< 지능제어 4차 숙제 결과보고서 >

2018142102 홍의진

1. "agent.py"의 "write your code here" 부분 code

1.1. "agent.py" Q value update 부분 code

```
66          while not (done or timeout):
67              # Next state and action generation
68              action = self.get_action(state,epsilon)
69              movement = ACTIONS[action]
70              next_state, reward = self.env.interaction(state, movement)
71
72              # ***********   Q value update   ***************
73              # Greedy method를 통해 현재 policy를 바탕으로 expected return이 가장 큰 action을 찾음
74              max_action = np.argmax(self.Q_values[next_state[0], next_state[1]])
75              # Constant alpha 방식으로 sample mean을 구하여 현재 state, action에서의 Q value를 update
76              self.Q_values[state[0], state[1], action] += alpha * (reward + discount \
77              * self.Q_values[next_state[0], next_state[1], max_action] \
78              - self.Q_values[state[0], state[1], action])
79              # Interaction을 통해 얻은 next state를 다음 iteration에서의 현재 state로 사용한다.
80              state = next_state
81              # ************************************************
82
83              seq_len += 1
84              if (seq_len >= max_seq_len):
85                  timeout = True
86              done = self.env.is_terminal(state)
```

<그림1. "agent.py" Monte_Carlo_Control 함수의 Sequence Generation 부분 code>

2. "agent.py"의 "Write your code here" 부분에 대한 자세한 주석

2.1. "agent.py" Q value update 부분에 대한 자세한 주석

Line 74: Greedy method를 통해 현재 policy를 바탕으로 expected return이 가장 큰 action을 찾는다.

Line 76~78: Constant alpha 방식으로 sample mean을 구하여 현재 state, action에서의 Q value를 update 한다. 이때, Q value update function은 아래와 같다.

$$Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$$

Line 80: Interaction을 통해 얻은 next state를 다음 iteration에서의 현재 state로 사용한다.

3. “main_train.py” code 실행 결과

Policy Evaluation: Iteration:1

1	-17.0	-16.0	X	-10.0	X	-8.0	-7.0	-6.0	-5.0	-4.0
2	-16.0	-15.0	X	-9.0	-8.0	-7.0	-6.0	-5.0	-4.0	-3.0
3	-15.0	-14.0	X	-10.0	X	-8.0	X	X	X	-2.0
4	-14.0	-13.0	-12.0	-11.0	-10.0	-9.0	X	-3.0	-2.0	-1.0
5	-15.0	-14.0	-13.0	-12.0	X	-10.0	X	-2.0	-1.0	
	1	2	3	4	5	6	7	8	9	10

<그림2. Q-learning에서 구한 state value function의 값>

Policy Improvement: Iteration:1

1	→	↓	X	↓	X	→	→	→	→	↓
2	→	↓	X	→	→	→	→	→	→	↓
3	→	↓	X	↑	X	↑	X	X	X	↓
4	→	→	→	→	→	↑	X	→	→	↓
5	→	→	→	↑	X	↑	X	→	→	
	1	2	3	4	5	6	7	8	9	10

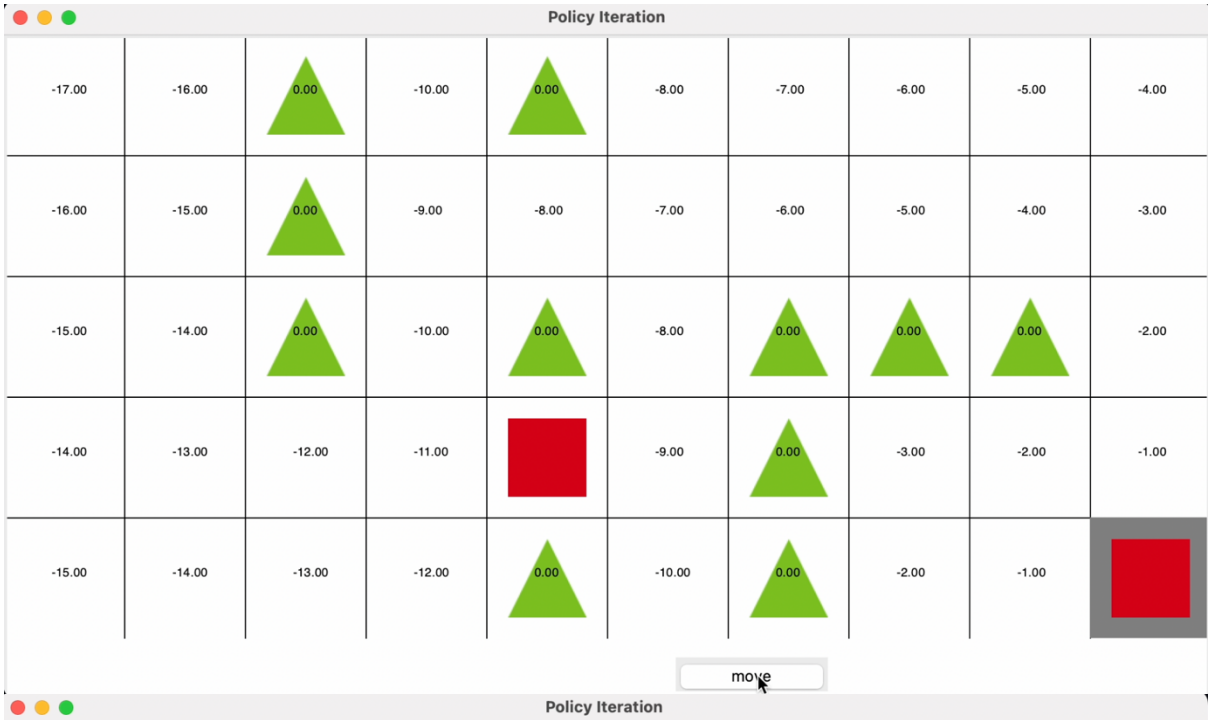
<그림3. Q-learning에서 구한 behavior policy의 값>

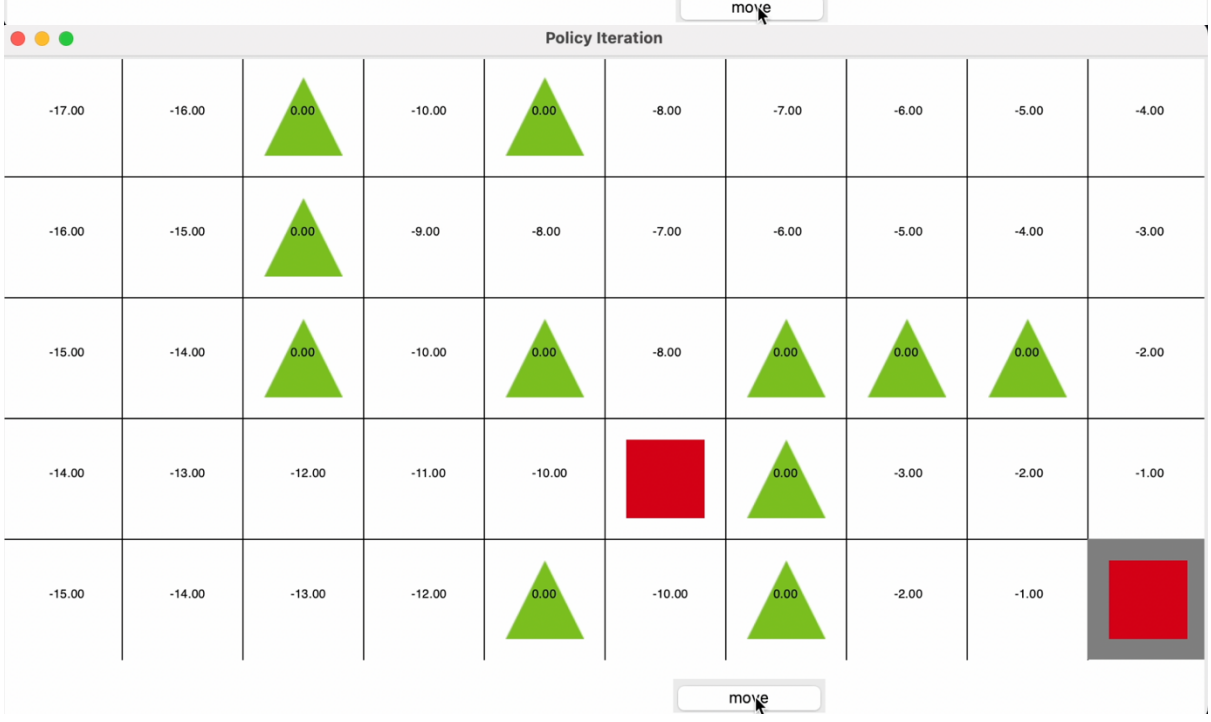
```
Num of episodes = 0, epsilon=0.4000
Num of episodes = 10000, epsilon=0.3200
Num of episodes = 20000, epsilon=0.2560
Num of episodes = 30000, epsilon=0.2048
Num of episodes = 40000, epsilon=0.1638
Num of episodes = 50000, epsilon=0.1311
Num of episodes = 60000, epsilon=0.1049
Num of episodes = 70000, epsilon=0.0839
Num of episodes = 80000, epsilon=0.0671
Num of episodes = 90000, epsilon=0.0537
Num of episodes = 100000, epsilon=0.0429
Num of episodes = 110000, epsilon=0.0344
Num of episodes = 120000, epsilon=0.0275
Num of episodes = 130000, epsilon=0.0220
Num of episodes = 140000, epsilon=0.0176
Num of episodes = 150000, epsilon=0.0141
Num of episodes = 160000, epsilon=0.0113
Num of episodes = 170000, epsilon=0.0090
Num of episodes = 180000, epsilon=0.0072
Num of episodes = 190000, epsilon=0.0058
Num of episodes = 200000, epsilon=0.0046
Num of episodes = 210000, epsilon=0.0037
Num of episodes = 220000, epsilon=0.0030
Num of episodes = 230000, epsilon=0.0024
Num of episodes = 240000, epsilon=0.0019
Num of episodes = 250000, epsilon=0.0015
Num of episodes = 260000, epsilon=0.0012
Num of episodes = 270000, epsilon=0.0010
Num of episodes = 280000, epsilon=0.0008
Num of episodes = 290000, epsilon=0.0006
Num of episodes = 300000, epsilon=0.0005
Num of episodes = 310000, epsilon=0.0004
Num of episodes = 320000, epsilon=0.0003
Num of episodes = 330000, epsilon=0.0003
Num of episodes = 340000, epsilon=0.0002
Num of episodes = 350000, epsilon=0.0002
Num of episodes = 360000, epsilon=0.0001
Num of episodes = 370000, epsilon=0.0001
Num of episodes = 380000, epsilon=0.0001
Num of episodes = 390000, epsilon=0.0001
Num of episodes = 400000, epsilon=0.0001
Num of episodes = 410000, epsilon=0.0000
Num of episodes = 420000, epsilon=0.0000
Num of episodes = 430000, epsilon=0.0000
Num of episodes = 440000, epsilon=0.0000
Num of episodes = 450000, epsilon=0.0000
Num of episodes = 460000, epsilon=0.0000
Num of episodes = 470000, epsilon=0.0000
Num of episodes = 480000, epsilon=0.0000
Num of episodes = 490000, epsilon=0.0000
```

<그림4. Episode 개수에 따른 epsilon 감소 추세>

4. “main_test.py” code 실행 결과


Policy Iteration
-17.00 -16.00 0.00 -10.00 0.00 -8.00 -7.00 -6.00 -5.00 -4.00
-16.00 -15.00 0.00 -9.00 -8.00 -7.00 -6.00 -5.00 -4.00 -3.00
-15.00 -14.00 0.00 -10.00 0.00 -8.00 0.00 0.00 0.00 -2.00
-14.00 -13.00 -11.00 -10.00 -9.00 0.00 -3.00 -2.00 -1.00
-15.00 -14.00 -13.00 -12.00 0.00 -10.00 0.00 -2.00 -1.00
move
Policy Iteration
-17.00 -16.00 0.00 -10.00 0.00 -8.00 -7.00 -6.00 -5.00 -4.00
-16.00 -15.00 0.00 -9.00 -8.00 -7.00 -6.00 -5.00 -4.00 -3.00
-15.00 -14.00 0.00 -10.00 0.00 -8.00 0.00 0.00 0.00 -2.00
-14.00 -13.00 -12.00 -10.00 -9.00 0.00 -3.00 -2.00 -1.00
-15.00 -14.00 -13.00 -12.00 0.00 -10.00 0.00 -2.00 -1.00
move


Policy Iteration
-17.00 -16.00 0.00 -10.00 0.00 -8.00 -7.00 -6.00 -5.00 -4.00
-16.00 -15.00 0.00 -9.00 -8.00 -7.00 -6.00 -5.00 -4.00 -3.00
-15.00 -14.00 0.00 -10.00 0.00 -8.00 0.00 0.00 0.00 -2.00
-14.00 -13.00 -12.00 -11.00 -9.00 0.00 -3.00 -2.00 -1.00
-15.00 -14.00 -13.00 -12.00 0.00 -10.00 0.00 -2.00 -1.00
move

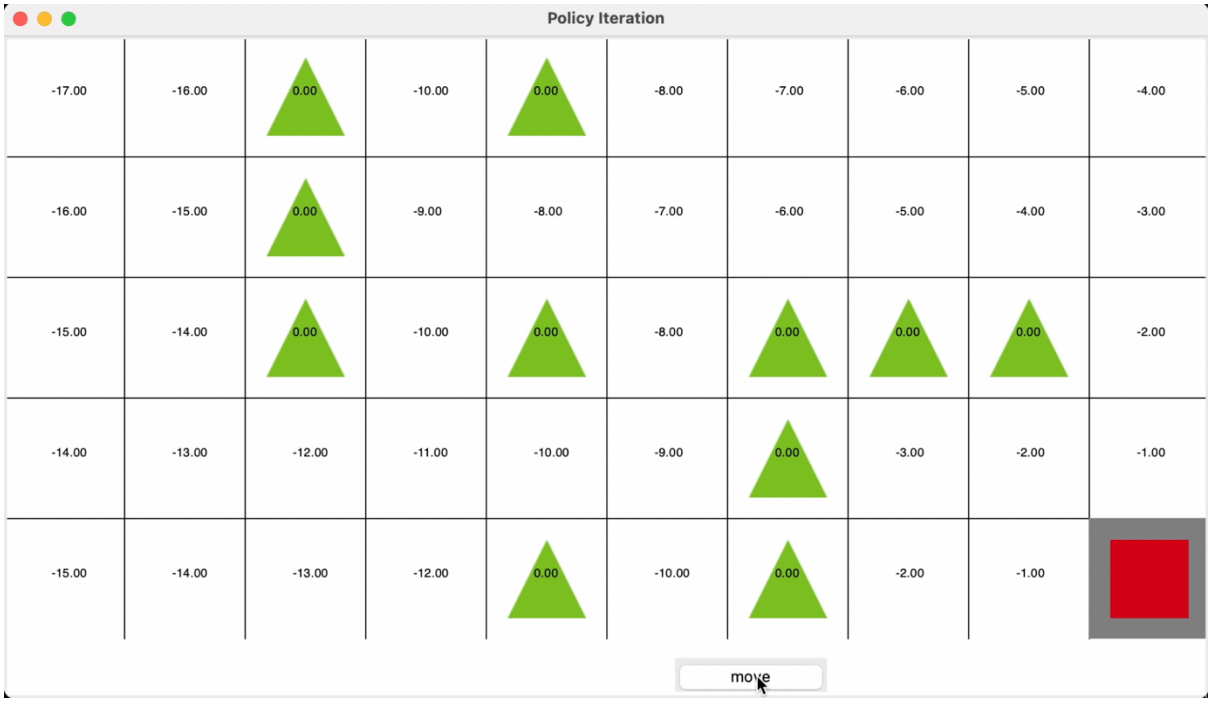

Policy Iteration
-17.00 -16.00 0.00 -10.00 0.00 -8.00 -7.00 -6.00 -5.00 -4.00
-16.00 -15.00 0.00 -9.00 -8.00 -7.00 -6.00 -5.00 -4.00 -3.00
-15.00 -14.00 0.00 -10.00 0.00 -8.00 0.00 0.00 0.00 -2.00
-14.00 -13.00 -12.00 -11.00 -10.00 0.00 -3.00 -2.00 -1.00
-15.00 -14.00 -13.00 -12.00 0.00 -10.00 0.00 -2.00 -1.00
move


Policy Iteration
-17.00 -16.00 0.00 -10.00 0.00 -8.00 -7.00 -6.00 -5.00 -4.00
-16.00 -15.00 0.00 -9.00 -8.00 -7.00 -6.00 -5.00 -4.00 -3.00
-15.00 -14.00 0.00 -10.00 0.00 0.00 0.00 0.00 -2.00
-14.00 -13.00 -12.00 -11.00 -10.00 -9.00 0.00 -3.00 -2.00 -1.00
-15.00 -14.00 -13.00 -12.00 0.00 -10.00 0.00 -2.00 -1.00
move
Policy Iteration
-17.00 -16.00 0.00 -10.00 0.00 -8.00 -7.00 -6.00 -5.00 -4.00
-16.00 -15.00 0.00 -9.00 -8.00 -6.00 -5.00 -4.00 -3.00
-15.00 -14.00 0.00 -10.00 0.00 -8.00 0.00 0.00 0.00 -2.00
-14.00 -13.00 -12.00 -11.00 -10.00 -9.00 0.00 -3.00 -2.00 -1.00
-15.00 -14.00 -13.00 -12.00 0.00 -10.00 0.00 -2.00 -1.00
move


Policy Iteration
-17.00 -16.00 0.00 -10.00 0.00 -8.00 -7.00 -6.00 -5.00 -4.00
-16.00 -15.00 0.00 -9.00 -8.00 -7.00 -5.00 -4.00 -3.00
-15.00 -14.00 0.00 -10.00 0.00 -8.00 0.00 0.00 0.00 -2.00
-14.00 -13.00 -12.00 -11.00 -10.00 -9.00 0.00 -3.00 -2.00 -1.00
-15.00 -14.00 -13.00 -12.00 0.00 -10.00 0.00 -2.00 -1.00
move
Policy Iteration
-17.00 -16.00 0.00 -10.00 0.00 -8.00 -7.00 -6.00 -5.00 -4.00
-16.00 -15.00 0.00 -9.00 -8.00 -7.00 -6.00 -4.00 -3.00
-15.00 -14.00 0.00 -10.00 0.00 -8.00 0.00 0.00 0.00 -2.00
-14.00 -13.00 -12.00 -11.00 -10.00 -9.00 0.00 -3.00 -2.00 -1.00
-15.00 -14.00 -13.00 -12.00 0.00 -10.00 0.00 -2.00 -1.00
move


Policy Iteration
-17.00 -16.00 0.00 -10.00 0.00 -8.00 -7.00 -6.00 -5.00 -4.00
-16.00 -15.00 0.00 -9.00 -8.00 -7.00 -6.00 -5.00 -3.00
-15.00 -14.00 0.00 -10.00 0.00 -8.00 0.00 0.00 0.00 -2.00
-14.00 -13.00 -12.00 -11.00 -10.00 -9.00 0.00 -3.00 -2.00 -1.00
-15.00 -14.00 -13.00 -12.00 0.00 -10.00 0.00 -2.00 -1.00
move
Policy Iteration
-17.00 -16.00 0.00 -10.00 0.00 -8.00 -7.00 -6.00 -5.00 -4.00
-16.00 -15.00 0.00 -9.00 -8.00 -7.00 -6.00 -5.00 -4.00
-15.00 -14.00 0.00 -10.00 0.00 -8.00 0.00 0.00 0.00 -2.00
-14.00 -13.00 -12.00 -11.00 -10.00 -9.00 0.00 -3.00 -2.00 -1.00
-15.00 -14.00 -13.00 -12.00 0.00 -10.00 0.00 -2.00 -1.00
move


Policy Iteration
-17.00 -16.00 0.00 -10.00 0.00 -8.00 -7.00 -6.00 -5.00 -4.00
-16.00 -15.00 0.00 -9.00 -8.00 -7.00 -6.00 -5.00 -4.00 -3.00
-15.00 -14.00 0.00 -10.00 0.00 -8.00 0.00 0.00 0.00
-14.00 -13.00 -12.00 -11.00 -10.00 -9.00 0.00 -3.00 -2.00 -1.00
-15.00 -14.00 -13.00 -12.00 0.00 -10.00 0.00 -2.00 -1.00
move
Policy Iteration
-17.00 -16.00 0.00 -10.00 0.00 -8.00 -7.00 -6.00 -5.00 -4.00
-16.00 -15.00 0.00 -9.00 -8.00 -7.00 -6.00 -5.00 -4.00 -3.00
-15.00 -14.00 0.00 -10.00 0.00 -8.00 0.00 0.00 0.00 -2.00
-14.00 -13.00 -12.00 -11.00 -10.00 -9.00 0.00 -3.00 -2.00
-15.00 -14.00 -13.00 -12.00 0.00 -10.00 0.00 -2.00 -1.00
move



5. 결과에 대한 분석 (Discussion)

- 그림 2에서도 볼 수 있듯이, 각 state에서의 state value들이 optimal한 값으로 수렴했음을 알 수 있다. 이는 이전에 Monte Carlo Control에서 구했던 state value값들이 소수점 아래 값들이 존재했음을 고려한다면 더욱 향상된 성능임을 알 수 있다. 이러한 이유로는 Temporal Difference control은 매 순간의 state transition마다 behavior policy와 Q value (즉, target policy)를 업데이트 하기 때문에 optimal policy와 optimal state value로의 수렴이 더 빠르고 정확하게 이뤄진다는 해석이 가능하다. 그림 2에서는 greedy하게 target policy와 Q value를 업데이트 하면서 최종적으로 얻은 state value들을 나타낸다. 이를 통해 처음부터 greedy method를 통해 업데이트된 target policy는 곧바로 optimal policy로 수렴함을 확인할 수 있었다.
- 그림 3는 그림 2에서 구한 state value 및 target policy와는 별개로, behavior policy를 구한 결과를 plot한 모습이다. 이때 behavior policy역시 agent가 이동할 수 있는 최적의 이동 경로를 구할 수 있었음을 보여준다. 우리는 이 policy가 optimal policy의 조건을 만족함을 직관적으로 확인할 수 있다.
- 그림 3의 결과를 얻을 수 있었던 이유는 그림 4를 통해 설명이 가능하다. 그림 4에서는 생성된 episode의 개수에 따라 epsilon decay가 이뤄지는 모습을 print하여 관찰한 결과다. Epsilon decay period는 10000으로 설정되어 있었고, 총 50만개의 episode에 대한 반복을 거듭하며 결과적으로 epsilon이 0에 수렴하는 모습을 확인할 수 있다. 이를 통해, epsilon이 0에 수렴하도록 줄어들면서 epsilon-greedy method로 구했을 때 policy가 수렴하는 값은 optimal policy의 수렴값과 같아지게 된다는 사실을 검증했다. 따라서, behavior policy 역시 target policy와 같이 optimal policy에 수렴했음을 알 수 있다.
- 4번째 챕터에서 main_test.py 코드를 실행하여 실제 grid world에서 agent의 움직임을 확인한 결과, Q-learning을 구현한 코드를 통해 얻은 policy에 따라, agent가 장애물을 피해 정확히 최단 경로를 거쳐 terminal state에 도달하는 것을 확인할 수 있다. 이를 통해 Q-learning으로 찾은 value function과 policy가 optimal한 결과를 보여준다는 사실을 검증하였다.