5x5 Grid World

Monte Carlo Method 를 적용하여 5x5 Grid World 에 대한 Q 함수를 시각화하고 및 policy 를 구하라.

- (1) 파라메터 ε 을 변경하며 결과 비교
- (2) 파라메터 α 를 변경하며 결과 비교

GridWorld class

```
In [22]:
          import numpy as np
          import common.gridworld5_render as render_helper
          class GridWorld:
             def __init__(self):
                 self.action_space = [0, 1, 2, 3] # 행동 공간(가능한 행동들)
                  self.action meaning = { # 행동의 의미
                     0: "UP",
                     1: "DOWN"
                     2: "LEFT",
                     3: "RIGHT",
                  }
                 self.reward_map = np.array( # 보상 맵(각 좌표의 보상 값)
                      [[0, 0, 0, -1.0, 1.0],
                       [0, 0, 0, 0, 0],
                       [0, None, None, 0, 0],
                       [0, 0, 0, 0, -1.0],
                       [0, 0, 0, 0, 0]
                       ]
                  )
                  self.goal_state = (0, 4) # 목표 상태(좌표)
                  self.wall_state = [(2, 1), (2,2)] # 2,1 2,2 # 벽 상태(좌표)
                  self.start_state = (4, 0) # 시작 상태(좌표)
                  self.agent_state = self.start_state # 에이전트 초기 상태(좌표)
             @property
              def height(self):
                  return len(self.reward map)
             @property
             def width(self):
                  return len(self.reward_map[0])
             @property
             def shape(self):
                  return self.reward_map.shape
              def actions(self):
                  return self.action_space
             def states(self):
                  for h in range(self.height):
                      for w in range(self.width):
                         yield (h, w)
```

```
def next state(self, state, action):
   # 이동 위치 계산
   action_move_map = [(-1, 0), (1, 0), (0, -1), (0, 1)]
   move = action move map[action]
   next_state = (state[0] + move[0], state[1] + move[1])
   ny, nx = next_state
   # 이동한 위치가 그리드 월드의 테두리 밖이나 벽인가?
   if nx < 0 or nx >= self.width or ny < 0 or ny >= self.height
       next state = state
   elif next_state == self.wall_state[0] or next_state == self
       next_state = state
    return next_state # 다음 상태 반환
def reward(self, state, action, next_state):
    if self.reward map[next state] == None:
        return 0
    return self.reward_map[next_state]
def reset(self):
    self.agent_state = self.start_state
    return self.agent_state
def step(self, action):
    state = self.agent_state
    next_state = self.next_state(state, action)
    reward = self.reward(state, action, next_state)
   done = (next_state == self.goal_state)
    self.agent_state = next_state
    return next_state, reward, done
def render_v(self, v=None, policy=None, print_value=True):
    renderer = render helper.Renderer(self.reward map, self.goa
                                      self.wall state)
    renderer.render_v(v, policy, print_value)
def render_q(self, q=None, print_value=True):
    renderer = render_helper.Renderer(self.reward_map, self.goa
                                      self.wall_state)
    renderer.render_q(q, print_value)
```

Policy Evaluation

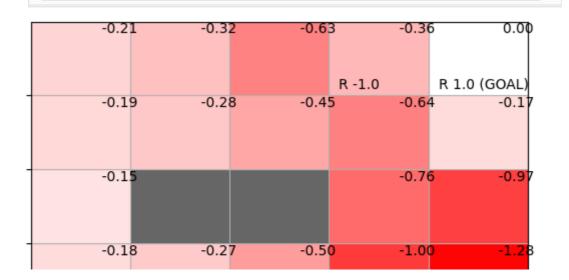
```
In [26]:
    from collections import defaultdict
import numpy as np

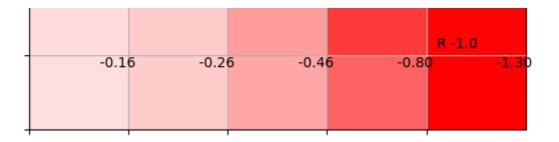
class RandomAgent:
    def __init__(self):
        self.gamma = 0.9
        self.batch_size = 4

        random_actions = {0: 0.25, 1:0.25, 2:0.25, 3:0.25}
        self.pi = defaultdict(lambda: random_actions)
        self.V = defaultdict(lambda: 0)
        self.cnts = defaultdict(lambda: 0)
        self.memory = []
```

```
uci get_action(set), state/.
    action_probs = self.pi[state]
    actions = list(action_probs.keys())
    probs = list(action_probs.values())
    return np.random.choice(actions, p=probs)
def add(self, state, action, reward):
    data = [state, action, reward]
    self.memory.append(data)
def reset(self):
    self.memory.clear()
def eval(self):
    G = \emptyset
    for data in reversed(self.memory): # 역방향으로(reversed) 따리
        state, action, reward = data
        G = self.gamma * G + reward
        self.cnts[state] += 1
        self.V[state] += (G - self.V[state]) / self.cnts[state]
```

```
In [27]:
         env = GridWorld()
         agent = RandomAgent()
         episodes = 1000
         for episode in range(episodes):
             state = env.reset()
             agent.reset()
             while True:
                 action = agent.get_action(state)
                                                 # 행동 선택
                 next_state, reward, done = env.step(action) # 행동 수행
                agent.add(state, action, reward) # (상태, 행동, 보상) 저
                 if done: # 목표에 도달 시
                    agent.eval() # 몬테카를로 방식으로 가치 함수 갱신
                                 # 다음 에피소드 시작
                    break
                 state = next_state
         env.render_v(agent.V)
```





Policy control

```
In [30]:
          import numpy as np
          from collections import defaultdict
          #from common.gridworld import GridWorld
          def greedy_probs(Q, state, epsilon=0, action_size=4):
              qs = [Q[(state, action)] for action in range(action_size)]
              max_action = np.argmax(qs)
              base prob = epsilon / action size
              action_probs = {action: base_prob for action in range(action_si
              action_probs[max_action] += (1 - epsilon)
              return action_probs
          class McAgent:
              def __init__(self, gamma=0.9, epsilon=0.1, alpha=0.1, action_si
                  self.gamma = gamma
                  self.epsilon = epsilon
                  self.alpha = alpha
                  self.action_size = action_size
                  random_actions = {0:0.25, 1:0.25, 2:0.25, 3:0.25}
                  self.pi = defaultdict(lambda: random_actions)
                  self.Q = defaultdict(lambda: 0)
                  self.memory = []
              def get_action(self, state):
                  action_probs = self.pi[state]
                  actions = list(action_probs.keys())
                  probs = list(action_probs.values())
                  return np.random.choice(actions, p=probs)
              def add(self, state, action, reward):
                  data = (state, action, reward)
                  self.memory.append(data)
              def reset(self):
                  self.memory.clear()
              def update(self):
                  G = 0
                  for data in reversed(self.memory):
                      state, action, reward = data
                      G = self.gamma * G + reward
                      key = (state, action)
                      self.Q[key] += (G-self.Q[key]) * self.alpha
                      self.pi[state] = greedy_probs(self.Q, state, self.epsil
```

```
In [33]:
          env = GridWorld()
          # Parameters
          gamma = 0.9
          epsilon = 0.1
          alpha = 0.1
          action_size = 4
          agent = McAgent(gamma, epsilon, alpha, action_size)
          episodes = 10000
          for episode in range(episodes):
              state = env.reset()
              agent.reset()
              while True:
                  action = agent.get_action(state)
                  next_state, reward, done = env.step(action)
                  agent.add(state, action, reward)
                  if done:
                      agent.update()
                      break
                  state = next_state
          env.render_q(agent.Q)
```

0.00	0.00	-0.07	-0.03	
-0.02 0.08	8 -0.03 0.10	0 -0.11 -0.0	2 0.19 1.0	b
0.00	-0.02	0.40	R-1.00.33	R 1.0 (GOAL)
0.05	-0.01	0.11	-0.21	1.00
0.00 0.00	0.00 0.1	7 0.33 0.6	1 0.55 0.9	0.73 0.90
0.00	0.08	0.01	0.70	0.80
0.05			0.81	0.87
0.00 0.03	3		0.64 0.6	6 0.55 0.37
0.08			0.61	-0.32
0.07	0.29	0.17	0.72	0.77
0.01 0.1	7 0.24 0.5	4 0.33 0.6	1 0.32 -0.4	5 0.24 -0.38
0.30	0.09	0.31	0.47	R-1.00.30
0.27	0.47	0.39	0.47	-0.50
0.13 0.43	3 0.28 0.2	5 0.28 0.2	7 0.35 0.19	9 0.29 -0.00
0.20	0.20	0.00	0.37	0.02

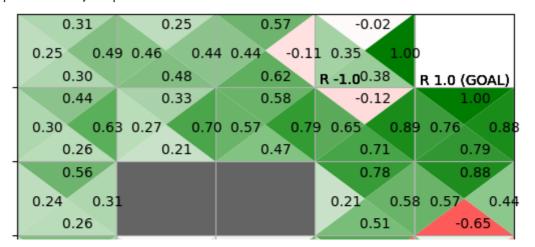
→	→	1	→ R -1.0	R 1.0 (GOAL)
Ť.	→	→	→	1
			_	

1			1	1
↓	→	→	†	1
				R -1.0
→	1	1	1	←

epsilon, alpha 값에 따른 결과 비교

```
In [34]:
          # Parameters
          epsilon_values = [0.1, 0.3, 0.5]
          alpha_values = [0.1, 0.3, 0.5]
          episodes = 10000
          for epsilon in epsilon values:
              for alpha in alpha_values:
                  env = GridWorld()
                  agent = McAgent(gamma=0.9, epsilon=epsilon, alpha=alpha, ad
                  for episode in range(episodes):
                      state = env.reset()
                      agent.reset()
                      while True:
                          action = agent.get_action(state)
                          next_state, reward, done = env.step(action)
                          agent.add(state, action, reward)
                          if done:
                              agent.update()
                              break
                          state = next_state
                  # Visualization
                  print(f"Epsilon={epsilon}, Alpha={alpha}")
                  env.render_q(agent.Q)
```

Epsilon=0.1, Alpha=0.1



0.50	-0.02	0.06	0.65	0.08
0.25 0.32	2 0.19 0.1	1 0.02 0.5	5 0.28 -0.5	7 0.54 -0.34
0.22	0.33	0.17	0.16	R-1.0 ^{0.24}
0.44	0.15	0.48	0.18	-0.28
0.20 0.2	5 0.04 0.4	3 0.12 -0.0	0 -0.04 -0.14	4 0.08 0.01
0.23	0.07	-0.02	0.01	-0.00

→	ţ	ţ	→ R -1.0	R 1.0 (GOAL)
→	→	→	→	†
Ť.			Ť	1
†	1	→	†	← R -1.0
Ť.	→	†	†	←

Epsilon=0.1, Alpha=0.3

0.10	0.02	0.12	0.22	
0.10	0.02	0.12	-0.23	
0.02 0.3			4 0.34 1.0	
0.03	0.00	0.63	R-1.00.61	R 1.0 (GOAL)
0.32		0.50		1.00
0.01 0.5	9 0.43 0.6	8 0.57 0.7	7 0.70 0.8	6 0.80 0.90
0.11	0.31	0.37	0.73	0.79
0.53			0.56	0.90
0.21 0.0	5		0.31 0.7	6 0.64 0.80
0.25			0.37	-0.33
0.48	0.03	0.00	0.37	0.80
0.19 0.1	1 0.00 0.0	0 0.05 0.2	9 0.21 -0.3	2 0.35 -0.3
0.27	0.01	0.05	0.15	R-1.0 ^{0.19}
0.42	0.00	0.14	0.16	-0.40
0.00 0.00	0 0.02 0.0	6 0.07 0.0	7 0.09 -0.0	0.00 0.08
0.05	0.00	0.00	0.07	-0.02

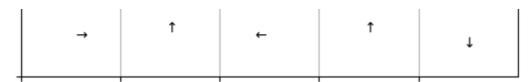
→	→	1	→ R -1 0	R 1 0 (GOAL)
----------	----------	---	-------------	--------------

				,,
→	→	→	→	†
1			→	†
1	†	→	†	↑ R -1.0
1	→	Ť	†	→

Epsilon=0.1, Alpha=0.5

0.00	0.47	-0.10	
4 0.35 0.2	7 -0.03 -0.1	8 0.50 1.0	b
0.43	0.30	R -1.0 ^{0.17}	R 1.0 (GOAL)
0.56	0.56	-0.10	1.00
4 0.02 0.13	2 0.45 0.3	4 0.70 0.90	0.81 0.86
0.05	0.64	0.56	0.80
		0.81	0.90
7		0.36 0.3	3 0.52 -0.68
		0.36	-0.66
-0.08	-0.00	0.71	0.13
5 0.11 0.5	6 -0.06 0.6	2 0.39 -0.7	9 -0.08 -0.71
0.36	0.20	-0.02	R-1.0 ^{0.21}
0.47	-0.03	0.01	-1.06
1 0.00 0.2	2 0.36 -0.1	1 -0.12 -0.6	4 -0.04 -0.98
-0.00	-0.02	-0.10	-0.01
	0.43 0.56 4 0.02 0.05 7 -0.08 5 0.11 0.36 0.47 1 0.00 0.22	0.43 0.30 0.56 0.56 0.56 0.56 0.05 0.64 0.02 0.12 0.45 0.36 0.05 0.64 0.36 0.20 0.47 -0.03 0.20 0.47 -0.03 0.20 0.47 -0.03 0.20 0.47 -0.03 0.20 0.47 -0.03 0.20 0.47 -0.03 0.20 0.47 -0.03 0.20 0.47 -0.03 0.20 0.47 -0.03 0.20 0.47 -0.03 0.20 0.47 -0.03 0.20 0.47 -0.03 0.20 0.47 -0.03 0.20 0.47 -0.03 0.20 0.47 -0.03 0.20 0.20 0.47 -0.03 0.20 0.20 0.20 0.20 0.20 0.20 0.20	4 0.35 0.27 -0.03 -0.18 0.50 1.00 0.43 0.30 R -1.00.17 -0.10 0.56 0.56 -0.10 4 0.02 0.12 0.45 0.34 0.70 0.90 0.05 0.64 0.56 0.36 0.31 0.36 0.33 0.36 0.00 0.71 0.79 0.36 0.20 -0.02 0.47 -0.03 0.01 1 0.00 0.22 0.36 -0.11 -0.12 -0.64

î	↓	↑	→	
			R -1.0	R 1.0 (GOAL)
→	†	1	→	†
→			†	Ť
→	→	→	1	1
				R -1.0

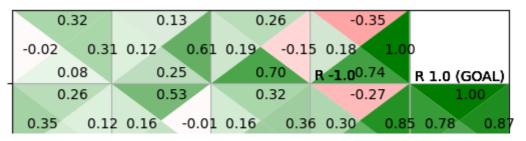


Epsilon=0.3, Alpha=0.1

0.24	0.18	0.21	-0.28	
0.17 0.28			3 0.32 1.00	
0.08	0.25	0.29	R -1.0 ^{0.52}	R 1.0 (GOAL)
0.24		0.17		1.00
0.11 0.0	7 0.12 0.2	7 0.23 0.4	7 0.48 0.8	5 0.73 0.84
0.20	0.18	-0.08	0.47	0.68
0.14			0.41	0.87
0.10 0.18	8		0.39 0.59	9 0.45 0.55
0.09			0.20	-0.62
0.05	0.13	0.36	0.50	0.66
0.03 0.03	3 0.03 0.2	3 0.05 0.20	6 -0.00 -0.54	4 0.38 -0.60
0.04	-0.00	0.09	-0.01	R-1.0 ^{0.06}
0.01	0.15	0.11	-0.04	-0.50
0.10 0.1	7 0.08 0.0	3 0.02 0.03	2 0.05 -0.02	2 0.01 0.02
0.06	0.04	0.05	-0.05	-0.03

→	Ţ	1	→ R -1.0	R 1.0 (GOAL)
↑	→	→	→	1
→			→	†
†	→	Ť	†	↑ R -1.0
→	†	†	←	→

Epsilon=0.3, Alpha=0.3



0.16	0.23	0.36	0.59	0.68
0.09			0.47	0.85
0.05 0.03	2		0.45 0.74	4 0.56 0.49
0.06			0.33	-0.48
0.05	0.21	-0.02	0.66	0.71
0.04 0.1	1 0.02 0.1	5 0.18 0.5	5 0.42 -0.4	4 0.35 -0.68
0.06	0.24	0.27	0.12	R-1.0 ^{0.09}
0.01	0.05	0.46	-0.04	-0.64
0.00 0.30	0.04 0.3	7 0.20 0.2	3 0.33 -0.24	4 -0.19 -0.19
0.02	0.01	0.15	-0.08	-0.16

Î	→	ţ	→ R -1.0	R 1.0 (GOAL)
←	†	1	→	†
Ť.			→	†
→	1	→	†	↑ R -1.0
→	→	†	←	ţ

Epsilon=0.3, Alpha=0.5

	-0.04	-0.05	-0.04	-0.10	
	-0.16 -0.03	3 0.01 -0.1	1 0.03 -0.3	5 -0.03 1.00	b
	-0.07	-0.02	0.23	R -1.0 ^{0.20}	R 1.0 (GOAL)
	-0.04			-0.10	
	-0.01 -0.0	1 0.01 -0.0	3 -0.07 0.49	9 0.27 0.58	3 0.62 0.74
	0.03	0.06	0.61	0.27	0.19
	0.02			0.38	0.85
	0.04 -0.0	0		0.24 0.45	5 0.62 0.74
	0.08			0.14	-0.51
	0.04	0.05	0.05	0.22	0.65
	-0.01 0.00	0 -0.00 0.1	3 -0.01 0.1	6 0.00 -0.5	3 0.20 -0.5 6
	0.03	-0.02	0.00	0.02	R-1.0 ^{0.27}
-	-0.00	0.11	-0.03	-0.04	-0.87
	-0.01 0.10	0 0.00 -0.1	5 0.00 -0.0	7 -0.08 -0.4	5 -0.20 -0.03
	0.00	0.00	0.01	-0.07	7 7

+				
<u> </u>		<u>'</u>		<u> </u>
→	←	1	→ R -1.0	R 1.0 (GOAL)
1	1	ļ	→	1
ţ			→	1
1	→	→	†	↑ R -1.0
→	†	ļ	†	→

Epsilon=0.5, Alpha=0.1

0.07	-0.12	-0.01	-0.39	
0.14 0.23	0.07 0.10	0 0.15 -0.5	5 0.15 1.00	
0.02	0.28	0.44	R -1.0 ^{0.53}	R 1.0 (GOAL)
0.16	0.16		-0.30	
0.13 0.24	1 0.08 0.3	5 0.09 0.48	8 0.27 0.62	2 0.36 0.69
0.08	0.17	0.33	0.18	0.18
0.22			0.33	0.65
0.20 0.08	3		0.19 0.17	7 0.22 0.29
0.10			-0.22	-1.04
0.14	-0.06	-0.13	0.19	0.34
0.02 0.0	7 0.12 0.03	3 0.03 -0.1	8 -0.13 -1.2	3 -0.10 -0.73
0.09	-0.00	-0.02	-0.35	R -1.0 ^{0.24}
0.11	0.07	-0.03	-0.57	-0.95
0.05 0.02	2 0.04 -0.1	8 -0.09 -0.3	2 -0.14 -0.52	2 -0.30 -0.48
0.07	0.03	-0.03	0.01	-0.35

→	ţ	ţ	→ R -1.0	R 1.0 (GOAL)
→	→	→	→	†
î			†	1

1	←	←	1	1
				R -1.0
1	†	1	1	←

Epsilon=0.5, Alpha=0.3

	0.28	0.08	0.17	-0.32	
	0.20 0.08	8 0.03 0.00	0 0.14 -0.2	6 0.04 1.00	b
	0.21	0.10	0.24	R-1.0 ^{0.41}	R 1.0 (GOAL)
	0.08	-0.01		-0.55	
	0.31 0.34	4 0.35 0.0	7 0.07 0.14	4 0.19 0.42	2 0.03 0.69
	0.11	0.18	-0.18	-0.06	0.71
	0.06			0.30	0.85
	-0.02 0.23	2		-0.00 0.6	7 0.13 0.60
	0.07			0.02	-0.78
	0.06	0.02	-0.02	0.58	0.61
	0.06 0.20	0.14 0.23	2 0.02 0.23	2 0.02 -0.7	7 -0.39 -0.99
	0.03	0.01	0.12	-0.12	R-1.0 ^{0.37}
	0.14	0.05	-0.08	-0.09	-0.77
	-0.01 0.02	2 0.02 -0.1	7 0.10 -0.0	9 0.07 -0.10	6 0.02 -0.08
	-0.00	0.13	-0.13	0.05	-0.55
7					

Î	1	ţ	→ R -1.0	R 1.0 (GOAL)
→	←	†	→	1
→			→	1
→	→	→	†	↑ R -1.0
Ť.	1	←	←	←

Epsilon=0.5, Alpha=0.5

	-0.30	0.04	0.01	0.00
--	-------	------	------	------

0.00 0.0	5 -0.32 0.1	8 -0.26 -0.7	4 0.35 1.00	b
0.10	0.04	0.05	R-1.0 ^{0.69}	R 1.0 (GOAL)
0.02	0.13		-0.43	1.00
0.02 -0.0	4 0.05 0.19	9 0.42 0.70	6 0.66 0.8 9	0.78 0.90
0.07	0.41	-0.23	0.45	0.66
0.04			0.33	0.87
0.04 0.03	2		-0.55 0.07	7 0.03 0.16
0.05			-0.07	-0.56
-0.00	-0.05	-0.20	0.04	0.76
-0.02 -0.0	4 -0.04 -0.0	8 0.04 -0.2	3 0.06 -0.5	5 -0.13 -0.89
-0.02	-0.02	-0.07	-0.03	R-1.0 ^{0.08}
-0.02	-0.07	-0.18	-0.11	-0.51
-0.03 -0.0	6 -0.02 -0.0	3 -0.03 -0.1	3 -0.10 -0.10	0 -0.07 -0.32
-0.03	-0.00	-0.01	-0.15	-0.01

1	→	1	→ R -1.0	R 1.0 (GOAL)
1	1	→	→	1
1			1	1
†	1	←	←	↑ R -1.0
				11.0