Reinforcement Learning Project2

Hui Liu(hliu58@vols.utk.edu)

Yang Xu(yxu71@vols.utk.edu)

1. Introduction

A mouse is searching for a cheese which is placed somewhere in a grid of size $d \times d$. This grid is set up with m traps that are also unknown to this mouse. The mouse moves north, south, west, or east each time, and it simply returns to the same position if it takes an action that takes it out of the board. The mouse always starts from the top left corner and searches for its goal. Keep mind, for each task, cheese and traps are fixed. In every searching, mouse ends with getting the cheese or being trapped. In this project, we applied TD learning with n-step SARSA and Monte Carlo method to guide the mouse finding the cheese and avoiding traps.

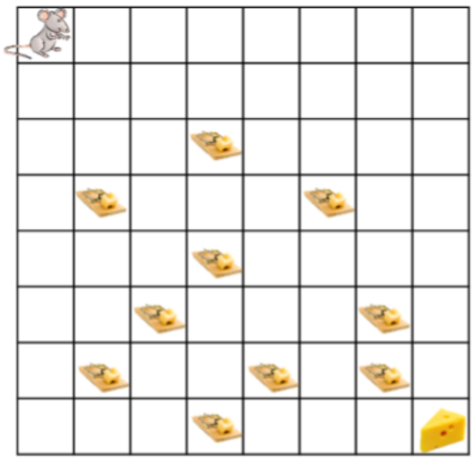


Fig.1 An example setup for the project

2. Description of learning methods

We approach to this problem by applying TD learnig and n-step Sarsa to the guide the mouse. Temporal-difference (TD) learning combines the ideas of Monte Carlo and dynamic programming. It can learn the value functions and policies directly from the experience for free-model problems that have no knowledge of environment's dymaics. Also, it updates the estimates of value functions based on part of estimates in the episode without need of finishing the whole episode and final outcomes (bootstrap).

In general, n-step TD learning uses the estimates of values and rewards after the current state to update the estimate of the current value according to samples and equation 1.

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha\left[G_{t:t+n} - V_{t+n-1}(S_t)\right], \qquad 0 \le t < T \tag{1}$$

The process of sample updates are based on a single sample successor in the sample rather than on a complete distribution of all possible successors. The estimates of value functions will converge to the real expected one when the sample size is big enough.

Sarsa is a kind of on-policy TD control method which follows the patter of generalized policy iteration (GPI) for the evaluation and prediction work. It learns an action-value function rather than a state-value function.

Each step contains the current state-action pair, a reward after taking the action and transmit from current state to the next state and the next state-action pair. Aimming to estimate action-value functions for all states s and actions a, the actions are choosed according to $\epsilon - \text{greedy}$. In one-step Sarsa, the estimate of current state-action values updates after every transition from a nonterminal state S_t according to equation 2. If next state S_{t+1} is terminal, the $Q(S_{t+1}, A_{t+1})$ is defined as zero.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big] \tag{2}$$

In n-step Sarsa, the current state-action values updates after a n-step bootstrap according to the equation 3.

$$Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha\left[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)\right], \qquad 0 \le t < T \tag{3}$$

3. MDP frame for the problem

Assuming that there are m traps in a $d \times d$ grid world. We denote $s \in S^2$ to represent the state. For each state, $s = [x, y]$ for all $0 \le x < d, 0 \le y < d$, which represent the current position of the mouse stays in.

Also, the action is defined as $a \in A$, where $a \in [North, South, East, West]$. The action value indicates which direction the mouse will choose to go next step. The mouse will get reward of r_3 when finding the cheese, and will obtain reward of r_2 when falling in traps. For other situation, the mouse will get r_1 in each step. As a result, the reward of each time step follows $r \in R$, where $r \in \{r_1, r_2, r_3\}$.

$P_{sa} = p(s', r|s, a)$ denotes the transition probabilities from current state s transmitting to next state s' and obtaining a reward r. Since it is a model-free problem, we do not have knowledge about dynamic of the environment.

As a result, the problem is designed as an MDP $M = (S, A, P_{sa}, R)$.

4. Data structures and code design

To express action-state value, we designed it as a $d \times d \times 4$ dimension array. At each state, there are 4 possible actions. To express policy, we also created a $d \times d \times 4$ dimension array to store the probability for choosing the next action given the current state. The pragram consists of three parts: Monte Carlo method, n-step Sarsa method and policy effect test.

In our code, we built 9 functions in total, excluding the main function. The description of the functions are shown if Table.1.

Table.1 Description of functions

Function name	Description
initialize_qvalue()	Generate full-zero action-state values for policy iteration
initialize_policy()	Generate a random policy for policy iteration
stimulate_episode(policy)	Generate an episode for Monte Carlo method given the policy
policy_iteration(epoch)	Update Q values and policy using Monte Carlo method.
policy_test(policy)	By testing 1000 episode with given policy, output the number of finding cheese, falling traps and follwing the shortest ways.
step(state, action)	Design for n-step Sarsa method. Given the current state and action, return the next state, reward and indicator of reaching terminal state.
make_policy(q_value, state)	Design for n-step Sarsa method. Given the current Q value and state, return the next action using $\epsilon - \mathrm{greedy}$ policy.
nStep_Sarsa_policy_iteration(n_step, num_episodes)	Design for n-step Sarsa method. Given the number of step and episode, update the Q values and policy.
plot_policy(policy)	Plot optimal policy

For Monte Carlo method, the code starts with initiating Q values and generating a random policy. In the loop of function policy_iteration(), it generates episodes and updates the Q value and policy iteratively. The loop will end depending on the number of episodes.

For n-step Sarsa, the code starts with initiating Q values and generating a random policy. In function nStep_Sarsa_policy _iteration(), the Q values are updated in a loop where the action is choosing by $\epsilon - \mathrm{greedy}$ policy. Finishing the updating of Q values, the policy begains to update using $\epsilon - \mathrm{greedy}$ policy by going through all the Q values.

After finishing the policy iteration, we designed a test demo to show the effect of generated policies by record the number of falling traps, finding cheese and following the shortest way.

Our program runs with the following parameters:

```
python3 ECE517_Project2.py d m n a g e
```

Where d is the dimension of the grid, m is the number of traps randomly placed (if m = −1 set the traps according to the example setup), n is the number of steps used in the n-step Sarsa, especially n-step Sarsa will be used when $n \leq 16$, Monte Carlo method will be used when $n > 16$, a is the value of α, g is the discount factor and e is the value of ϵ.

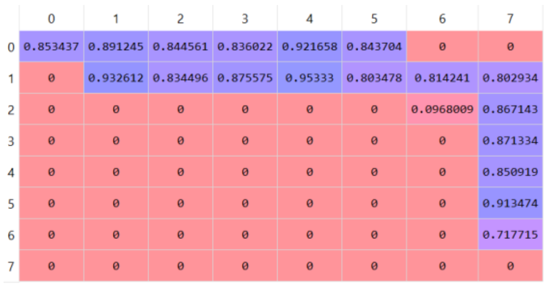
5. Results of example

In example setup, all rewards are zero except that the mouse reaches the cheese and the reward is 1. Learning rate α is 0.1, and no discount is considered. Using TD learning and Monte Carlo method, we are trying to find the ϵ-greedy policy ($\epsilon = 0.01$).

5.1 Optimal action-value function

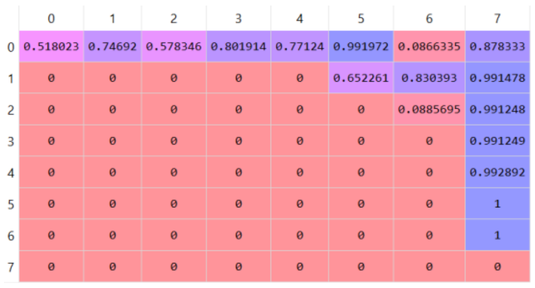
We ran 10000 episodes for TD(0) and Monte Carlo. The optimal action-value functions are presented below, where the squares represent states.

TD(0):



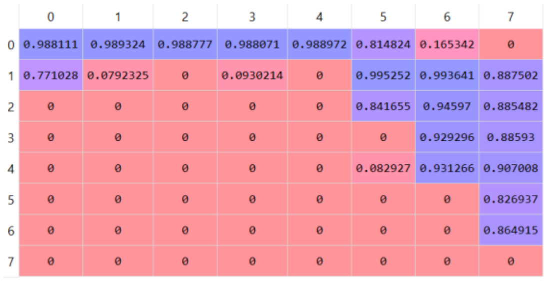
	0	1	2	3	4	5	6	7
0	0.853437	0.891245	0.844561	0.836022	0.921658	0.843704	0	0
1	0	0.932612	0.834496	0.875575	0.95333	0.803478	0.814241	0.802934
2	0	0	0	0	0	0	0.0968009	0.867143
3	0	0	0	0	0	0	0	0.871334
4	0	0	0	0	0	0	0	0.850919
5	0	0	0	0	0	0	0	0.913474
6	0	0	0	0	0	0	0	0.717715
7	0	0	0	0	0	0	0	0

(a) North



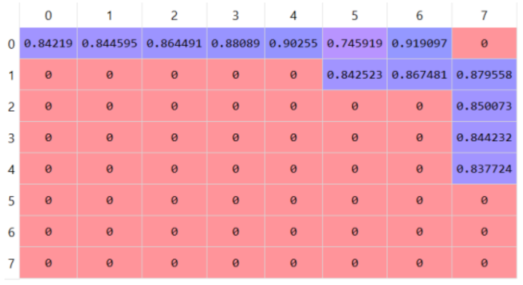
	0	1	2	3	4	5	6	7
0	0.518023	0.74692	0.578346	0.801914	0.77124	0.991972	0.0866335	0.878333
1	0	0	0	0	0	0.652261	0.830393	0.991478
2	0	0	0	0	0	0	0.0885695	0.991248
3	0	0	0	0	0	0	0	0.991249
4	0	0	0	0	0	0	0	0.992892
5	0	0	0	0	0	0	0	1
6	0	0	0	0	0	0	0	1
7	0	0	0	0	0	0	0	0

(b) South



	0	1	2	3	4	5	6	7
0	0.988111	0.989324	0.988777	0.988071	0.988972	0.814824	0.165342	0
1	0.771028	0.0792325	0	0.0930214	0	0.995252	0.993641	0.887502
2	0	0	0	0	0	0.841655	0.94597	0.885482
3	0	0	0	0	0	0	0.929296	0.88593
4	0	0	0	0	0	0.082927	0.931266	0.907008
5	0	0	0	0	0	0	0	0.826937
6	0	0	0	0	0	0	0	0.864915
7	0	0	0	0	0	0	0	0

(c) East

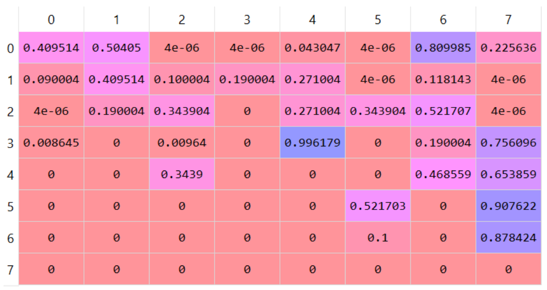


	0	1	2	3	4	5	6	7
0	0.84219	0.844595	0.864491	0.88089	0.90255	0.745919	0.919097	0
1	0	0	0	0	0	0.842523	0.867481	0.879558
2	0	0	0	0	0	0	0	0.850073
3	0	0	0	0	0	0	0	0.844232
4	0	0	0	0	0	0	0	0.837724
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0

(d) West

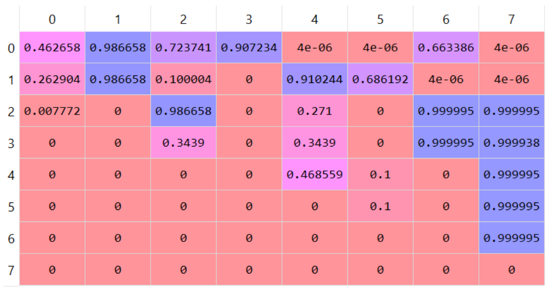
Fig.2 Optimal action-state value function with TD(0)

Monte Carlo:



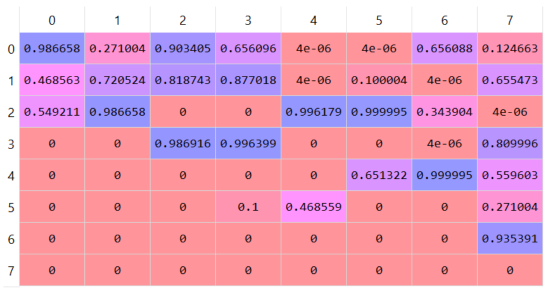
	0	1	2	3	4	5	6	7
0	0.409514	0.50405	4e-06	4e-06	0.043047	4e-06	0.809985	0.225636
1	0.090004	0.409514	0.100004	0.190004	0.271004	4e-06	0.118143	4e-06
2	4e-06	0.190004	0.343904	0	0.271004	0.343904	0.521707	4e-06
3	0.008645	0	0.00964	0	0.996179	0	0.190004	0.756096
4	0	0	0.3439	0	0	0	0.468559	0.653859
5	0	0	0	0	0	0.521703	0	0.907622
6	0	0	0	0	0	0.1	0	0.878424
7	0	0	0	0	0	0	0	0

(a) North



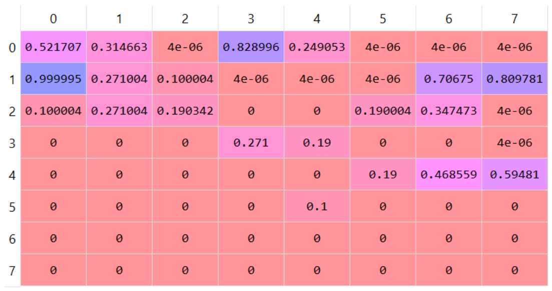
	0	1	2	3	4	5	6	7
0	0.462658	0.986658	0.723741	0.907234	4e-06	4e-06	0.663386	4e-06
1	0.262904	0.986658	0.100004	0	0.910244	0.686192	4e-06	4e-06
2	0.007772	0	0.986658	0	0.271	0	0.999995	0.999995
3	0	0	0.3439	0	0.3439	0	0.999995	0.999938
4	0	0	0	0	0.468559	0.1	0	0.999995
5	0	0	0	0	0	0.1	0	0.999995
6	0	0	0	0	0	0	0	0.999995
7	0	0	0	0	0	0	0	0

(b) South



	0	1	2	3	4	5	6	7
0	0.986658	0.271004	0.903405	0.656096	4e-06	4e-06	0.656088	0.124663
1	0.468563	0.720524	0.818743	0.877018	4e-06	0.100004	4e-06	0.655473
2	0.549211	0.986658	0	0	0.996179	0.999995	0.343904	4e-06
3	0	0	0.986916	0.996399	0	0	4e-06	0.809996
4	0	0	0	0	0	0.651322	0.999995	0.559603
5	0	0	0	0.1	0.468559	0	0	0.271004
6	0	0	0	0	0	0	0	0.935391
7	0	0	0	0	0	0	0	0

(c)East



	0	1	2	3	4	5	6	7
0	0.521707	0.314663	4e-06	0.828996	0.249053	4e-06	4e-06	4e-06
1	0.999995	0.271004	0.100004	4e-06	4e-06	4e-06	0.70675	0.809781
2	0.100004	0.271004	0.190342	0	0	0.190004	0.347473	4e-06
3	0	0	0	0.271	0.19	0	0	4e-06
4	0	0	0	0	0	0.19	0.468559	0.59481
5	0	0	0	0	0.1	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0

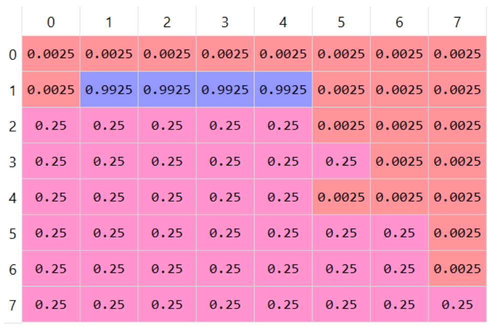
(d)West

Fig.3 Optimal action-state value function with Monte Carlo

5.2 Optimal policy

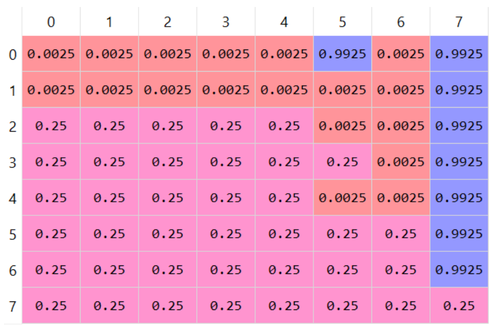
The optimal policies learned by TD(0) and Monte Carlo method are shown below, where the squares represent states and the numbers in the squares represent the probability of choosing the action in that square.

TD(0):



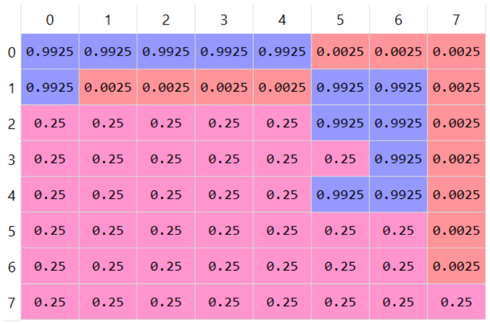
	0	1	2	3	4	5	6	7
0	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025
1	0.0025	0.9925	0.9925	0.9925	0.9925	0.0025	0.0025	0.0025
2	0.25	0.25	0.25	0.25	0.25	0.0025	0.0025	0.0025
3	0.25	0.25	0.25	0.25	0.25	0.25	0.0025	0.0025
4	0.25	0.25	0.25	0.25	0.25	0.0025	0.0025	0.0025
5	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.0025
6	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.0025
7	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25

(a) North



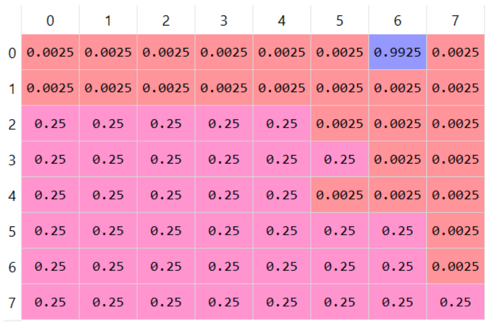
	0	1	2	3	4	5	6	7
0	0.0025	0.0025	0.0025	0.0025	0.0025	0.9925	0.0025	0.9925
1	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025	0.9925
2	0.25	0.25	0.25	0.25	0.25	0.0025	0.0025	0.9925
3	0.25	0.25	0.25	0.25	0.25	0.25	0.0025	0.9925
4	0.25	0.25	0.25	0.25	0.25	0.0025	0.0025	0.9925
5	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.9925
6	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.9925
7	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25

(b) South



	0	1	2	3	4	5	6	7
0	0.9925	0.9925	0.9925	0.9925	0.9925	0.0025	0.0025	0.0025
1	0.9925	0.0025	0.0025	0.0025	0.0025	0.9925	0.9925	0.0025
2	0.25	0.25	0.25	0.25	0.25	0.9925	0.9925	0.0025
3	0.25	0.25	0.25	0.25	0.25	0.25	0.9925	0.0025
4	0.25	0.25	0.25	0.25	0.25	0.9925	0.9925	0.0025
5	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.0025
6	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.0025
7	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25

(c)East

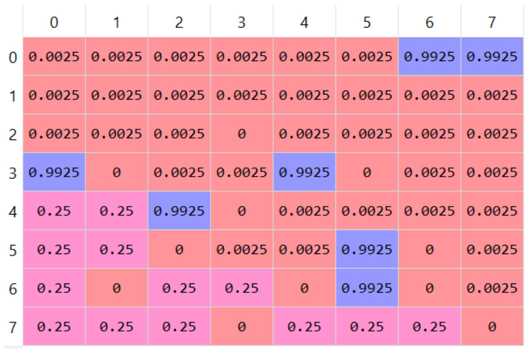


	0	1	2	3	4	5	6	7
0	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025	0.9925	0.0025
1	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025
2	0.25	0.25	0.25	0.25	0.25	0.0025	0.0025	0.0025
3	0.25	0.25	0.25	0.25	0.25	0.25	0.0025	0.0025
4	0.25	0.25	0.25	0.25	0.25	0.0025	0.0025	0.0025
5	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.0025
6	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.0025
7	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25

(d)West

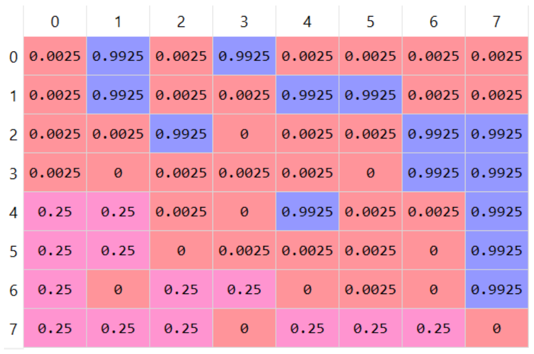
Fig.4 Optimal policy with TD(0)

Monte Carlo:



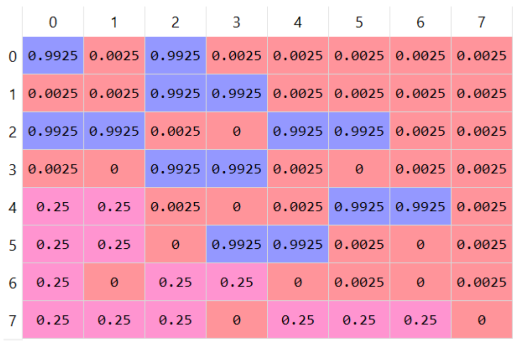
	0	1	2	3	4	5	6	7
0	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025	0.9925	0.9925
1	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025
2	0.0025	0.0025	0.0025	0	0.0025	0.0025	0.0025	0.0025
3	0.9925	0	0.0025	0.0025	0.9925	0	0.0025	0.0025
4	0.25	0.25	0.9925	0	0.0025	0.0025	0.0025	0.0025
5	0.25	0.25	0	0.0025	0.0025	0.9925	0	0.0025
6	0.25	0	0.25	0.25	0	0.9925	0	0.0025
7	0.25	0.25	0.25	0	0.25	0.25	0.25	0

(a) North



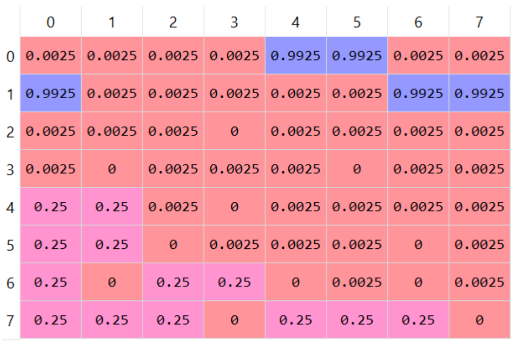
	0	1	2	3	4	5	6	7
0	0.0025	0.9925	0.0025	0.9925	0.0025	0.0025	0.0025	0.0025
1	0.0025	0.9925	0.0025	0.0025	0.9925	0.9925	0.0025	0.0025
2	0.0025	0.0025	0.9925	0	0.0025	0.0025	0.9925	0.9925
3	0.0025	0	0.0025	0.0025	0.0025	0	0.9925	0.9925
4	0.25	0.25	0.0025	0	0.9925	0.0025	0.0025	0.9925
5	0.25	0.25	0	0.0025	0.0025	0.0025	0	0.9925
6	0.25	0	0.25	0.25	0	0.0025	0	0.9925
7	0.25	0.25	0.25	0	0.25	0.25	0.25	0

(b) South



	0	1	2	3	4	5	6	7
0	0.9925	0.0025	0.9925	0.0025	0.0025	0.0025	0.0025	0.0025
1	0.0025	0.0025	0.9925	0.9925	0.0025	0.0025	0.0025	0.0025
2	0.9925	0.9925	0.0025	0	0.9925	0.9925	0.0025	0.0025
3	0.0025	0	0.9925	0.9925	0.0025	0	0.0025	0.0025
4	0.25	0.25	0.0025	0	0.0025	0.9925	0.9925	0.0025
5	0.25	0.25	0	0.9925	0.9925	0.0025	0	0.0025
6	0.25	0	0.25	0.25	0	0.0025	0	0.0025
7	0.25	0.25	0.25	0	0.25	0.25	0.25	0

(c) East



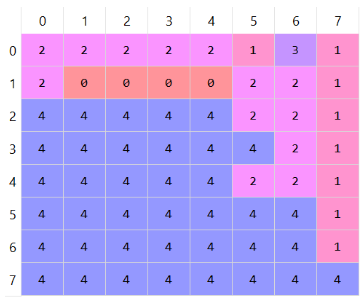
	0	1	2	3	4	5	6	7
0	0.0025	0.0025	0.0025	0.0025	0.9925	0.9925	0.0025	0.0025
1	0.9925	0.0025	0.0025	0.0025	0.0025	0.0025	0.9925	0.9925
2	0.0025	0.0025	0.0025	0	0.0025	0.0025	0.0025	0.0025
3	0.0025	0	0.0025	0.0025	0.0025	0	0.0025	0.0025
4	0.25	0.25	0.0025	0	0.0025	0.0025	0.0025	0.0025
5	0.25	0.25	0	0.0025	0.0025	0.0025	0	0.0025
6	0.25	0	0.25	0.25	0	0.0025	0	0.0025
7	0.25	0.25	0.25	0	0.25	0.25	0.25	0

(d) West

Fig.5 Optimal policy with Monte Carlo

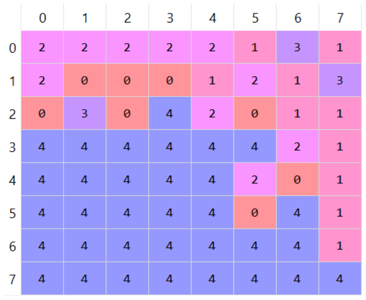
5.3 The progression of the policy

To present the progression of policy effectiveness under different values of n, we added 2-, 4-, 8-, and 16-step TD for policy comparison.



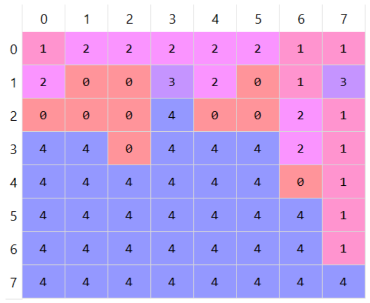
	0	1	2	3	4	5	6	7
0	2	2	2	2	2	1	3	1
1	2	0	0	0	0	2	2	1
2	4	4	4	4	4	2	2	1
3	4	4	4	4	4	4	2	1
4	4	4	4	4	4	2	2	1
5	4	4	4	4	4	4	4	1
6	4	4	4	4	4	4	4	1
7	4	4	4	4	4	4	4	4

(a) TD(0)



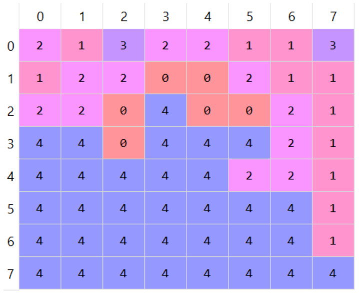
	0	1	2	3	4	5	6	7
0	2	2	2	2	2	1	3	1
1	2	0	0	0	1	2	1	3
2	0	3	0	4	2	0	1	1
3	4	4	4	4	4	4	2	1
4	4	4	4	4	4	2	0	1
5	4	4	4	4	4	0	4	1
6	4	4	4	4	4	4	4	1
7	4	4	4	4	4	4	4	4

(b) 2-step TD



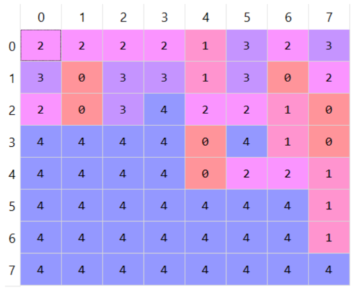
	0	1	2	3	4	5	6	7
0	1	2	2	2	2	2	1	1
1	2	0	0	3	2	0	1	3
2	0	0	0	4	0	0	2	1
3	4	4	0	4	4	4	2	1
4	4	4	4	4	4	4	0	1
5	4	4	4	4	4	4	4	1
6	4	4	4	4	4	4	4	1
7	4	4	4	4	4	4	4	4

(c) 4-step TD



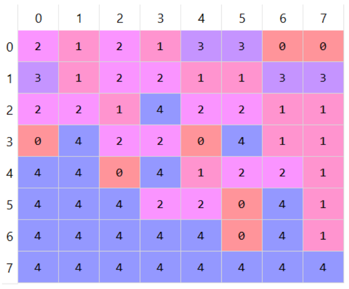
	0	1	2	3	4	5	6	7
0	2	1	3	2	2	1	1	3
1	1	2	2	0	0	2	1	1
2	2	2	0	4	0	0	2	1
3	4	4	0	4	4	4	2	1
4	4	4	4	4	4	2	2	1
5	4	4	4	4	4	4	4	1
6	4	4	4	4	4	4	4	1
7	4	4	4	4	4	4	4	4

(d) 8-step TD



	0	1	2	3	4	5	6	7
0	2	2	2	2	1	3	2	3
1	3	0	3	3	1	3	0	2
2	2	0	3	4	2	2	1	0
3	4	4	4	4	0	4	1	0
4	4	4	4	4	0	2	2	1
5	4	4	4	4	4	4	4	1
6	4	4	4	4	4	4	4	1
7	4	4	4	4	4	4	4	4

(e) 16-step TD



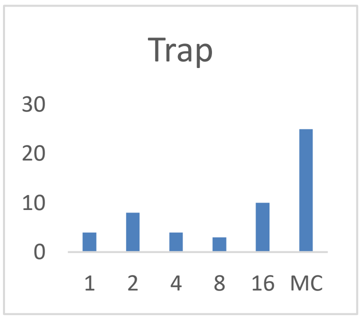
	0	1	2	3	4	5	6	7
0	2	1	2	1	3	3	0	0
1	3	1	2	2	1	1	3	3
2	2	2	1	4	2	2	1	1
3	0	4	2	2	0	4	1	1
4	4	4	0	4	1	2	2	1
5	4	4	4	2	2	0	4	1
6	4	4	4	4	4	0	4	1
7	4	4	4	4	4	4	4	4

(f) MC

Fig.6 Progression of policy effectiveness

The number indicates the most likely action the mouse will take at each grid. 0 indicates north, 1 south, 2 east and 3 west. 4 suggests that there is equal probability for the mouse to move in any direction, unless that grid is terminal state (cheese or trap). Graphs above suggests that the agent explores more regions located at the bottom left of the grid as the n increases, given the same episode. Most grids were explored by Monte Carlo method and least by TD(0).

We also show how many times the mouse gets the cheese under the ϵ-greedy policy given 1000 trials. The results are shown in Fig.7 and Table.2.



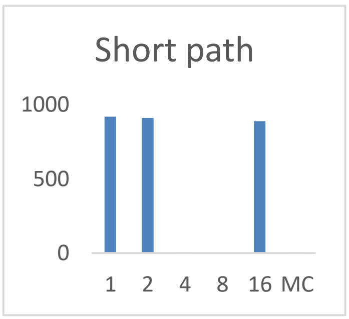


Fig.7 Results after 1000 trials with different policies

Table.2 Results after 1000 trials with different policies

n-step	Trap	Succeed	Short path
1	4	78	918
2	8	82	910
4	4	992	4
8	3	995	2
16	10	101	889
MC	25	971	4

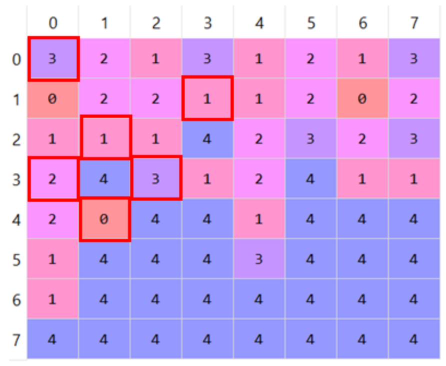
All policies are able to guide the mouse finding the short paths to the cheese. However, as n increase, the mouse have less chance to choose the shortest paths to the cheese, especially for Monte Carlo. Moreover, the mouse is more likely to get trapped as n increases, though rate of success is close to 99% in all cases.

6. Results Analysis

6.1 Interesting question 1

It is interesting to see that the policy learned by Monte Carlo method doesn't give most support to the shortest paths. This may result from no penalty to unsuccessful searches. Therefore, we add -1 as penalty if the mouse hits the trap. Surprisingly, the agent failed to find the path to the cheese.

By looking at policy in this case which is shown in Fig.8, especially grids that are marked by red square, we realize the agent fails to find the cheese and tries to get less penalty as it can. First, the agent fears to search the cheese and is more willing to stay at starting point, indicating the agent knows it has little chance to get rewards. Second, if the agent somehow reaches out, it chooses the "best" trap to end the task.

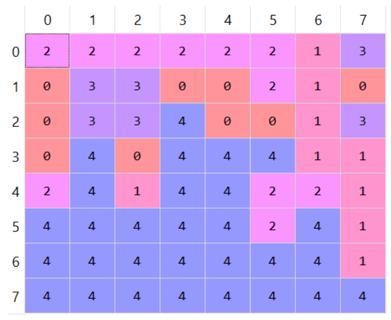


	0	1	2	3	4	5	6	7
0	3	2	1	3	1	2	1	3
1	0	2	2	1	1	2	0	2
2	1	1	1	4	2	3	2	3
3	2	4	3	1	2	4	1	1
4	2	0	4	4	1	4	4	4
5	1	4	4	4	3	4	4	4
6	1	4	4	4	4	4	4	4
7	4	4	4	4	4	4	4	4

Fig.8 Policy adding penalty on falling traps

$\alpha = 0.1$; $g = 1$; $\epsilon = 0.01$; episode = 10000

We gave a discount (0.9) to encourage the agent searching. The new policy is presented below.



	0	1	2	3	4	5	6	7
0	2	2	2	2	2	2	1	3
1	0	3	3	0	0	2	1	0
2	0	3	3	4	0	0	1	3
3	0	4	0	4	4	4	1	1
4	2	4	1	4	4	2	2	1
5	4	4	4	4	4	2	4	1
6	4	4	4	4	4	4	4	1
7	4	4	4	4	4	4	4	4

Fig.9 Policy adding discount

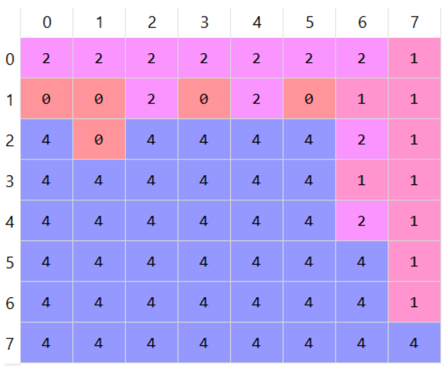
$\alpha = 0.1$; $g = 0.9$; $\epsilon = 0.01$; episode = 10000

After being given 0.9 discount, the agent learned the best way to get the cheese. We also noticed that in a new run of 10000 episodes, the agent adapted a similar policy that was not given discount. This prompted us to think while exploring the grids, if the agent doesn't find the cheese, the agent would choose the "best" trap that gives it least penalty or stay at starting point with no worry of penalty (results not shown).

6.2 Interesting question 2

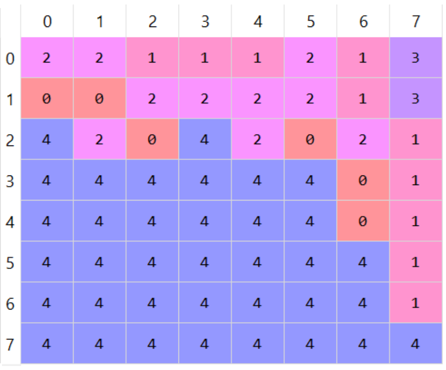
We ran 10000 episodes in our program and found some good policies that can guide the mouse to reach the cheese. However, as we can see in Fig.6, there are still some action-state values not been visited or updated. Also, we notice that adding the number of step in n-step Sarsa method encourages the mouse to explore. Intuitively, choosing larger ϵ gives more chance to explore. We apply different ϵ to see the difference, and the results are shown in Fig.10.

As we can see, by choosing larger ϵ, the mouse are encouraged to explore more states, and the final policy also changes. Also, larger ϵ leads to a small converge speed, and we found it very hard to visit every state even we choose a very large ϵ. However, it have no influence for generating a good enough policy (choose the shortest way). That shows the power of TD learning: it doesn't need to go through every state-action pair to provide a good enough policy, just let it run and try!



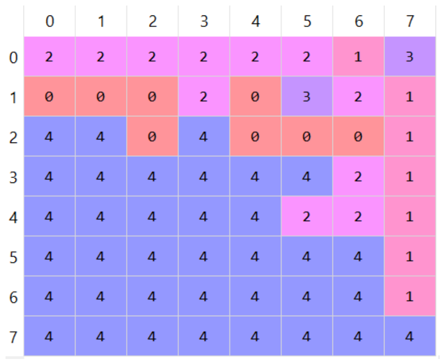
	0	1	2	3	4	5	6	7
0	2	2	2	2	2	2	2	1
1	0	0	2	0	2	0	1	1
2	4	0	4	4	4	4	2	1
3	4	4	4	4	4	4	1	1
4	4	4	4	4	4	4	2	1
5	4	4	4	4	4	4	4	1
6	4	4	4	4	4	4	4	1
7	4	4	4	4	4	4	4	4

(a) $\epsilon = 0.01$



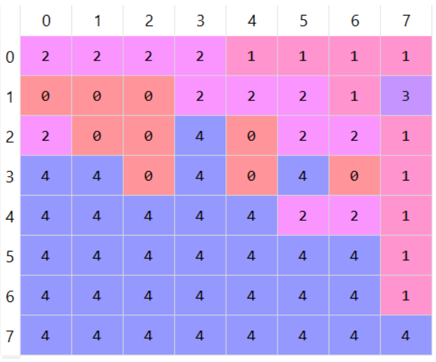
	0	1	2	3	4	5	6	7
0	2	2	1	1	1	2	1	3
1	0	0	2	2	2	2	1	3
2	4	2	0	4	2	0	2	1
3	4	4	4	4	4	4	0	1
4	4	4	4	4	4	4	0	1
5	4	4	4	4	4	4	4	1
6	4	4	4	4	4	4	4	1
7	4	4	4	4	4	4	4	4

(b) $\epsilon = 0.02$



	0	1	2	3	4	5	6	7
0	2	2	2	2	2	2	1	3
1	0	0	0	2	0	3	2	1
2	4	4	0	4	0	0	0	1
3	4	4	4	4	4	4	2	1
4	4	4	4	4	4	2	2	1
5	4	4	4	4	4	4	4	1
6	4	4	4	4	4	4	4	1
7	4	4	4	4	4	4	4	4

(c) $\epsilon = 0.05$



	0	1	2	3	4	5	6	7
0	2	2	2	2	1	1	1	1
1	0	0	0	2	2	2	1	3
2	2	0	0	4	0	2	2	1
3	4	4	0	4	0	4	0	1
4	4	4	4	4	4	2	2	1
5	4	4	4	4	4	4	4	1
6	4	4	4	4	4	4	4	1
7	4	4	4	4	4	4	4	4

(d) $\epsilon = 0.1$

Fig.10 Optimal policies with different ϵ using TD(0)

7. Conclusion

Both TD learning and Monte Carlo method can be used to find the optimal policy that guides the mouse finding the cheese. However, the policies learned by these two methods are slightly different. TD learning is more efficient to find the best path to the cheese, but Monte Carlo focuses only on getting the cheese. Meanwhile, the difference is also result from the learning parameters. For instance, by giving discount and negative rewards, the agent is able to find the best solution via Monte Carlo.

There are a lot of factors influencing the final results and converge speed. In most of our trials, even set the episodes as 10000, some action-states values are still not been visited or updated. However, when we choose larger ϵ to give more chance to explore, the converge speed decreased. It does need some time and patience to find better parameters fitting our problem.

Appendix

Tasks	Subtasks	Member in charge of
Coding	Monte Carlo method	Yang Xu
	Test demo	Yang Xu
	n-step Sarsa method	Hui Liu
Report	Introduction	Yang Xu
	Dynamic programming	Hui Liu
	Workframe for problem	Hui Liu
	Data structures and code design	Hui Liu
	Results of example	Yang Xu
	Results analysis	Pair
	Conclusion	Pair