Artificial Intelligence Report

2370060 Chongbin Ren

1. Analysis

The game is described using a grid like the following (for 8x8) with an unknown `problem_id`: in every id there is a different map and initial or goal point. In every episode, the agent controls the movement of a character and reach the goal. Some tiles of the grid are safe but some other tiles are hole. The episode ends when agent reach the goal or fall in a hole. If the agent get to the goal it will get a reward +1 from the environment.

However, there is a stochastic switch which user can specify whether the agent moves in chosen direction. In the grid game, S is the initial state, G is the goal, without falling into the holes, H. The squares marked with F are frozen, which means you can step on them.

Below is the PEAS analysis:

Performance measure

the steps to reach the goal, the possibility to reach the goal

Environment

safe place , hole, goal

Actuators

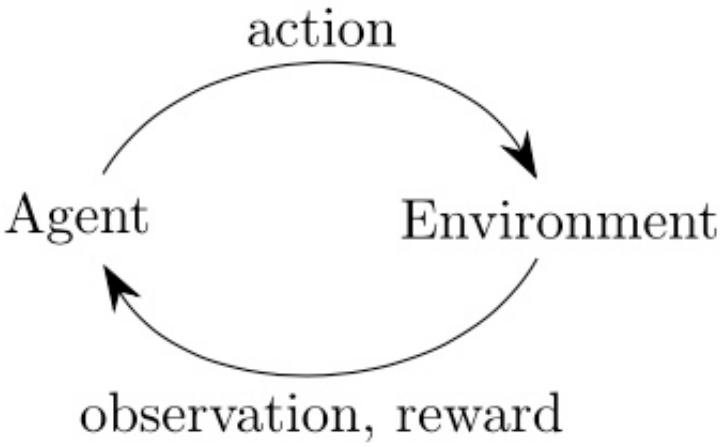
move left, move right, move up, move down

Sensors

keyboard input table

2. Method/Design

The basic process of agent and environment is as below:



1) Random agent

I designed the random agent by using a random action agent. When the environment return a state. The agent will randomly generate a responsible action to this state. When next time this state come, the agent will give the corresponding action to the environment. All the actions are generated randomly.

2) Simple agent

I designed this agent by using the AIMA toolbox to find the best solution path.

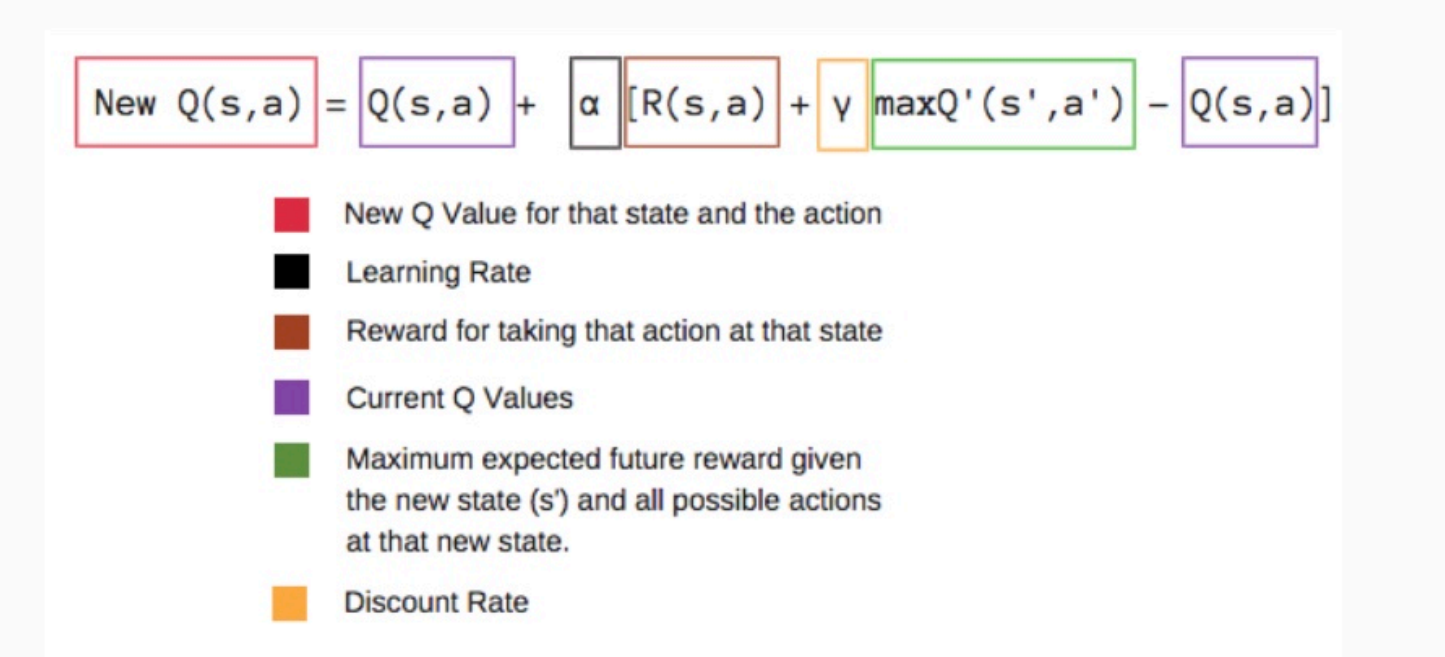
3) Reinforcement learning agent

Q table contains state-action pairs mapping to reward. So, we can construct a map for

different state and actions to reward values during run of algorithm. Its dimension is |states|*|actions|.

We update Q-table using something called the Bellman equation, which states that the expected long-term reward for a given action is equal to the immediate reward from the current action combined with the expected reward from the best future action taken at the following state.

Below is the Q-Learning Equation:



We should use this equation to design Q learning agent. The discount rate variable allows us to decide how important the possible future rewards are compared to the present reward. By updating in this way, the table can slowly obtain accurate measures of the expected future reward for a given action in a given state. In conclusion, this is the design for Q learning agent.

3. Implementation

1) Random agent

The random agent is in `run_random.py`.
The random agent loads the environment, according to the state from the environment, it saves the random action and its corresponding state to a dict. Then it will output the dict to a `pkl` file.

2) Simple agent

The simple agent is in `run_simple.py`.
The simple agent uses AIMA toolbox to calculate the best path for each problem. It will

generate the best path in an output file. Which is like as below:

```
Solution trace:[<Node S_7_4>, <Node S_7_5>, <Node S_6_5>, <Node S_5_5>, <Nod
e S_4_5>, <Node S_4_4>, <Node S_3_4>, <Node S_2_4>, <Node S_1_4>, <Node S_0_
4>, <Node S_0_3>, <Node S_0_2>, <Node S_0_1>, <Node S_0_0>]
```

3) Reinforcement learning agent

The q learning agent is in `run_rl.py`.
The q learning agent will update a Q table each time it runs. This agent will output a Q table in the output file. The output Q table is like as below:

```
Final Values Q-Table:
[[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [-3.71083012e-03 -3.96693773e-03  8.32321622e-05 -6.17296305e-03]
 [ 1.09689150e-04  1.56876953e-04  1.66480622e-04  1.81640584e-04]
 [ 2.51340794e-04  2.74553882e-04  2.85134409e-04  2.13001670e-04]
 [ 5.09621245e-04 -6.67857785e-03 -5.98672579e-03 -7.35307558e-03]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [-6.68042750e-03 -6.45147748e-03  3.75674708e-03 -6.11121362e-03]
 [ 4.61493307e-03  1.39807660e-03  1.50394749e-03  1.22759252e-03]
 [-4.60044015e-03  3.83889509e-05 -6.87800000e-03 -5.42000000e-03]
 [ 1.43912383e-05  2.86029324e-05  4.28036945e-05  8.08827701e-05]
 [-3.40586029e-04 -2.77931638e-04  1.53791185e-05  1.71518720e-04]
 [-5.82618825e-03 -6.11200727e-03 -6.21101168e-03  4.43699890e-04]
 [ 1.26812414e-04  7.32371080e-04  1.28874449e-03  6.13744146e-04]
 [-9.76822259e-03  2.89664143e-03 -7.79666874e-03 -9.94942441e-03]
 [ 1.78361431e-03  1.36798044e-03  6.03449497e-03  1.33967044e-03]
 [ 1.93989914e-03  6.70575616e-03  1.95809570e-03  1.75372719e-03]
 [ 1.29531507e-05 -1.29388319e-05 -3.61670808e-05 -2.83311351e-05]
 [-2.08543223e-04 -5.07471494e-04 -4.25576864e-04 -6.27668561e-05]
```

4. Evaluation

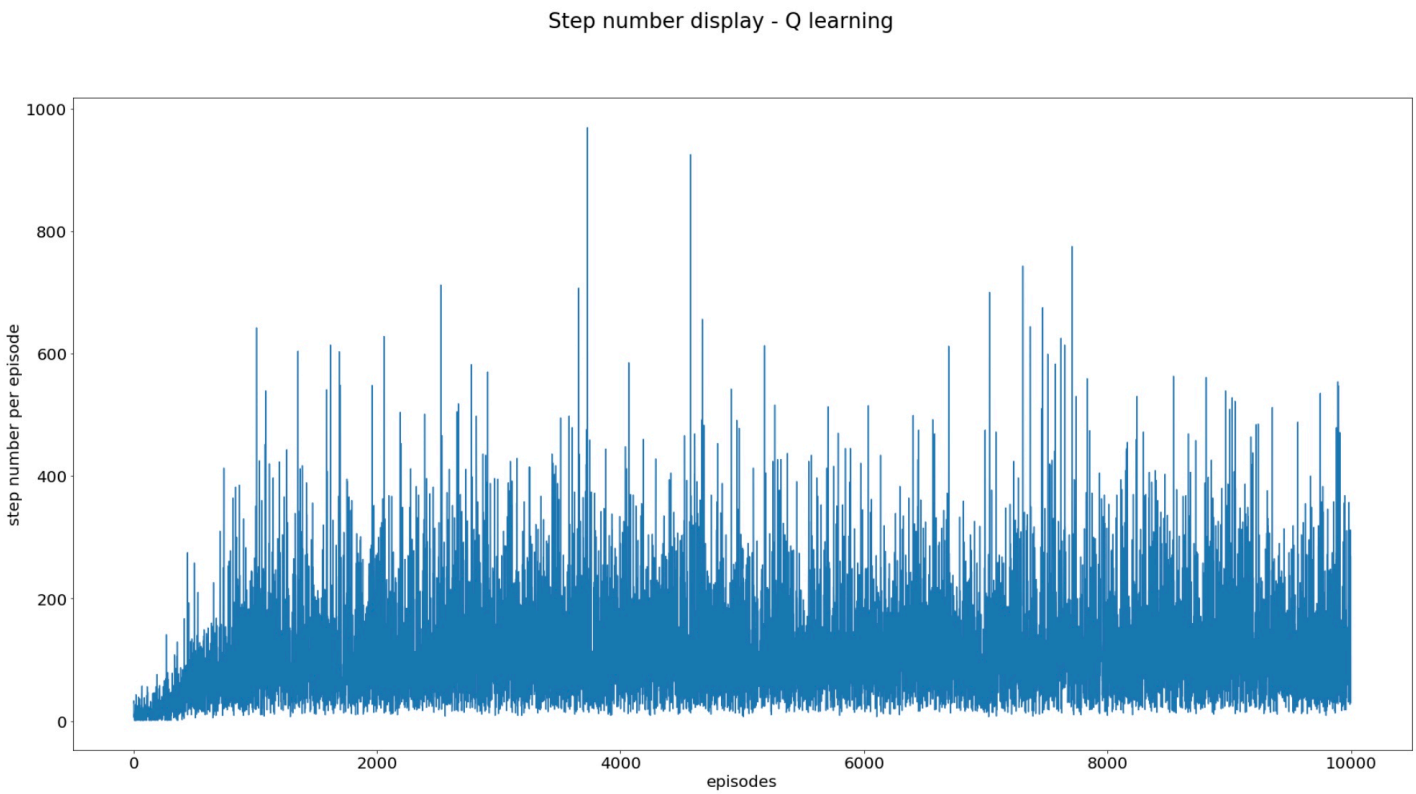
To evaluate the agents I considered different aspects to measure the agents performance.

Q learning performance in training phrase

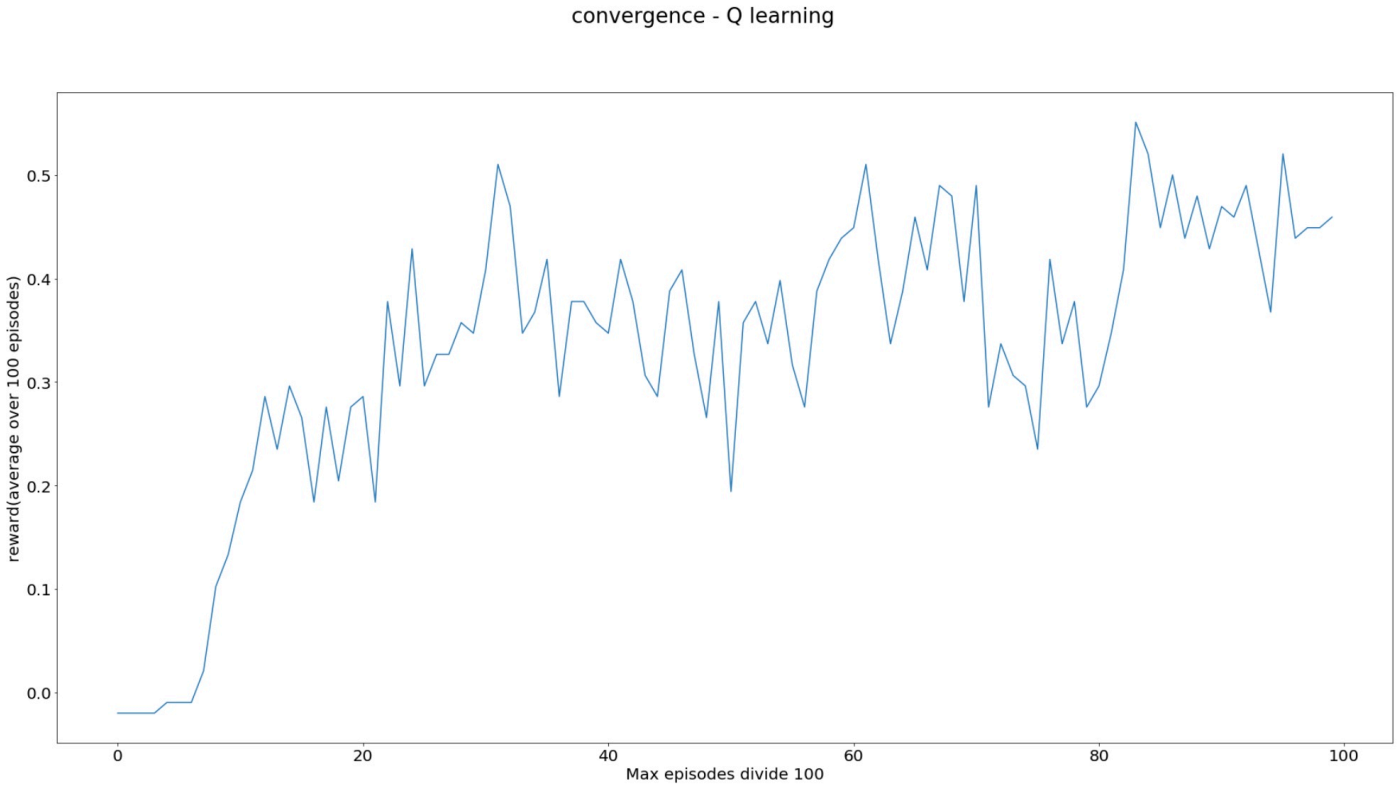
I did not show all problems graph in this phrase, I just show the **problem 6** graph and training 10000 times as an example.
Below is my parameter in training agent:

```
problem_id = 6
reward_hole = -0.02
max_episodes = 10000
```

The step number changing is as below:



The convergence of Q learning is as below:



The max episode is 10000 but I got the sum of reward per 100 episode. So the x axis is [0 - 100]

Performance comparison in different agents

In this evaluation phrase I give the `reward_hole = -0.02` and `max_episodes = 10000` to compare different performance of agents.

1) Reward accumulate compare

I compare random agent and Q learning agent performance by using reward accumulate in different problem id.

Below is the reward accumulate compare in random agent and q learning agent:

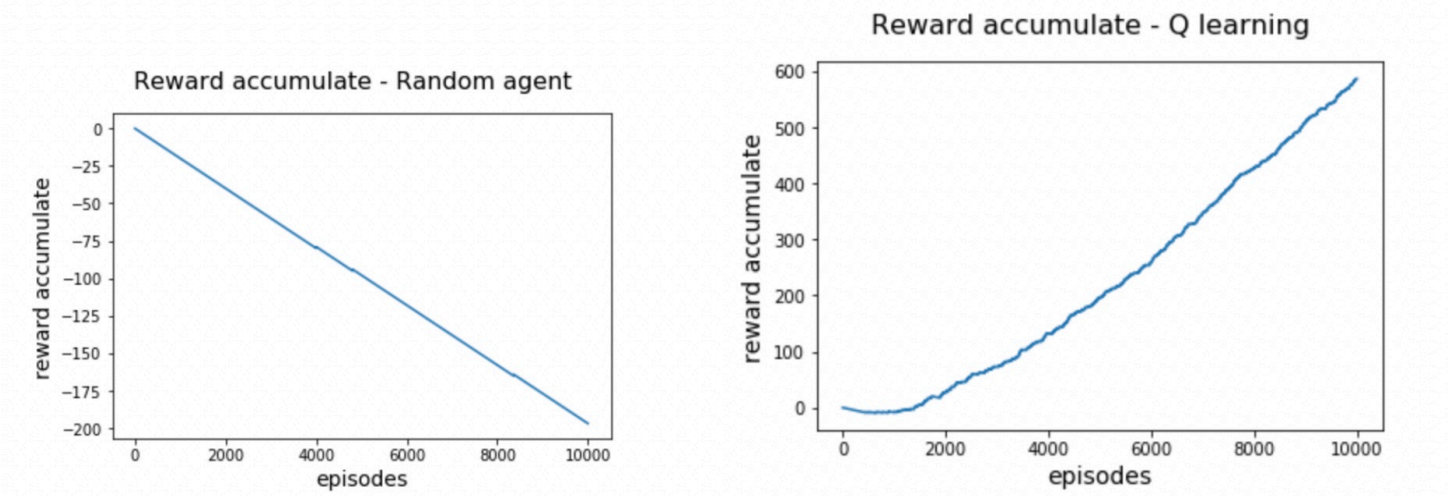
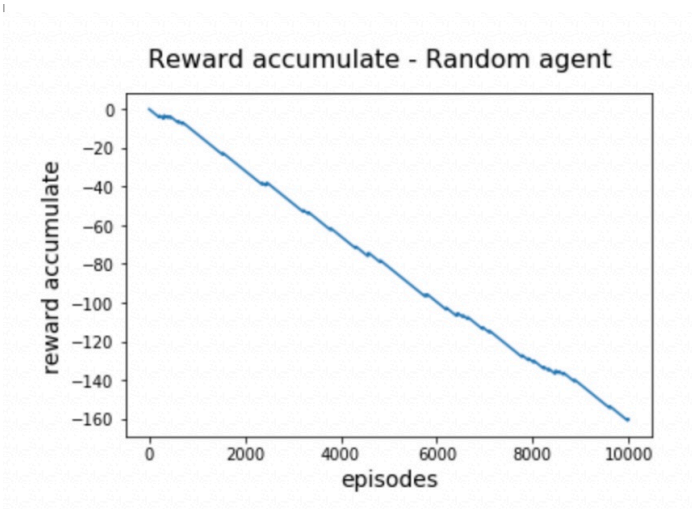


Figure problem_id = 0

Figure problem_id = 1



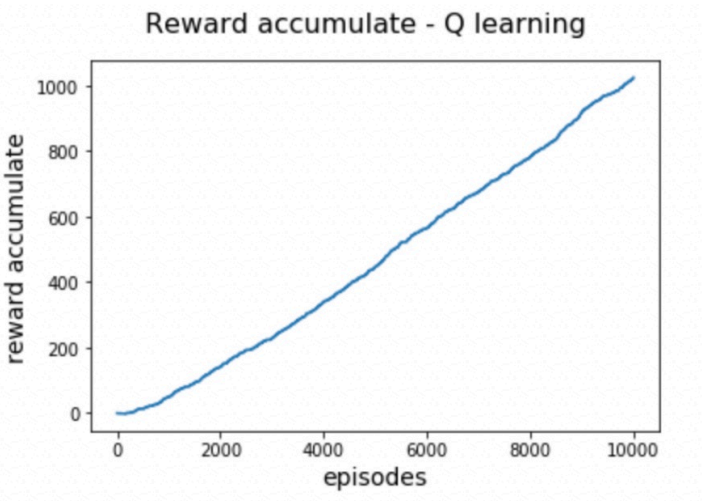
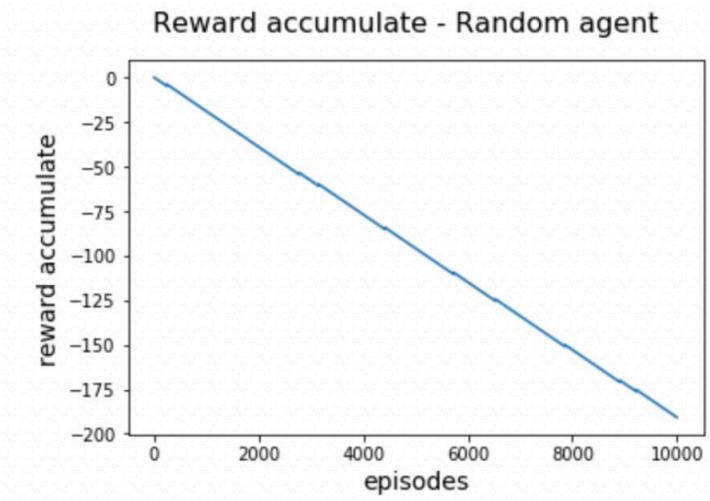


Figure problem_id = 2



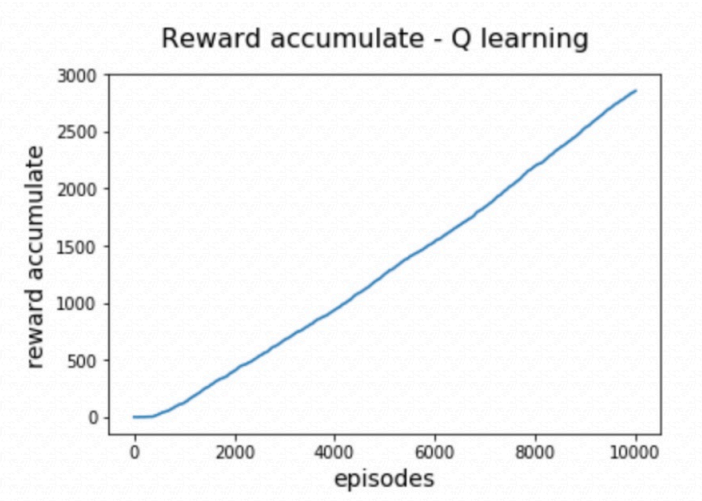
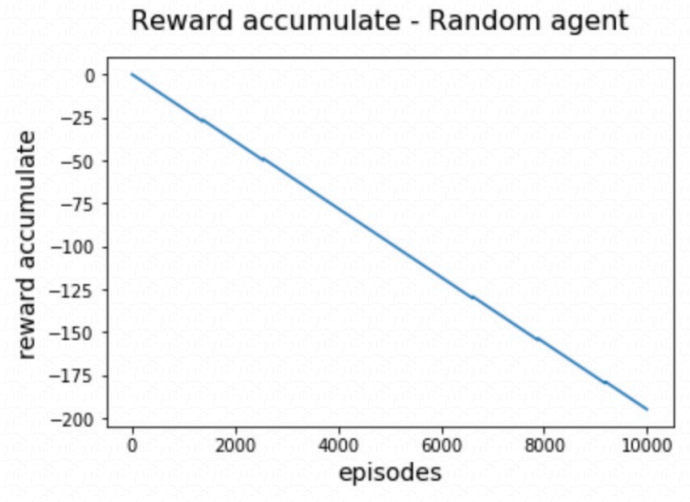


Figure problem_id = 3



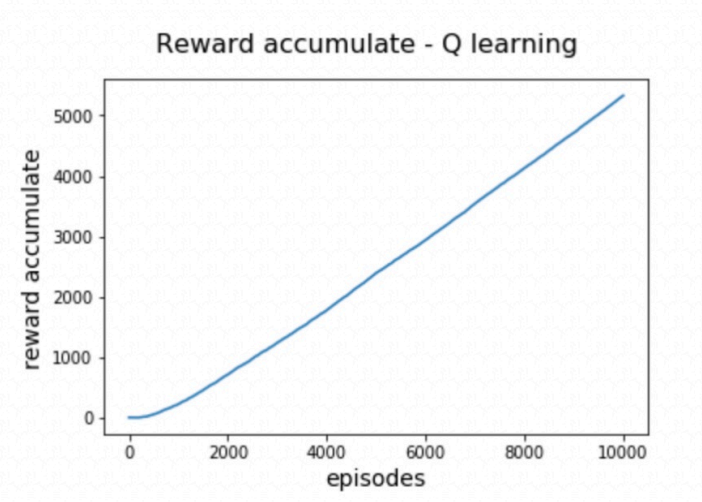


Figure problem_id = 4

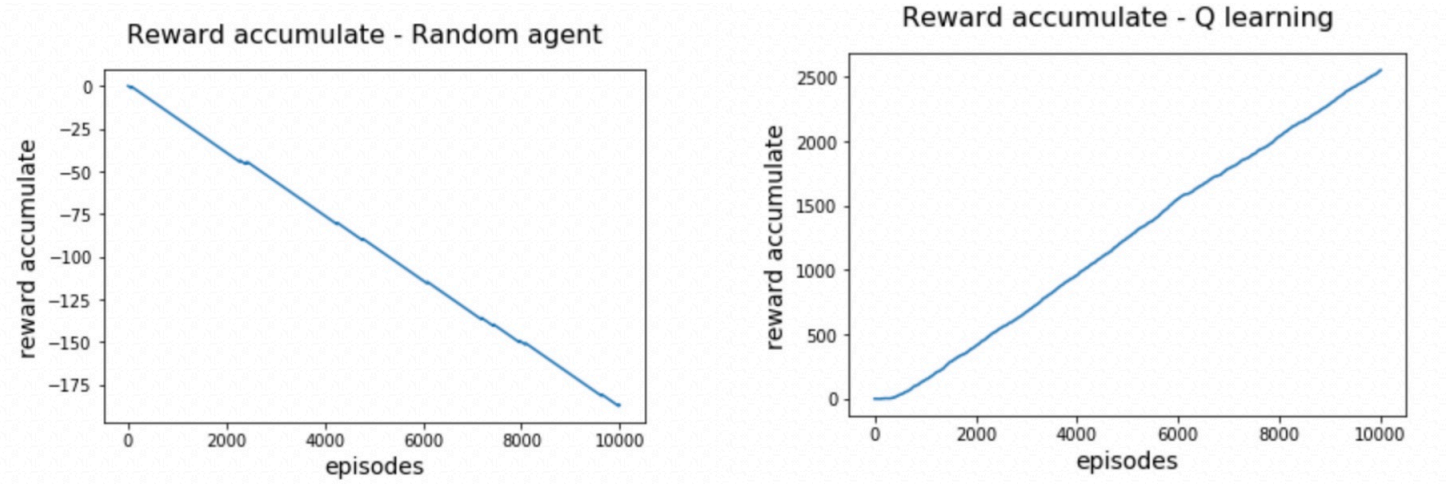


Figure problem_id = 5

Figure problem_id = 6

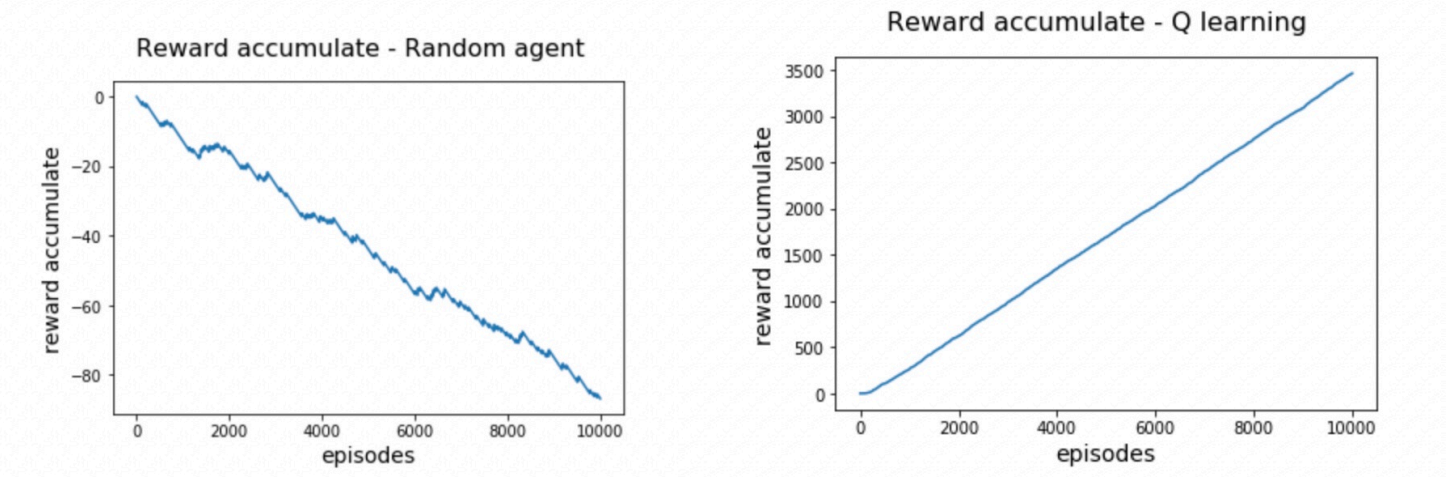


Figure problem_id = 7

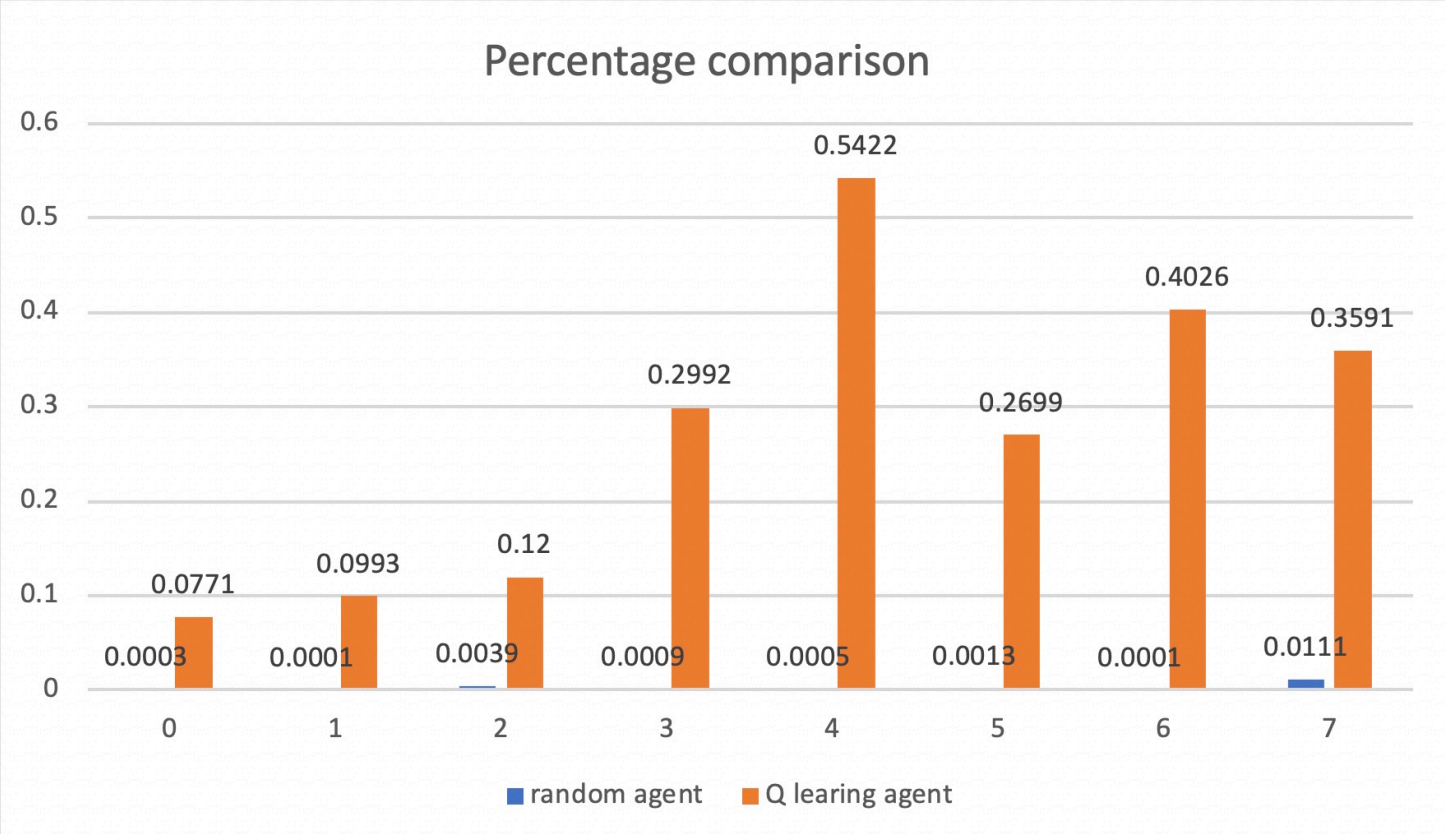
2) Compare of reward total in different agents

I display a table to show the total reward and different agents in different problem id. This

means the times which the agent get the goal per **10000** times.

↓Agent reward / problem id→	0	1	2	3	4	5	6	7
Random agent	3	1	39	9	5	13	1	111
Q-learning agent	771	993	1200	2992	5422	2699	4026	3591

Then I plot the histogram to show the goal percentage in different agents by using the reward total data.



5. Discussion and conclusion

1) In the training phrase of Q learning agent, the agent cannot reach the goal from start. But as the Q table update, Q learning starts to reach more and more goals. The number of step was low from the start because of the hole, but when it learns how to reach the goal, the step number will get higher and finally wave in a range.

2) When comparing different agents performance, we can see that Q learning is much better than random agent. In my 10000 times experiments, the Q agents can reach over 50% goals in a specific problem. However, the random agent is just from 0.01% to 0.39% in different problems.

To conclude, the Q learning agent can be trained by updating Q table and find the way to reach the goal in a high probability.