at the goal point from the start point.

Step0	s:17	a:2	r:-1	S_:29
Step1	s:29	a:2	r:-1	S_:41
Step2	s:41	a:0	r:-1	S_:40
Step3	s:40	a:0	r:-1	S_:39
Step4	s:39	a:0	r:-1	S_:38
Step5	s:38	a:0	r:-1	S_:26
Step6	s:26	a:1	r:10	S_:25

In 22*22 maze navigate case, there has two start point: (2,2) and (10, 8), the steps number fall gradually as the episode rises.

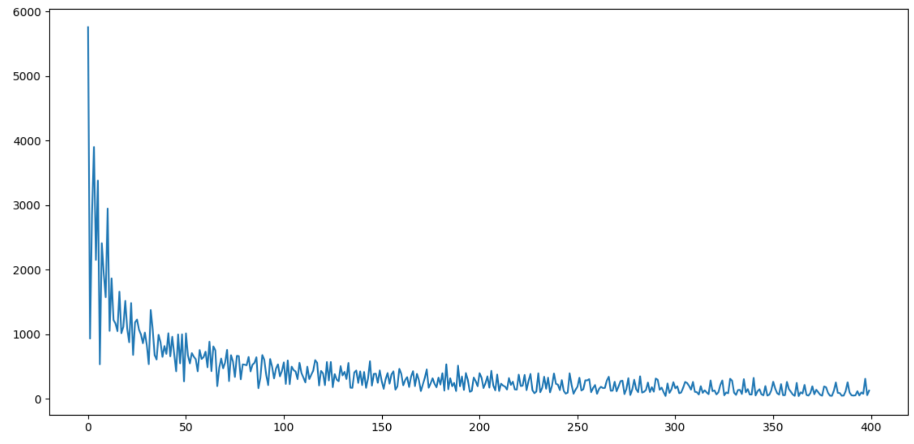


Figure 4. Steps over episodes for 12*12 maze: start at (2,2)

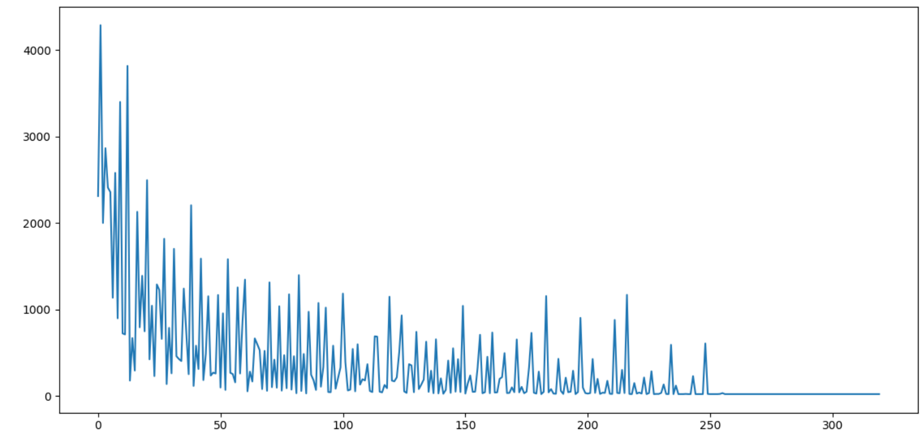


Figure 4. Steps over episodes for 22*22 maze: start at (18,8)

Comparing two different maze, we can realize that it takes 30 steps to find the optimal path in the first case and 250 steps in the second case. What's more, the result also indicates that the Q-learning algorithm can converge to the optimal strategy as long as iterate enough times.