HW#2 Report

2018312280
이상수
인공지능개론 41 분반
교수: 박호건

Implementation:

In solver.py file, made 4 functions and 1 class "Q_learning"

Function get_max_action takes q_table entry for k_curr, l_curr and finds the max action value among 4 actions (down, right, up, left).

If value is equal to max than 50% chance to change the action. This is for situations where more than two action has same expected reward(Q value).

Function get_max_reward takes the same parameter but return only max q value from q_table(s, among all a)

Function get_reward is same as HW requirement.

If there is wall between, return -1, if it is exit return 100, and if it is just cell return -0.1

Function q_learning is main function where learning for agent is made.

It learns for number of episodes and in each episode has 100000 iterations at most are made until it finds the exit. However, later I changed iteration limit for episode < 10 to have iteration of 100000 because initial episodes are more unlikely to find exit and rest of episodes to have 30000 iterations .

For each iteration it takes 4 neighbor cells and get random e_decision value from 0~1. If e_decision>e than choose action using q_table, and if e_decision<=e choose action in random(0~3). Then according to action, it lookup the q_table and change the value for the action.

Equation for updating the q value is same as HW PDF: Ex)

```
q_table[k_curr*row_max+l_curr*1][action]=
q_table[k_curr*row_max+l_curr*1][action]+
a*(
reward
+r*get_max_reward(q_table[(k_curr-1)*row_max+l_curr*1], k_curr-1,
l_curr,neighbour_indices2)
-q_table[k_curr*row_max+l_curr*1][action]
)
```

It takes current Q value and add (learning rate) * (reward + (decay rate) *(max Q value from next cell) – current Q value)

This makes Q value to be more and more low for each iteration for same action except if next cell is exiting cell.

If the chosen action is not blocked by wall, it goes on to next iteration. If it is blocked, it doesn't use max Q value from next cell to update Q value because it will not go there, and iteration doesn't proceed, and it must find new action.

For blocked way Q value update equation changes : Ex)

```
q_table[k_curr*row_max+l_curr*1][action]=
q_table[k_curr*row_max+l_curr*1][action]+
a*(reward-q_table[k_curr*row_max+l_curr*1][action])
```

without Q value of next cell.

After all the episodes ends, it returns q_table.

Class Q_learning:

This class is almost same as other classes.

After it calls q_learning function, it gets q_table.

It iterates 10000 times max to find exiting cell without any help except q_table. There is no random move unless the value of actions in q_table are same. If it is blocked by wall, it must find its own way to return. Therefore, there is no additional backtracking algorithm. If the q_table is not good enough, the agent will never find the exit.

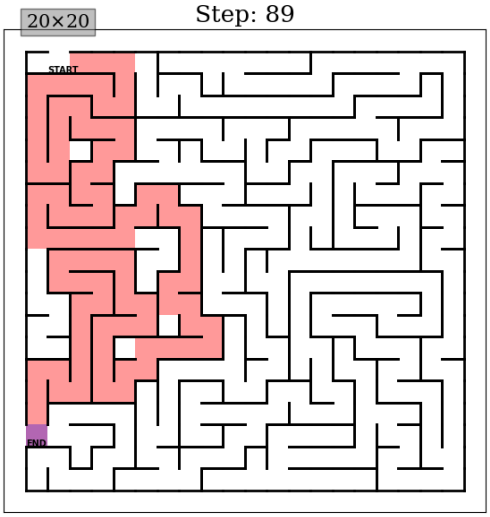
Additional changes are made in "maze_manager.py" to add method of Q_learning.

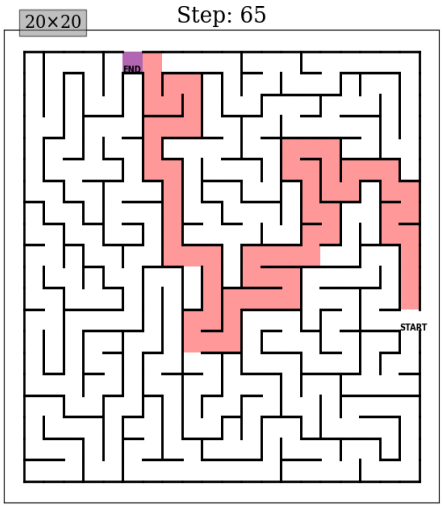
The method is made to apply episodes, a, r, and e value when calling the method in running main file("Q_learning.py").

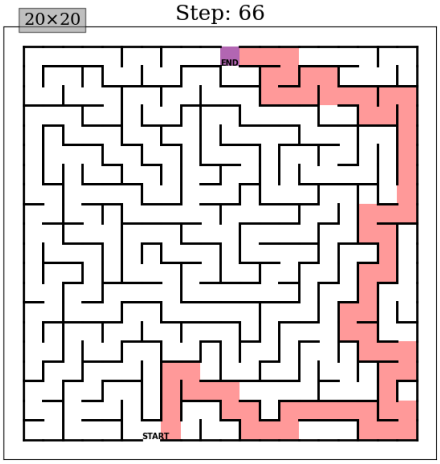
Results:

Random 3 mazes

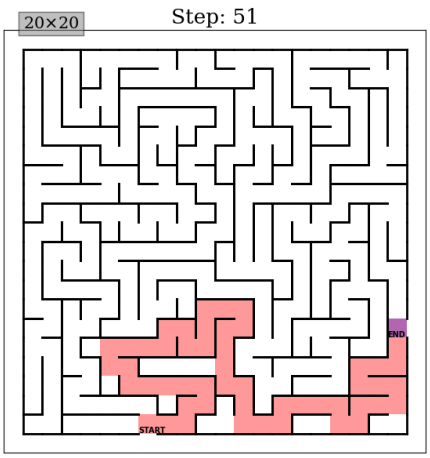
episodes: 100, a=0.9, r=0.9, e=0.5

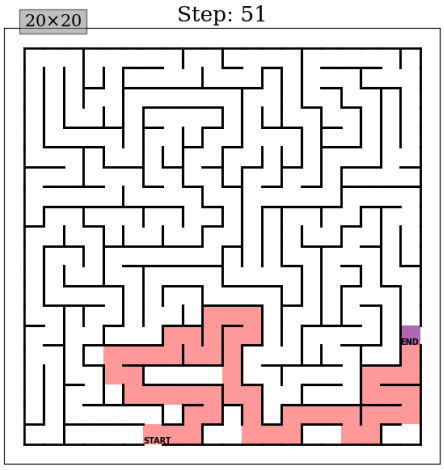


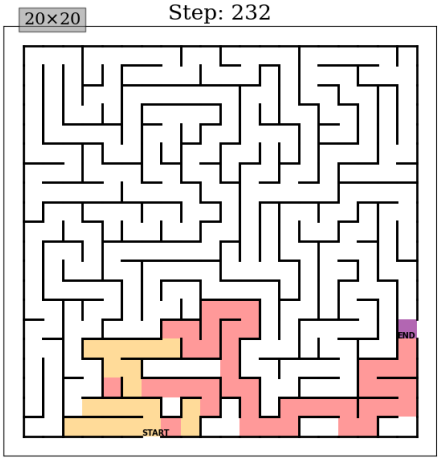


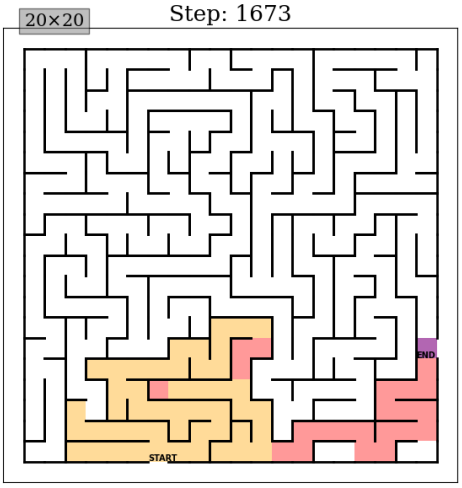


a	r	total_iteration for learning
0.9	0.9	12038
0.7	0.7	13820
0.5	0.5	17774
0.3	0.3	28492





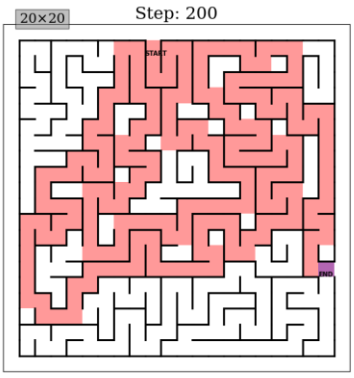




The agent back tracks by itself and therefore some cells are backtracked and is shown yellow even if it is part of optimal path.
Conditions: epsilon value=0.4, episodes=50
I found that learning rate 'a', and decay rate 'r' is crucial to finding optimal q_table.
When "a" and "r" are big, agent learns fast and q_table is optimized quickly. If 'a' is small, agent takes longer to find path and make q_table because its learning value changes slowly. If 'r' is small, q_table values are less affected by next step, and this leads to slow learning rate from goal state. It is crucial for 'r' to be big enough for other states to know which action leads to goal state.

Additionally, epsilon value is important too. Epsilon decides whether to explore in randomly or move according to q_table. It is better to have high 'e' value at initial learning episodes because q_table is nearly empty. However, as it learns through the episodes, it is better to have lower 'e' value so that it can train the data that is closer to optimal path. Therefore, some reinforcement learning uses epsilon decay rate but in this HW, we didn't have it so it is important to set 'e' value that is not too high or too low.

Episodes and iteration for each episode is important too. As you can see above, at first, I used high iteration limit and low episode value. High iteration is good for agent that is new to maze and has little info in q_table. However, as agent learns the maze, it is more important to go to goal state faster than before and train each state to have optimal value related to goal state. Moreover, I had some issues with Q value to be not optimal but needs more learning to do optimal action in each state especially if state is far from goal state and is not likely to be affected by goal state. Therefore, I had to limit the Q values to round up to 7 decimal places (.7f) and as you can see above, this made most Q values far from goal state to have same value and move in random action, especially for those with small 'a' and 'r' values and they needed to back track.
Only reward from goal state (reward=100) can show other states how to find it and to do so, agent must visit goal state more often and less visit the unneeded states. Therefore, I changed my way to have high episode value and lower iteration limit. This will make agent visit goal state more often and update the Q values in q_table to have higher value that is in optimal path.



This agent condition was:

episodes: 100000, a=0.7, b=0.7, e=0.4.

Even when maze has long optimal path, agent was able to find optimal path at one take. As same as before, agent had to back track by its own but agent didn't back track at all because q_table was able to show optimal path by its own and this time I didn't need to round up to 7 decimal places.