Assignment two Report

Hang Zheng 520021911347

March 23, 2023

Contents

1 Introduction

In this assignment, I implemented a program to evaluate the random policy under the Gridworld MDP environment without the knowledge of the MDP problem (model free), which mainly consists of three parts as below:

- First Visit Monte-Carlo learning
- Every Visit Monte-Carlo learning
- Temporal-Difference Learning

2 Experimental Principle

In this section, I will give a brief introduction to the principle of the three algorithms we mentioned above.

2.1 Model free algorithm

In reinforcement learning, a mathematical framework called a Markov decision process (MDP) is used to model the problem. Within an MDP, an agent learns to maximize rewards by interacting with an environment.

There are two basic types of algorithms in MDPs: model-based and model-free. The main difference between them is in how they represent and utilize the dynamic model of the MDP.

- Model-based algorithms. The algorithms learn the optimal policy using the complete dynamic model of the environment. The dynamic model describes the transition probabilities and reward function of the interaction between the agent and environment. Once the model is known, dynamic programming or similar methods can be used to calculate the optimal policy. The agent can then execute actions according to the calculated policy and update its value function and policy.

  Model-based algorithms typically require more computational and storage resources to learn and utilize the dynamic model of the environment, but they are usually able to find the optimal policy faster.

- Model-free algorithms. The algorithms learn the optimal policy directly from interacting with the environment, without using the complete dynamic model. Model-free algorithms use sample-based methods to update the agent's value function and policy. These methods collect experience by interacting with the environment and use that experience to update the agent's policy and value function.

  Model-free algorithms typically do not require the complete dynamic model of the environment, so they can learn and execute policies without it, but they usually require more interaction and updates to find the optimal policy.

Model-free algorithms can be divided into two types: value-based algorithms and policy-based algorithms. Value-based algorithms seek to find the optimal policy by learning a state value function, while policy-based algorithms directly learn the optimal policy.

The algorithms we will mention today, like Monte-Carlo learning and TD learning, are the simplest model-free algorithms to solve the MDP problem. Besides, there are some more complicated while more useful methods like Actor-Critic algorithm, Deep Q-Networks and Policy Gradient Method.

2.2 Monte-Carlo learning

The Monte Carlo algorithm is a simulation-based reinforcement learning method used to estimate the value function in MDP.In MDP, the value function is a function defined on states that represents the long-term reward obtainable by taking an action in the current state. The Monte Carlo algorithm is based on the idea that we can estimate the value function by conducting a series of random experiments in the environment, each of which starts from the initial state and takes a series of random actions until the terminal state is reached(called an episode). In each episode, we record the cumulative reward of each state and use these rewards to estimate the value of that state.

Specifically, the Monte Carlo algorithm can be divided into two steps:

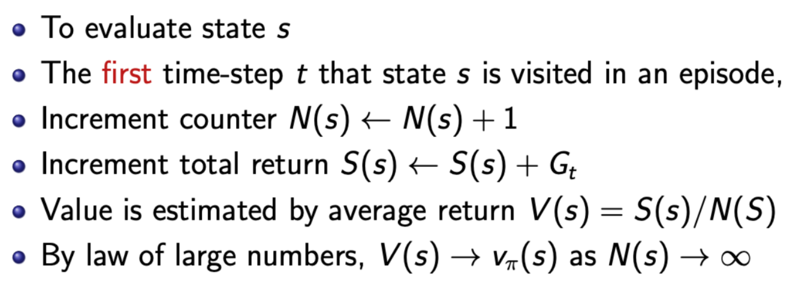
- Sampling: Randomly generate a series of trajectories (or samples) to estimate the value function. Each trajectory starts from the initial state and interacts by taking a series of states and actions until the terminal state is reached.
- Estimation: Estimate the value function based on the sampled trajectories. For each state, we calculate the average reward across all trajectories and use it as the estimate of the value function for that state.

In this assignment, I implemented two types of MC learning, first-visit MC learning and every-visit MC learning. The main idea of the two algorithms are similar, except for the updating process of the value function.

Here I will give a detailed introduction of the algorithms.

2.2.1 First-visit MC learning

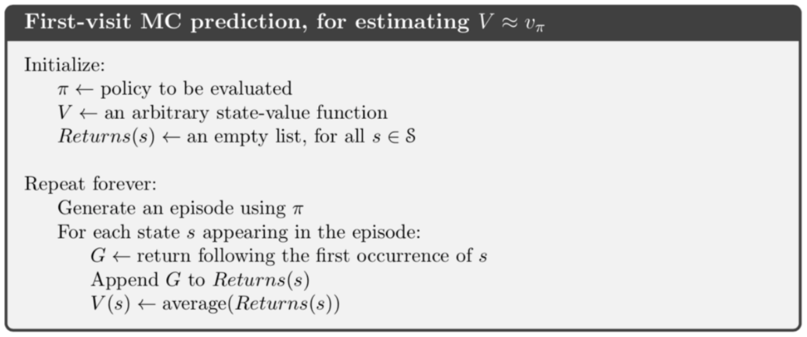
The algorithm can be described as that in figure 1



- To evaluate state s
- The first time-step t that state s is visited in an episode,
- Increment counter $N(s) \leftarrow N(s) + 1$
- Increment total return $S(s) \leftarrow S(s) + G_t$
- Value is estimated by average return $V(s) = S(s)/N(S)$
- By law of large numbers, $V(s) \to v_\pi(s)$ as $N(s) \to \infty$

Figure 1: Description of First-visit MC Learning

The pseudo code of the algorithm is as showed in figure 2.



First-visit MC prediction, for estimating $V \approx v_\pi$

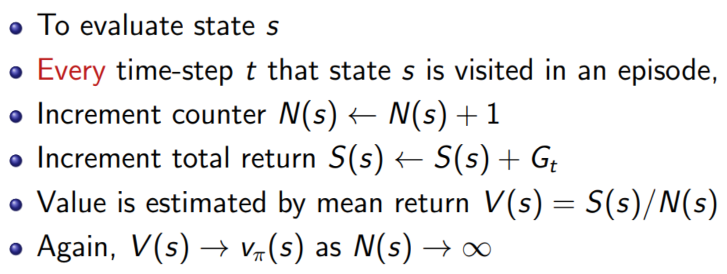
```
Initialize:
    $\pi \leftarrow$ policy to be evaluated
    $V \leftarrow$ an arbitrary state-value function
    $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Repeat forever:
    Generate an episode using $\pi$
    For each state $s$ appearing in the episode:
        $G \leftarrow$ return following the first occurrence of $s$
        Append $G$ to $Returns(s)$
        $V(s) \leftarrow$ average($Returns(s)$)
```

Figure 2: Pseudo code of First-visit MC Learning

2.2.2 Every-visit MC learning

The algorithm can be described as that in figure 3



- To evaluate state s
- Every time-step t that state s is visited in an episode,
- Increment counter $N(s) \leftarrow N(s) + 1$
- Increment total return $S(s) \leftarrow S(s) + G_t$
- Value is estimated by mean return $V(s) = S(s)/N(s)$
- Again, $V(s) \rightarrow v_\pi(s)$ as $N(s) \rightarrow \infty$

Figure 3: Description of Every-visit MC Learning

2.3 Temporal-Difference Learning

TD learning algorithm is a model-free algorithm that learns how to make optimal decisions in the current state by observing previous states and reward signals. Specifically, TD learning algorithm updates the value of the current state by comparing the current estimate and the estimate of the next state.

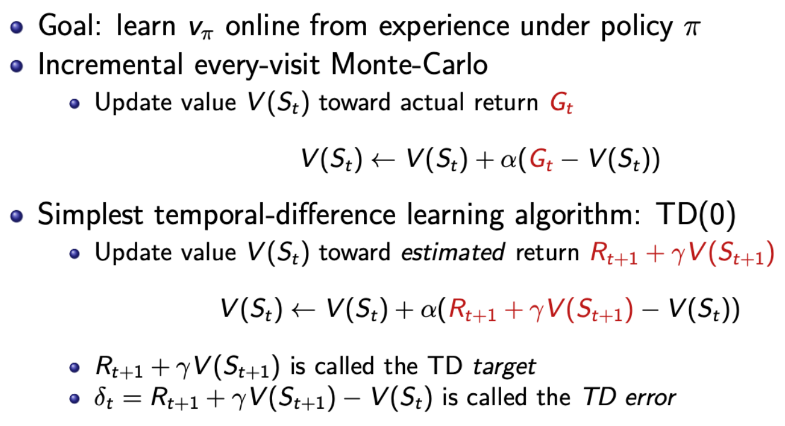
In TD learning algorithm, the value of each state is represented as $V(s)$, where s is the state. $V(s)$ represents the expected long-term reward that the agent can obtain in state s. The update rule of TD learning algorithm for updating $V(s)$ is:

$V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$

where α is the learning rate, r is the reward signal in the current state, γ is the discount factor, and s' is the next state.

The core idea of this update rule is to consider the future reward by multiplying the estimate of the next state by the discount factor γ, and then adding the current reward signal r to estimate the value of the current state. The difference between this estimate and the current estimate is used to update the value of the current state.By iteratively updating the value of each state, TD learning algorithm can learn the optimal policy and solve the MDP problem.

The algorithm can be described as that in figure 4



- Goal: learn v_π online from experience under policy π
- Incremental every-visit Monte-Carlo
  - Update value $V(S_t)$ toward actual return G_t

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$

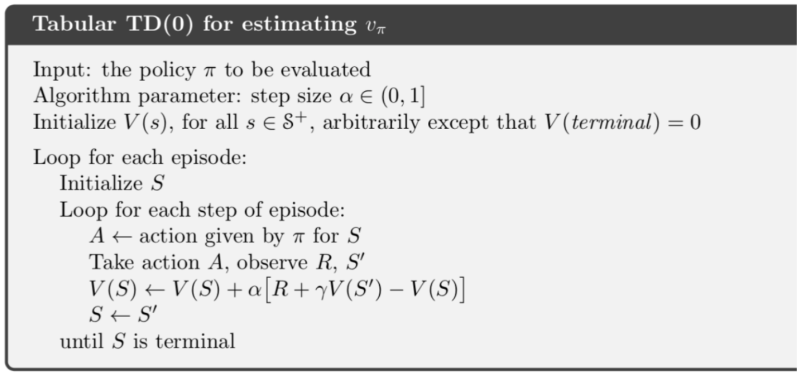
- Simplest temporal-difference learning algorithm: TD(0)
  - Update value $V(S_t)$ toward *estimated* return $R_{t+1} + \gamma V(S_{t+1})$

$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

  - $R_{t+1} + \gamma V(S_{t+1})$ is called the TD *target*
  - $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called the *TD error*

Figure 4: Description of TD Learning

The pseudo code of the algorithm is as showed in figure 5.



```
Tabular TD(0) for estimating $v_\pi$

Input: the policy $\pi$ to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop for each episode:
    Initialize $S$
    Loop for each step of episode:
        $A \leftarrow$ action given by $\pi$ for $S$
        Take action $A$, observe $R$, $S'$
        $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
        $S \leftarrow S'$
    until $S$ is terminal
```

Figure 5: Pseudo code of TD Learning

3 Experimental Content

The directory structure of my code is like:

- main.py: The main file to be run, in which I create an instance of the MDP_Agent class I implemented to complete the task of this assignment.
- algo.py: I implement a class named MDP_Agent to provide the functionality of the three algorithms we mentioned in the last section and a extra class named GridWorld to instantiate a Gridworld environment.

Here I will list the methods of each classes and the further details will not be explained here (Please refer to the detailed comments I added in the source code)

3.1 GridWorld

The methods of Class GridWorld are showed as below:

- setTerminal(x,y): set the state (x,y) as a terminal state.
- isTerminal(state): return a flag to indicate whether a state is a terminal state.
- getReward(state, action, next_state): return the reward from state to next_state taking action.
- getStates(): return all the states of current mdp environment.
- getPossibleActions(state): return a list of actions that the agent able to take at current state.
- isAllowed(x,y): return a flag to indicate whether a state (x,y) is legal in current mdp environment.
- getTransition(): return the transition between states and corresponding probability.
- getRandomState(): return a random non-terminal state.

3.2 MDP_Agent

The methods of Class MDP_Agent are showed as below:

- runFirstVisit_MC(e): run the first-visit MC learning method with minimum episodes e.
- runEveryVisit_MC(e): run the every-visit MC learning method with minimum episodes e.
- runTD_learning(e, alpha): run the TD learning method with minimum episodes e and a step size alpha.
- setEpsilon(e): set the epsilon value as e, where epsilon is the accuracy flag to indicate whether the algorithm has converged.

- extractPolicy(): extract policy out of values of the states.
- printValuesAndPolicy(): print the values and policies of the states.

3.3 Program Running

To run the codes, you only need to type in the command like:
python main.py
and the program will run and save the running result in a file named ”output.txt”
Further more, the program are implemented to takes several arguments and you are able to tune the hyper-parameters flexibly.

- ’-e’,’–episode’: the minimum episodes to run the three methods.
- ’-d’,’–discount’: the discounting factor.
- ’–mc_epsilon’: the epsilon in MC Learning algorithm.
- ’–td_epsilon’: the epsilon in TD Learning algorithm.
- ’-o’,’–output_file’: the output file path to save the results.
- ’-l’,’–path_length’: the length limitation to the random generated state path.
- ’-s’,’–successful_count’: the successful counts to make the algorithm more stable, where a successful count is defined as an episode where all the value updating difference are smaller than the threshold.
- ’-a’,’–alpha’: the step size in TD learning.

4 Experimental Result

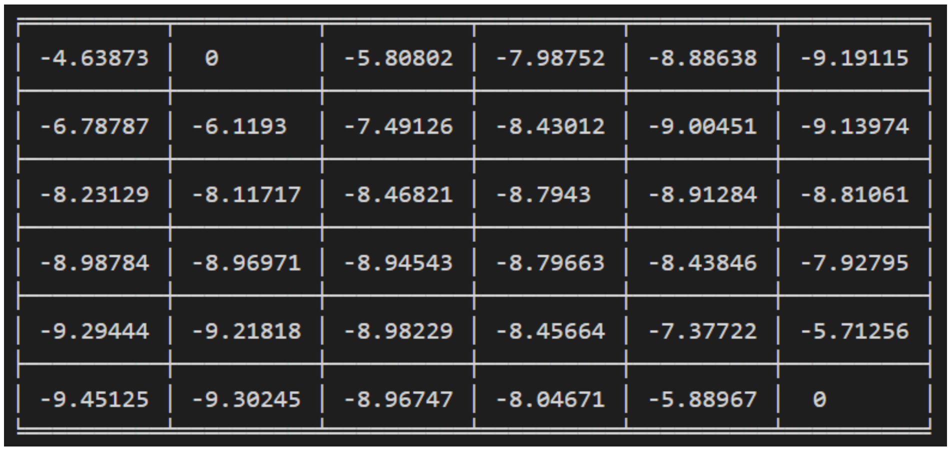
4.1 Basical experiment

The result of the experiment is by default redirected and saved in the file ”output/output.txt”, here I will showed the result as well.

The arguments are set as: $discount\ factor:\ 1.0$, $path\ length\ limit:\ 1000$, $successful\ counts:\ 5$

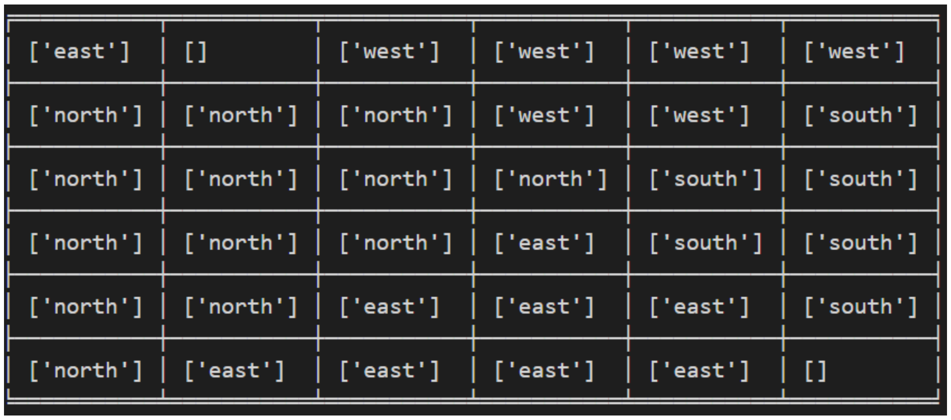
4.1.1 First-visit MC learning

The result of conducting First-visit MC learning on random policies is showed as in figure 6 and figure 7



```
| -4.63873 | 0        | -5.80802 | -7.98752 | -8.88638 | -9.19115 |
| -6.78787 | -6.1193  | -7.49126 | -8.43012 | -9.00451 | -9.13974 |
| -8.23129 | -8.11717 | -8.46821 | -8.7943  | -8.91284 | -8.81061 |
| -8.98784 | -8.96971 | -8.94543 | -8.79663 | -8.43846 | -7.92795 |
| -9.29444 | -9.21818 | -8.98229 | -8.45664 | -7.37722 | -5.71256 |
| -9.45125 | -9.30245 | -8.96747 | -8.04671 | -5.88967 | 0        |
```

Figure 6: Values in First-visit MC learning

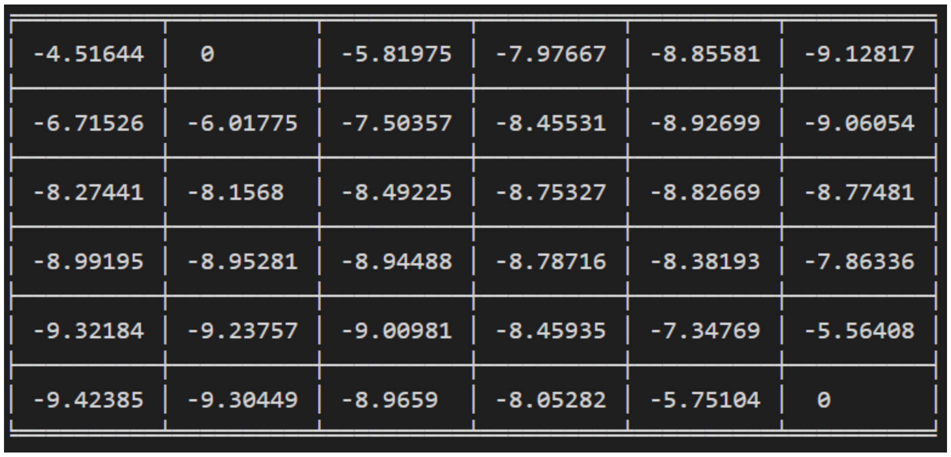


```
| ['east']  | []        | ['west']  | ['west']  | ['west']  | ['west']  |
| ['north'] | ['north'] | ['north'] | ['west']  | ['west']  | ['south'] |
| ['north'] | ['north'] | ['north'] | ['north'] | ['south'] | ['south'] |
| ['north'] | ['north'] | ['north'] | ['east']  | ['south'] | ['south'] |
| ['north'] | ['north'] | ['east']  | ['east']  | ['east']  | ['south'] |
| ['north'] | ['east']  | ['east']  | ['east']  | ['east']  | []        |
```

Figure 7: Policies in First-visit MC learning

4.1.2 Every-visit MC learning

The result of Every-visit MC learning is showed as in figure 8 and figure 9



```
| -4.51644 |  0       | -5.81975 | -7.97667 | -8.85581 | -9.12817 |
| -6.71526 | -6.01775 | -7.50357 | -8.45531 | -8.92699 | -9.06054 |
| -8.27441 | -8.1568  | -8.49225 | -8.75327 | -8.82669 | -8.77481 |
| -8.99195 | -8.95281 | -8.94488 | -8.78716 | -8.38193 | -7.86336 |
| -9.32184 | -9.23757 | -9.00981 | -8.45935 | -7.34769 | -5.56408 |
| -9.42385 | -9.30449 | -8.9659  | -8.05282 | -5.75104 |  0       |
```

Figure 8: Values in Every-visit MC learning

4.1.3 Temporal-Difference Learning

The result of Temporal-Difference Learning is showed as in figure 10 and figure 11

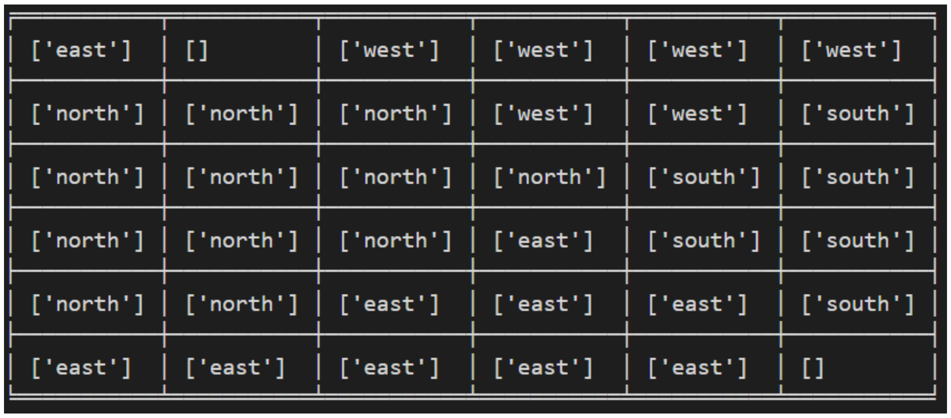
4.1.4 Analysis

With the experimental results showed in the last several subsections, we can see that:

All the three methods converged to the same values and policies, and **the policies are all optimal**, which indicates that the three methods are all useful.

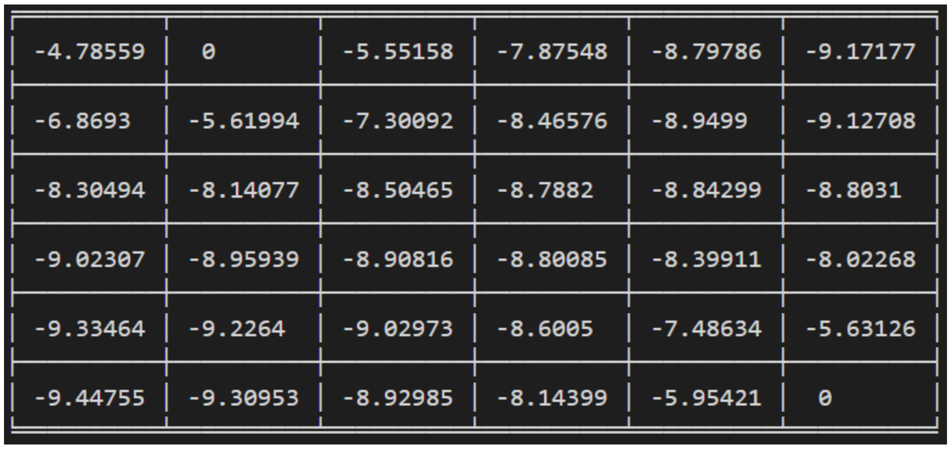
4.2 Experiment on state path length limitation

Due to the stochasticity of the policy, the length of the generated paths in the Gridworld environment is of high uncertainty, and there may exist paths with very long lengths (the agent gets trapped in a local area and unable to reach the terminal state), which can result in rewards with a large absolute value and cause a significant impact on the value of the state, leading to value instability and affecting convergence. Therefore, it may be necessary to consider limiting the length of the path to address this issue.



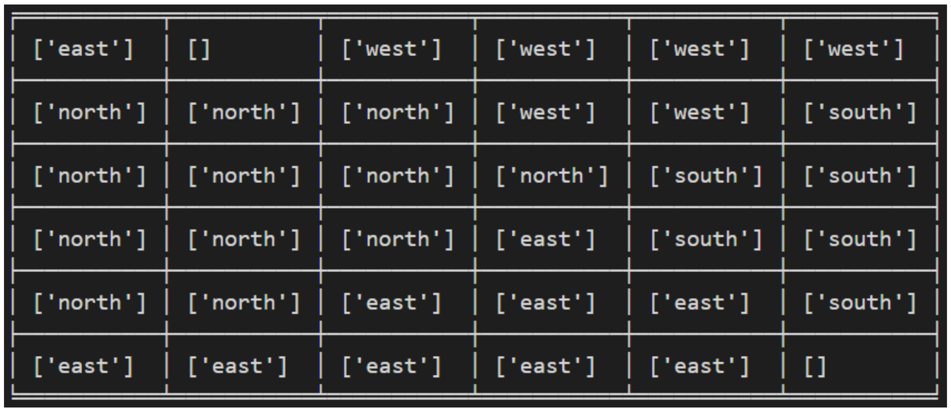
```
| ['east']  | []        | ['west']  | ['west']  | ['west']  | ['west']  |
| ['north'] | ['north'] | ['north'] | ['west']  | ['west']  | ['south'] |
| ['north'] | ['north'] | ['north'] | ['north'] | ['south'] | ['south'] |
| ['north'] | ['north'] | ['north'] | ['east']  | ['south'] | ['south'] |
| ['north'] | ['north'] | ['east']  | ['east']  | ['east']  | ['south'] |
| ['east']  | ['east']  | ['east']  | ['east']  | ['east']  | []        |
```

Figure 9: Policies in Every-visit MC learning



```
| -4.78559 |  0       | -5.55158 | -7.87548 | -8.79786 | -9.17177 |
| -6.8693  | -5.61994 | -7.30092 | -8.46576 | -8.9499  | -9.12708 |
| -8.30494 | -8.14077 | -8.50465 | -8.7882  | -8.84299 | -8.8031  |
| -9.02307 | -8.95939 | -8.90816 | -8.80085 | -8.39911 | -8.02268 |
| -9.33464 | -9.2264  | -9.02973 | -8.6005  | -7.48634 | -5.63126 |
| -9.44755 | -9.30953 | -8.92985 | -8.14399 | -5.95421 |  0       |
```

Figure 10: Values in Temporal-Difference Learning



```
| ['east']  | []        | ['west']  | ['west']  | ['west']  | ['west']  |
| ['north'] | ['north'] | ['north'] | ['west']  | ['west']  | ['south'] |
| ['north'] | ['north'] | ['north'] | ['north'] | ['south'] | ['south'] |
| ['north'] | ['north'] | ['north'] | ['east']  | ['south'] | ['south'] |
| ['north'] | ['north'] | ['east']  | ['east']  | ['east']  | ['south'] |
| ['east']  | ['east']  | ['east']  | ['east']  | ['east']  | []        |
```

Figure 11: Policies in Temporal-Difference Learning

In my program, I've designed an argument to limit the state path length. And furthermore, I conducted an experiment on this factor and the result are showed in table 1. Noticed that the other arguments setting are:

$discount\,factor:\ 0.9,\ mc\_epsilon:\ 1e{-4},\ \ td\_epsilon:\ 0.1,\ successful\,counts:\ 5,\ minimumepisodes:$ 10000

Algorithm	Path Length Limit	Converging Episodes	Converging Time Taken(s)	Value of State (0,0)
First-visit MC learning	30	<10000	4.8037755489349365	-2.96952
	100	<10000	4.941810846328735	-4.02509
	1000	19894	7.054937362670898	-4.59541
Every-visit MC learning	30	10515	9.64867877960205	-3.1545
	100	39786	29.17559576034546	-4.06758
	1000	54607	24.855352878570557	-4.60696
Temporal-Difference Learning	30	<10000	6.746103286743164	-2.88148
	100	<10000	4.441248178482056	-4.02408
	1000	<10000	5.0685341358184814	-4.35718

Table 1: Results of Experiment on State Path Length Limitation

From the results, we can see that:

- First-visit MC method converges faster than the very-visit MC method(but generally less stable).
- The stricter the length limitation, the faster the algorithms converge and the greater the values of the states will be (take state(0,0) as a flag). This really makes sense since that with a stricter limit on the state path length, the paths are less likely to grow too long to lead to great reward for some states, which will help the values of the states converge more stable and averagely increase the absolute values of the states.

4.3 Experiment on discounting factor

In MDP solving algorithm, the discounting factor is a parameter between 0 and 1, typically denoted by the symbol γ, which represents the degree to which future rewards are discounted. Specifically, if γ is close to 0, the influence of future rewards will be greatly weakened, while if γ is close to 1, the influence of future rewards will remain strong.

In my program, I've designed an argument to set the discounting factor. And furthermore, I conducted an experiment on this factor and the result are showed in table 2 and table 3. Noticed that the other arguments setting are:

$lengthlimit:\ 1000,\ mc\_epsilon:\ 1e-4,\ \ td\_epsilon:\ 0.1,\ minimumepisodes:\ 10000$

Discounting Factor	Converging Episodes First-visit MC	Converging Time Taken(s) First-visit MC	Converging Episodes Every-visit MC	Converging Time Taken(s) Every-visit MC
0.9	15682	6.65680193901062	55329	23.0633442401886
	25386	10.635751008987427	55551	26.554508447647095
	<10000	3.8353474140167236	55436	40.1331102848053
	19894	7.054937362670898	54607	24.855352878570557
1.0	328182	144.81438422203064	318453	240.97880482673645
	329323	250.3868749141693	322159	229.33462142944336
	350274	135.0753219127655	324979	211.63087105751038
	268868	94.3323335647583	326635	149.46089363098145

Table 2: Convergence Speed Comparison between different Discounting Factor

Discounting Factor	Value of state(0,0) First-visit MC	Value of state(0,0) Every-visit MC	Value of state(0,0) TD Learning
0.9	-4.53952	-4.59816	-4.58583
	-4.65126	-4.60957	-4.67646
	-4.62349	-4.58533	-4.68286
	-4.59541	-4.60696	-4.35718
1.0	-18.5522	-18.1054	-17.5325
	-18.1217	-18.2136	-19.1363
	-18.0943	-18.2572	-15.6454
	-18.154	-18.4631	-16.3987

Table 3: Convergence Accuracy Comparison between different Discounting Factor

From the two tables, we can see that:

- A smaller discounting factor can significantly speed up the convergence of the algorithms. The reason of this is that the value of each state will end up to be with a smaller absolute value and thus easier to satisfy the threshold, which makes the algorithm converges faster.
- A smaller discounting factor makes the algorithm more stable. From table 3, we can see that the algorithms are easier to converge and gets more stable state values with a smaller discounting factor, especially the TD Learning method. The reason for this may be that a smaller discounting factor makes the agent focus more on the recent reward while caring less about the future reward, which will mitigate the effect of long state paths(a long state path makes the algorithm less stable as we talked about before).

5 Conclusion

During the process of this experiment, I first systematically learned the relevant knowledge of two basic kinds of model-free MDP algorithms, Monte-Carlo learning and Temporal-Difference Learning, on the basis of the course content, and then implemented the two algorithms on my own to better understanding the implementation details.

Furthermore, just as in the last assignment, based on the first simple version, I further implement the code so that it can be adapted to different Gridworld problems, and even different more complex MDP problems (only if it implements specific methods such as getStates(), etc.) In the future, I may further optimized the logic of the code and test it on more complex MDP problems.