숙제 5 (First Visit Monte Carlo Method)

다음의 7 × 7 Grid Map에서 출발점, 목표지점, 장애물이 각각 표시된 위치에 있을 때, 목표지점을 찾아가는 행동을 **First Visit Monte Carlo 방법**으로 구하고 학습 진행(epoch)에 따를 결과(학습된 각 상태의 상태함수값 및 정책에 의한 행동 확률을 격자 위에 표시)를 제시하시오. (제출: 6월 3일 6시까지, Google Classroom)

위 문제는 몬테카를로 방법 중 first visit 방법이다. 몬테카를로 방법을 간단히 설명하자면 가능한 많은 시행을 해서 얻은 리턴값들의 평균값과 이론적으로 분석해서 나온 값과 같아지는 것을 말한다.
몬테카를로에서 가장 중요한 식을 소개하겠다.
평균 구하는 방법:

$$V(s) \leftarrow V(s) + \alpha(G(s) - V(s))$$

이전까지의 평균 + 1/n(현재 시행한 샘플의 평균 + 이전까지의 평균의 차)
이 식의 의미하는 바는 현재 시행한 샘플의 평균을 더해 평균을 구할 수 있다는 것이다.
이제 First vist를 간단히 설명하자면 동선이 겹칠 때 최초로 방문한 이후의 return을 한 번만 허용하는 것이다.
아래 사진은 Fisrt visit의 알고리즘이다.
아래 알고리즘에서 필요한 것은 리턴값을 구하는 것이다. 리턴을 구하기 위해 에피소드를 맨 뒤에서부터 구한다. 여기서 중요한 것은 뒤에 상태에 리턴값으로 앞에 반환값을 구하는 것이다. ex) 마지막 10 일 때, 반환값 G9 = return값 R10, G8 = R9 + r*G9,
또 중요한 것은 first visit이라는 것을 알아야 된다. 이후 리턴값을 계속 받아줘야 한다.(뒤에서부터 받고 있기 때문에 계속 받아야 맨 처음에 리턴값을 받을 수 있다.)

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

Input: a policy π to be evaluated 정책
Initialize:
  $V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$
  $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$ 비어있는 배열 만들기

Loop forever (for each episode): 임의의s에 대해 에피소드 생성
  Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$: 맨 뒤부터 봄, 뒤에서 구한 값을 이용해 q함수 구할 수
    $G \leftarrow \gamma G + R_{t+1}$
    Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$: 첫번째 값이면 리턴
      Append G to $Returns(S_t)$
      $V(S_t) \leftarrow$ average$(Returns(S_t))$

이번 과제는 몬테카를로를 이용하여 최적의 경로를 찾아 목표 지점에 찾아가는 행동을 하는 것이다.
주어진 코드에는 알기 어려운 부분이 많았으므로 고쳐서 돌려보았다.
결과를 보여주기 전 고친 코드에 대해 설명을 하겠다.

1) reward 리스트

```
class Env(tk.Tk):
    def __init__(self):
        super(Env, self).__init__()
        self.action_space = ['u', 'd', 'l', 'r']
        self.n_actions = len(self.action_space)
        self.title('monte carlo')
        self.geometry('{0}x{1}'.format(HEIGHT * UNIT, HEIGHT * UNIT))
        self.shapes = self.load_images()
        self.canvas = self._build_canvas()
        self.texts = []
        self.reward = [[0] * 7 for _ in range(7)] #reward 값 넣을 공간 초기화
        self.reward[3][4] = 5  # (3,4) 좌표 동그라미 위치에 보상 5
        self.reward[0][4] = -1
        self.reward[1][2] = -1
        self.reward[2][4] = -1
        self.reward[3][3] = -1
        self.reward[3][5] = -1
        self.reward[4][2] = -1
        self.reward[5][3] = -1
##############################################################################
```

```
    # 보상 함수
    if next_state == self.canvas.coords(self.circle): #목표지점인가?
        reward = 5
        done = True
    elif next_state in [self.canvas.coords(self.triangle1), #장애물 위치인가
                        self.canvas.coords(self.triangle2),
                        self.canvas.coords(self.triangle3),
                        self.canvas.coords(self.triangle4),
                        self.canvas.coords(self.triangle5),
                        self.canvas.coords(self.triangle6),
                        self.canvas.coords(self.triangle7)]:
        reward = -1
        done = True
    else:
        reward = 0
        done = False
```

첫 번째로 고친 부분은 reward list를 만든 것이다. get_action이라는 함수가 뒤에 나올텐데 그때 리워드의 값을 사용하기 위해 만들어주었다. 아래 사진은 터미널 스테이트와 세모를 만나면 다시 시작하도록 해주는 코드이다.

2) 리워드 값을 구하는 함수

```
##############################################################################
    def get_reward(self, state, action):
        next_state = self.state_after_action(state, action)
        return self.reward[next_state[0]][next_state[1]]

    def state_after_action(self, state, action_index):
        action = ACTIONS[action_index]
        return self.check_boundary([state[0] + action[0], state[1] + action[1]])

    @staticmethod
    def check_boundary(state):
        state[0] = (0 if state[0] < 0 else WIDTH - 1
        if state[0] > WIDTH - 1 else state[0])
        state[1] = (0 if state[1] < 0 else HEIGHT - 1
        if state[1] > HEIGHT - 1 else state[1])
        return state
```

두 번째로 고친 부분은 리워드 값을 구하는 부분이다. 이 또한 get_action 부분에서 R+r*V(S')에서 사용될 reward 값을 얻기 위해 에전에 공부했던 value-iteration에 environment에 있는 함수를 가져와 사용하였다.

3) get_action

```
def get_action(self, state): #지금 상태에서 입실론 그리디 정책으로 행동을 정함
    reward = []
    q = []
    if np.random.rand() < self.epsilon: #랜덤값이 정해진 초기 확률보다 낮을 때
        # 랜덤 행동
        action = np.random.choice(self.actions) #랜덤한걸 선택한다
    else:
        # 큐 함수 사용
        for i in range(4):
            reward.append(env.get_reward(state, i)) # reward 찾음
        next_state = self.possible_next_state(state) #그리디한 정책을 찾는다, 다음상태의 벨류값
        #next_value = np.array(next_state) # 값 구할 수 있게 변환
        for i in range(4):
            q.append(reward[i] + self.discount_factor * next_state[i]) # 큐 사용 이제 액션 구해야됨
        action = self.arg_max(q)

    return int(action)
#########################################################################################################
```

get_action 함수는 입실론 그리디 정책으로 행동을 정하는 함수이다. 고친 부분은 else부분이다. else부분을 보면 아까 앞서 2번에서 받아온 함수로 상하좌우의 reward 값을 받아온다. 이후의 value값을 받아서 R+r*V(S') 식을 이용하여 큐함수를 구한다. 여기서 나온 값을 이용하여 arg_max()함수를 이용해 어느 방향이 큐값이 큰지 알 수 있다.

4) 그리디 맥스값

```
def max(self,next_state): #맥스값 구하기
    max_index_list = [] #상하좌우 빈 리스트
    max_value = next_state[0] #상을 취했을 때
    for index, value in enumerate(next_state): #인덱스 벨류
        if value > max_value:
            max_index_list.clear()
            max_value = value
            max_index_list.append(index)
        elif value == max_value: #맥시멈이 하나가 아닐 수 있음
            max_index_list.append(index)
    return max_index_list
```

앞에서 사용한 arg_max()함수에서 return만 max_index_list로 바꾸어 주었다. update 함수에서 사용된다.

5)update 함수

```
def update(self):
    m = []
    G_t = 0
    visit_state = []
    for reward in reversed(self.samples): #리스트들 순서를 뒤집음
        # 에이전트 입장에서 결과의 스테이트 가지고 업데이트, 현재로 업데이트 해야됨
        state = str(reward[0]) # 좌표값
        G_t = reward[1] + self.discount_factor * G_t #감가율 적용되는거라 무조건 계산해야되는 것임 띄어 넣어가면 안됨, Rt+1
        value = self.value_table[state] #다음 스테이트 벨류 받아 업데이트, 현재로 해야됨
        if state not in visit_state: #당연히 없을때 업데이트 하겠다, 근데 순서가 달라짐 뒤에 있는 value가 업데이트됨
            visit_state.append(state)
            self.value_table[state] = (value + self.learning_rate * (G_t - value)) #이전 스테이트로 계산 해야됨
            v = env.text_value(int(state[4]), int(state[1]), round(G_t, 2))
            a = env.text_action(state, self.max(self.possible_next_state(reward[0])), self.epsilon)
```

이 코드는 다 고치지 못하였다. 고친 부분은 G_t와 value 값을 if문 위로 뺀 것이다. 이렇게 if문에 들어가 있어 업데이트 되지 못했던 값들을 밖으로 뺌으로서 계속해서 업데이트 되게 한다.

대신 설명을 덧붙이자면. 함수의 리턴값을 구해서 반환값을 구하는 것인데 앞서 문제를 설명하기 전에 몬테카를로에 대해 설명할 때 썼던 내용이다.

리턴을 구하기 위해 에피소드를 맨 뒤에서부터 구한다. 여기서 중요한 것은 뒤에 상태에 리턴값으로 앞에 반환값을 구하는 것이다.

ex) 마지막 10 일 때, 반환값 G9 = return값 R10, G8 = R9 + r*G9

이와 같은 일을 하는 함수이다.

처음에는 if문을 지우려고 했으나 그렇게 되면 every visit 함수가 되어 버리므로 처음가는 부분을 구하는 방법을 어떻게 찾을까 고민하던 중 앞에서부터 처음 가는 곳을 미리 찾아서 거꾸로 바꾸어 구할 때 처음 가는 부분이 아닌 부분을 건너띄면 어떨까라는 생각을 하였다. 물론 구현을 하지 못하였다.

6) 출력하는 부분

```
#벨류
def text_value(self, row, col, contents, font='Helvetica', size=10,
               style='normal', anchor="nw"):
    origin_x, origin_y = 80, 70
    x, y = origin_y + (UNIT * col), origin_x + (UNIT * row)
    font = (font, str(size), style)
    text = self.canvas.create_text(x, y, fill="black", text=contents, font=font, anchor=anchor)
    return self.texts.append(text)
#정책
def text_action(self, state, action, epsilon):
    list = [epsilon/4, epsilon/4, epsilon/4, epsilon/4]
    for i in action:
        list[i] += (1 - epsilon)/len(action)

    txt = "     " + format(list[0], ".3f") + "\n" + format(list[2], ".3f") + "    " \
          + format(list[3], ".3f") + "\n     " + format(list[1], ".3f")
    text = self.canvas.create_text(int(state[1])*100 + 50, int(state[4])*100 + 50, text = txt)
    return self.texts.append(text)
#에폭
def text_epoch(self, cnt):
    txt = 'epoch : ' + str(cnt)
    text = self.canvas.create_text(40, 10, text=txt)
    return text
#지우기
def delete_print(self, label):
    self.canvas.delete(label)
```

1) 벨류를 출력하는 함수는 value_iteration에 있는 함수를 얻어썼다.
2) 입실론: 기존 정책에 따른 행동을 하지 않을 확률 ex) 0.2면 0.8 확률로 큐함수 값 중 제일 큰 행동을 한다. 나머지 중에서 랜덤으로 0.2의 확률로
이 두 개는 self.texts.append(text)를 해주어서 바로바로 초기화될 줄 알았는데 초기화가 안되었다. 그래서 3번코드는 다르게 만들어보았다.
3) 에폭을 출력하는 함수이다.
4) 지우는 함수이다. 변수에 에폭함수값을 넣은 후 이 함수에 넣으면 값이 지워진다.

```
    v = env.text_value(int(state[4]), int(state[1]), round(G_t, 2))
    a = env.text_action(state, self.max(self.possible_next_state(reward[0])), self.epsilon)
```

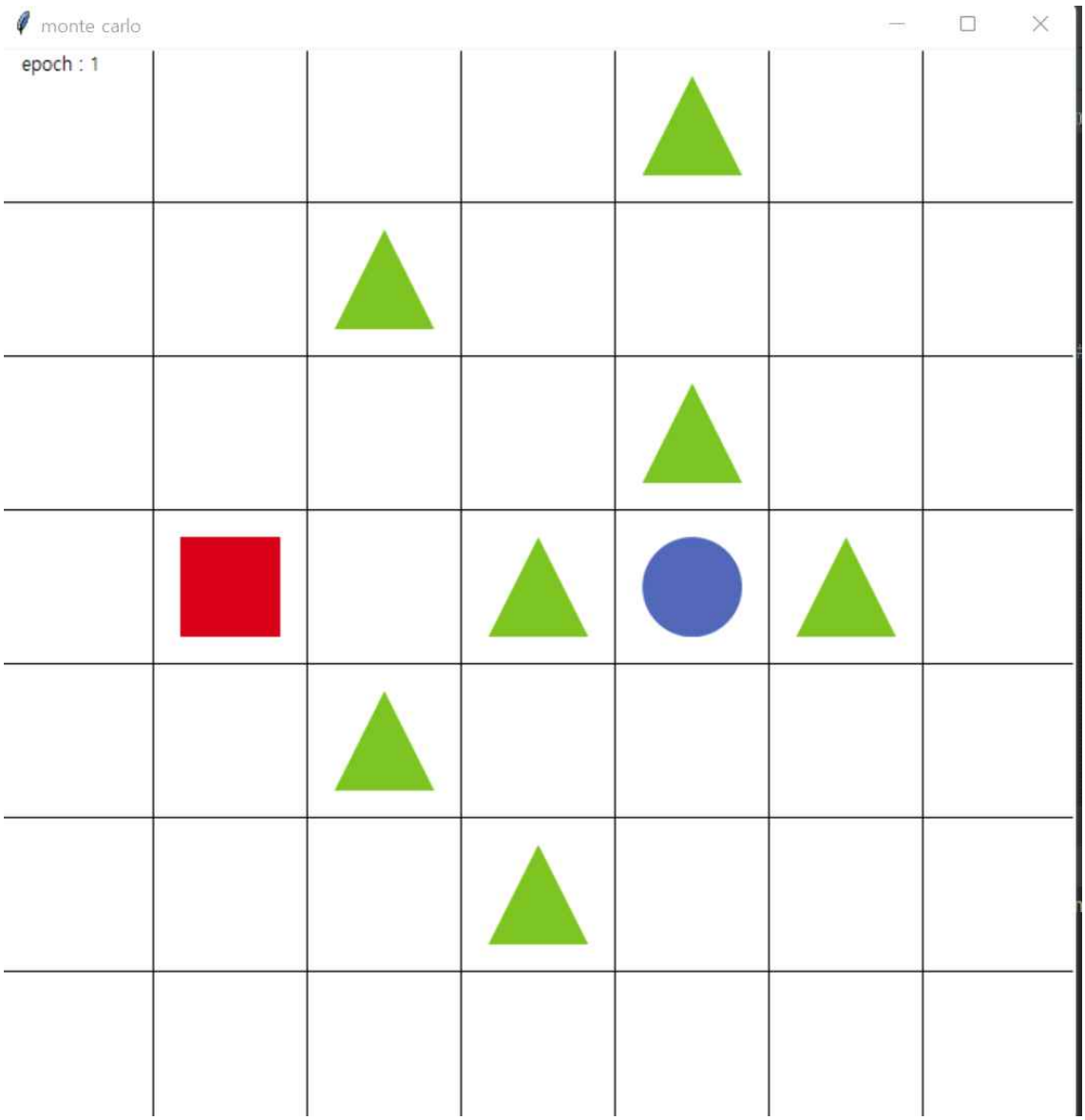
1,2번 함수를 돌리는 과정이다.

```python
for episode in range(4):
    epoch = env.text_epoch(episode+1)
    state = env.reset() #0.0
    action = agent.get_action(state)
    while True:
        env.render() #표시할 그래프 이미지를 업데이트

        # 다음 상태로 이동
        # 보상은 숫자이고, 완료 여부는 boolean
        next_state, reward, done = env.step(action) #액션을 받고 리워드, 상태변화, 종료 되었는지 받음
        agent.save_sample(next_state, reward, done) #리스트로 만들어 추가해줌

        # 다음 행동 받아옴
        action = agent.get_action(next_state) #다음 상태 넣고 와일문 돎, reward 추가해줌

        #삭제

        # 에피소드가 완료됐을 때, 큐 함수 업데이트
        if done:
            # 모든 벨류,큐함수를 화면에 표시
            agent.update() #value fuction update
            agent.samples.clear()
            break
    env.delete_print(epoch)
```

3,4번 함수를 사용하는 과정이다.

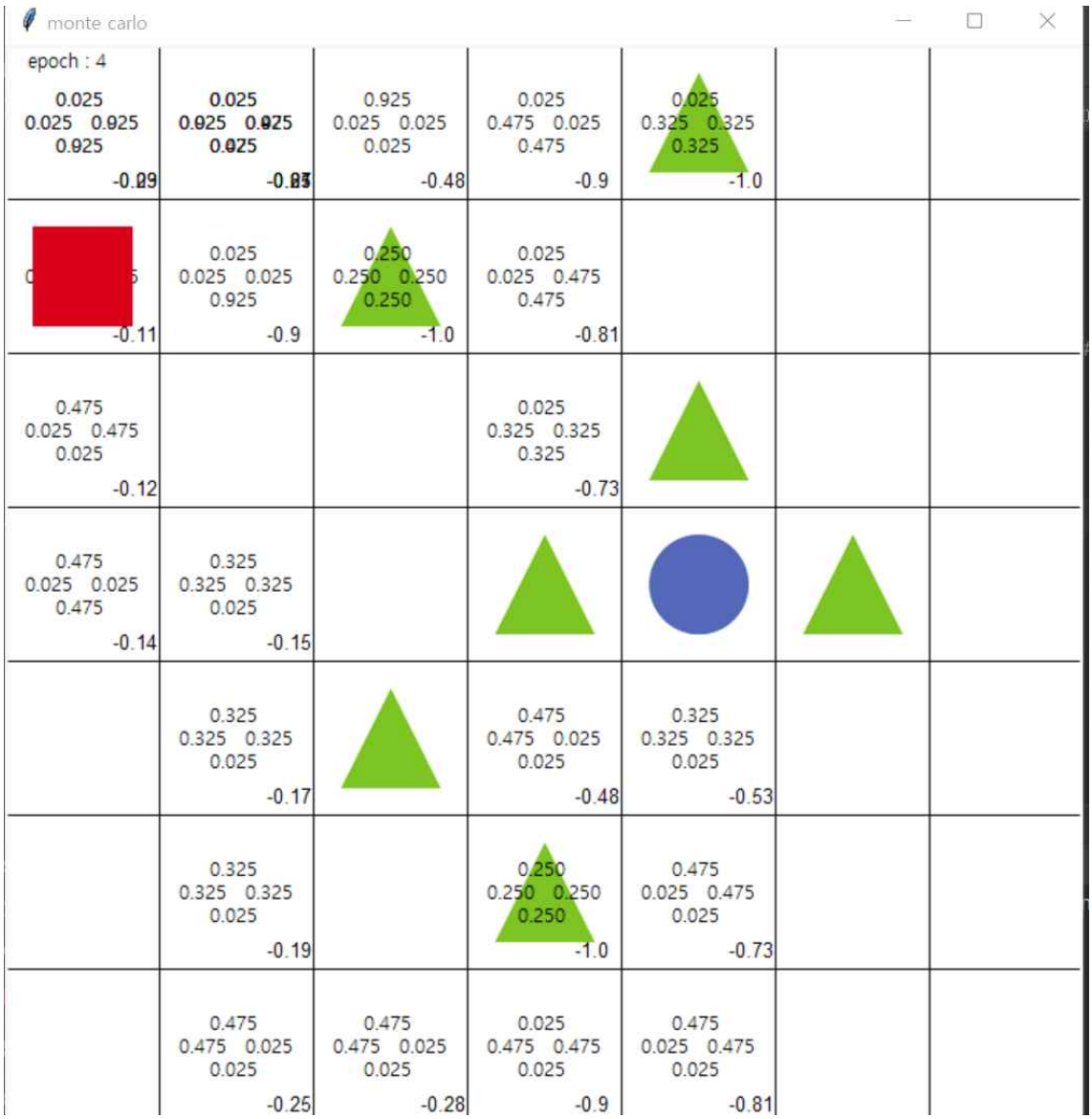
이제 결과값을 출력해 보겠다.

1) epoch:1 일 때



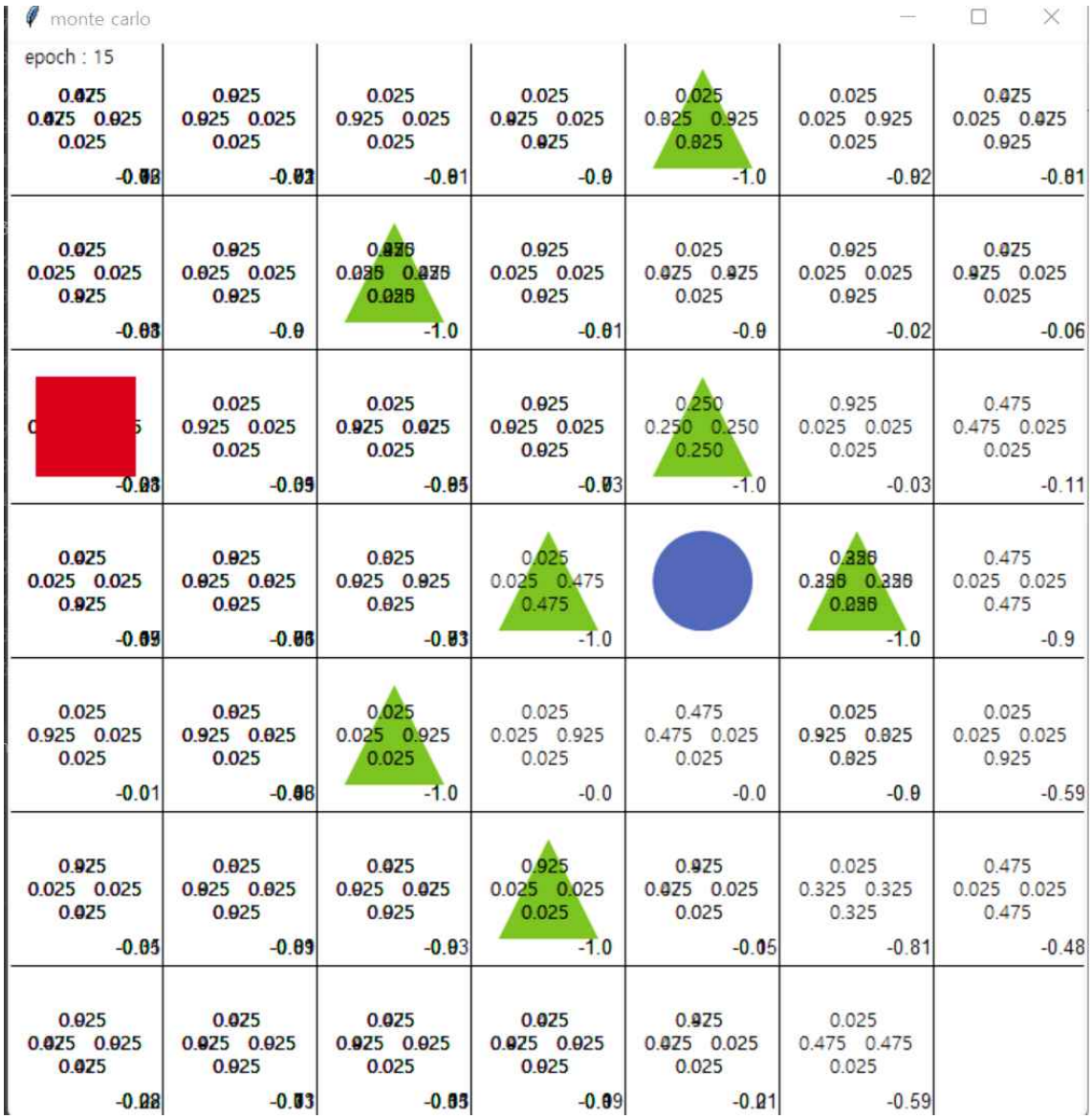
처음 모습이다.

2) epoch:4 일 때



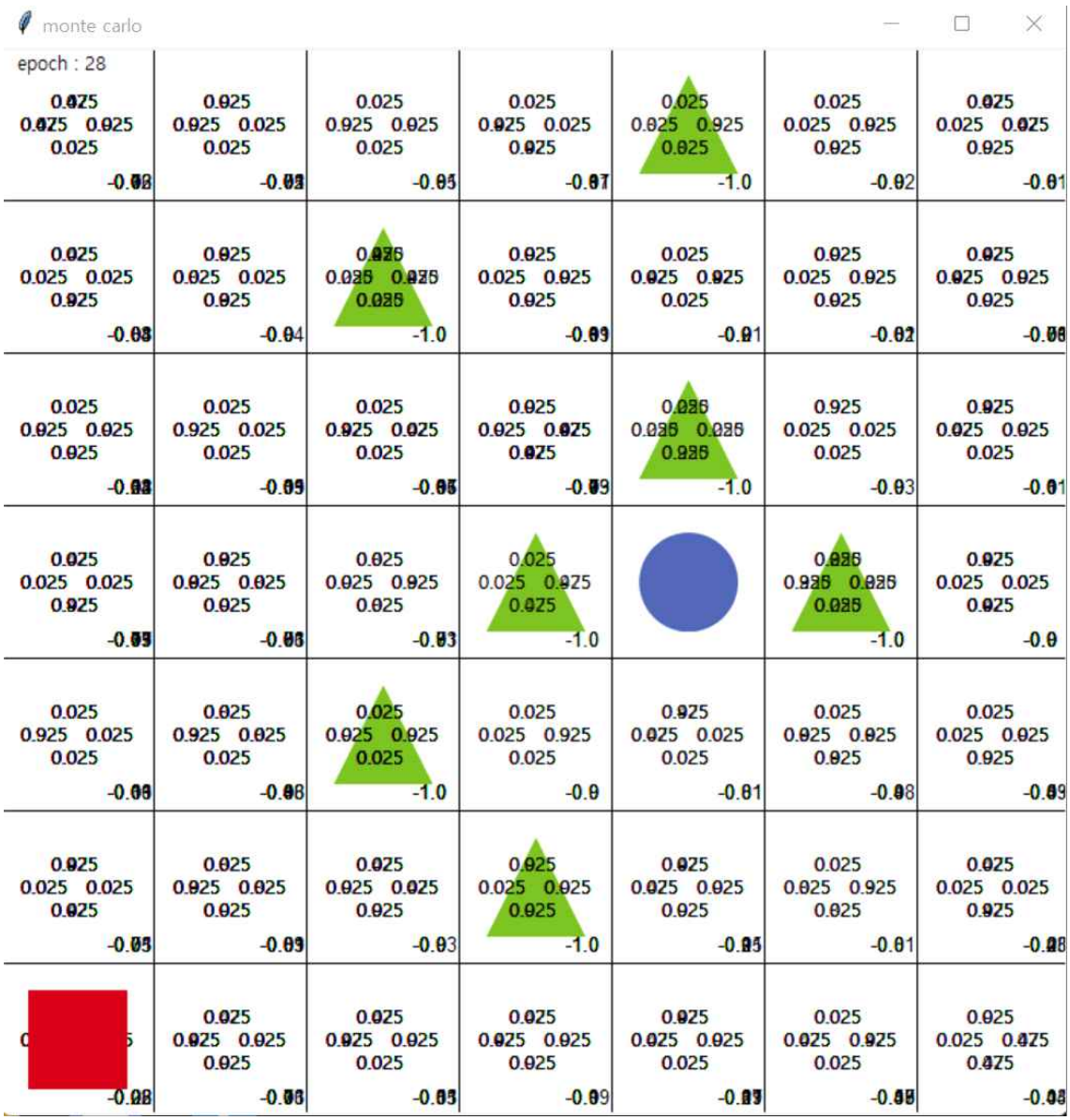
처음엔 잘 나오다가 도중부터 결과가 겹쳐서 나온다.

3) epoch: 15일 때



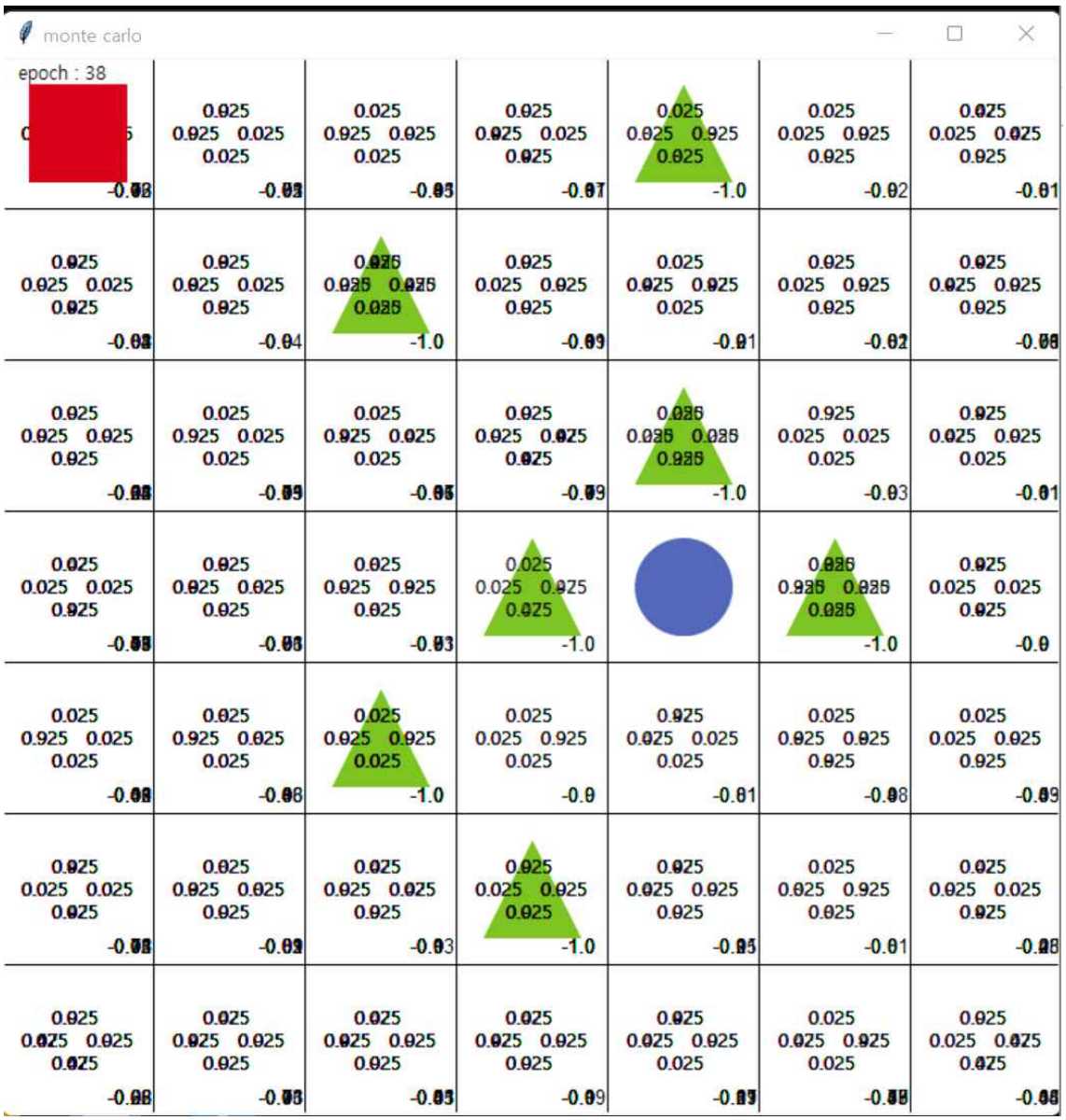
아직까지 터미널 스테이트까지 도달하지 못하였다.

4) epoch: 28일 때



대부분의 경로를 지나갔다.

5) epoch: 38일 때



제출시간이 되어 시간관계상 38번까지 밖에 돌리지 못하였다. 목적지에는 도착하지 못하였지만 계속해서 하다보면 아마 100번째 안에 도착할거 같다.