State-Value Function Grid World

Dynamic Programming

In [ ]:

```
V = {'L1': 0.0, 'L2': 0.0}
new_V = V.copy()

cnt = 0  # 갱신 횟수 기록
while True:
    new_V['L1'] = 0.5 * (-1 + 0.9 * V['L1']) + 0.5 * (1 + 0.9 * V['L2
    new_V['L2'] = 0.5 * (0 + 0.9 * V['L1']) + 0.5 * (-1 + 0.9 * V['L2

     # 갱신된 양의 최댓값
    delta = abs(new_V['L1'] - V['L1'])
    delta = max(delta, abs(new_V['L2'] - V['L2']))
    V = new_V.copy()

    cnt += 1
    if delta < 0.0001:  # 임계값 = 0.0001
        print(V)
        print('갱신 횟수:', cnt)
        break
```

```
{'L1': -2.249167525908671, 'L2': -2.749167525908671}
갱신 횟수: 76
```

Dynamic Programming inplace

In [ ]:

```
V = {'L1': 0.0, 'L2': 0.0}

cnt = 0
while True:
    t = 0.5 * (-1 + 0.9 * V['L1']) + 0.5 * (1 + 0.9 * V['L2'])
    delta = abs(t - V['L1'])
    V['L1'] = t

    t = 0.5 * (0 + 0.9 * V['L1']) + 0.5 * (-1 + 0.9 * V['L2'])
    delta = max(delta, abs(t - V['L2']))
    V['L2'] = t

    cnt += 1
    if delta < 0.0001:
        print(V)
        print('갱신 횟수:', cnt)
        break
```

```
{'L1': -2.2493782177156936, 'L2': -2.7494201578106514}
갱신 횟수: 60
```

In [ ]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

GridWorld class

In [ ]:

```python
import sys
sys.path.append('/content/drive/MyDrive/강화학습') #google colab 경로 지

import numpy as np
import common.gridworld_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],
             [0, None, 0, -1.0],
             [0, 0, 0, 0]]
        )
        self.goal_state = (0, 3)    # 목표 상태(좌표)
        self.wall_state = (1, 1)    # 벽 상태(좌표)
        self.start_state = (2, 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:
            next_state = state

        return next_state  # 다음 상태 반환

    def reward(self, state, action, next_state):
```

```
    def reward(self, state, action, next_state):
        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.goal_
                                          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.goal_
                                          self.wall_state)
        renderer.render_q(q, print_value)
```

test code

In [ ]:

```
env = GridWorld()

print(env.height)
print(env.width)
print(env.shape)
```

```
3
4
(3, 4)
```

In [ ]:

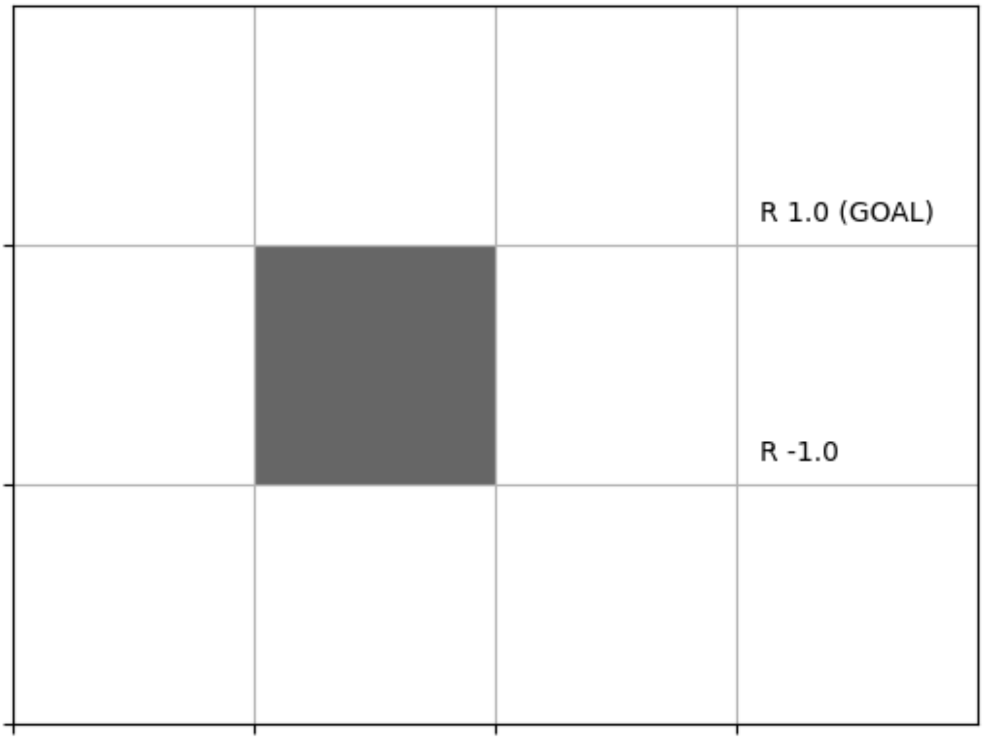
```
for action in env.actions():
    print(action)

print('====================')

for state in env.states():
    print(state)
```

```
0
1
2
3
====================
(0, 0)
(0, 1)
(0, 2)
(0, 3)
(1, 0)
(1, 1)
(1, 2)
(1, 3)
(2, 0)
(2, 1)
```

```
(2, 2)
(2, 3)
```

In [ ]:

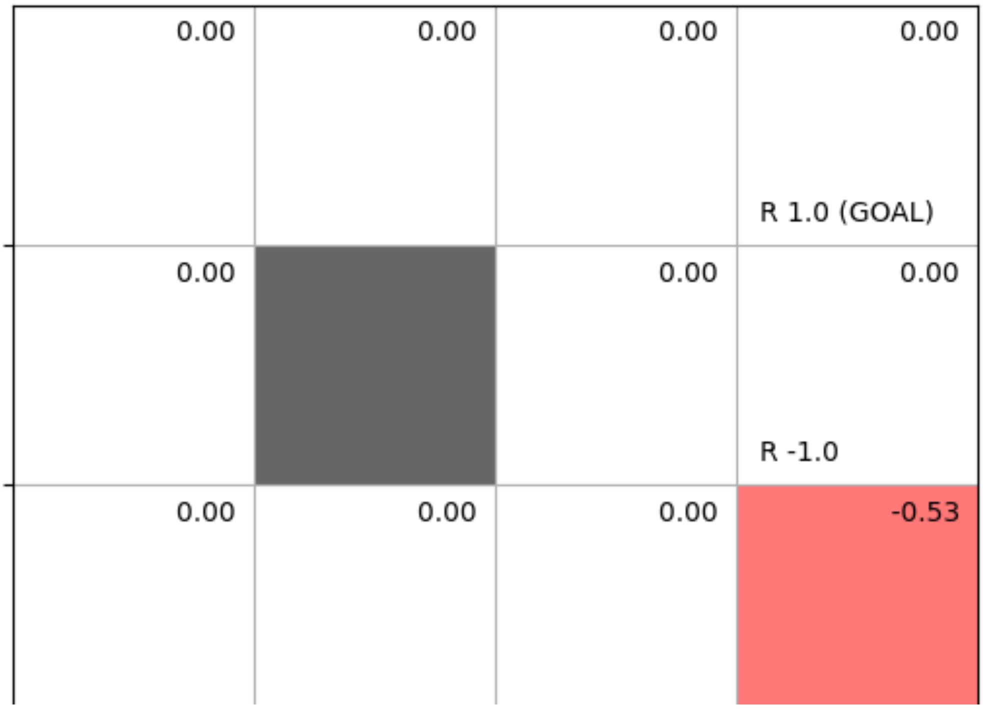
```
env = GridWorld()
env.render_v()
```



GridWorld Play

In [ ]:

```
env = GridWorld()
V = {}
for stete in env.states():
    V[state] = np.random.randn()
env.render_v(V)
```



In [ ]:

```python
env = GridWorld()
V = {}

for stete in env.states():
    V[state] = 0

state = (1,2)
print(V[stete])
```

```
0
```

In [ ]:

```python
#from collections import defaultdict

env = GridWorld()
V = defaultdict(lambda: 0)

state= (1,2)
print(V[state])
```

```
0
```

In [ ]:

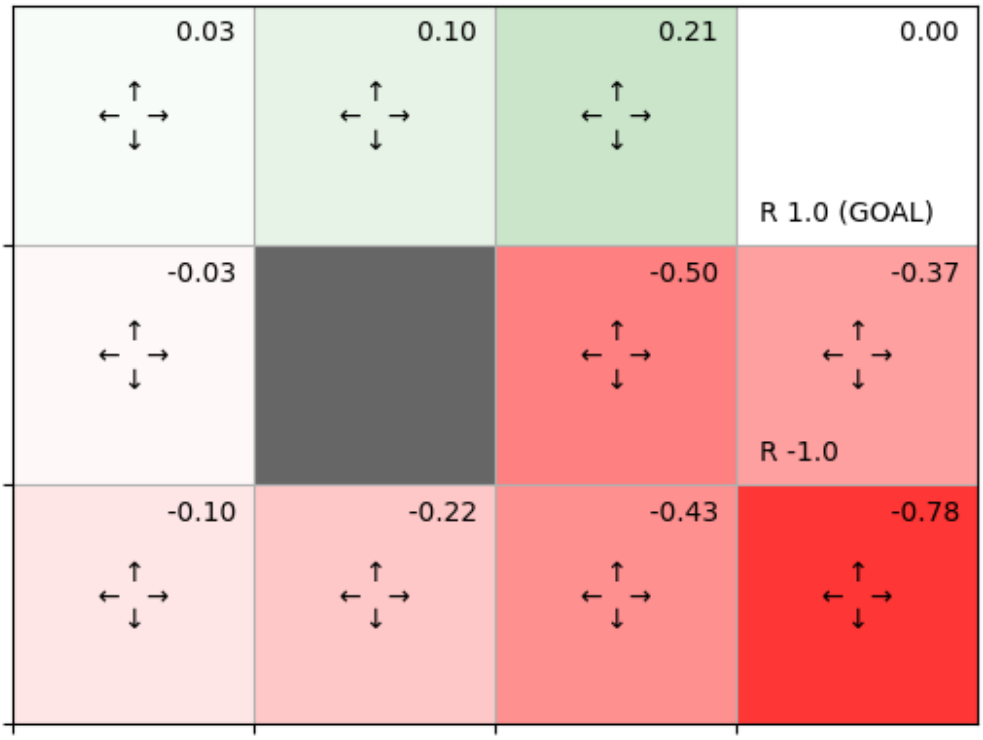
```python
pi = defaultdict(lambda :{0:0.25, 1:0.25, 2:0.25, 3:0.25})

state = (0,1)
print(pi[state])
```

```
{0: 0.25, 1: 0.25, 2: 0.25, 3: 0.25}
```

Policy Evaluation

In [ ]:

```python
from collections import defaultdict

def eval_onestep(pi, V, env, gamma=0.9):
    for state in env.states():  # 각 상태에 접근
        if state == env.goal_state:  # 목표 상태에서의 가치 함수는 항상 0
            V[state] = 0
            continue

        action_probs = pi[state]
        new_V = 0

        # 각 행동에 접근
        for action, action_prob in action_probs.items():
            next_state = env.next_state(state, action)
            r = env.reward(state, action, next_state)
            # 새로운 가치 함수
            new_V += action_prob * (r + gamma * V[next_state])

        V[state] = new_V
    return V


def policy_eval(pi, V, env, gamma, threshold=0.001):
    while True:
        old_V = V.copy()  # 갱신 전 가치 함수
        V = eval_onestep(pi, V, env, gamma)

        # 갱신된 양의 최댓값 계산
        delta = 0
        delta = 0
        for state in V.keys():
            t = abs(V[state] - old_V[state])
            if delta < t:
                delta = t

        # 임계값과 비교
        if delta < threshold:
            break
    return V


if __name__ == '__main__':
    env = GridWorld()
    gamma = 0.9  # 할인율

    pi = defaultdict(lambda: {0: 0.25, 1: 0.25, 2: 0.25, 3: 0.25})  #
    V = defaultdict(lambda: 0)  # 가치 함수

    V = policy_eval(pi, V, env, gamma)  # 정책 평가

    env.render_v(V, pi)
```

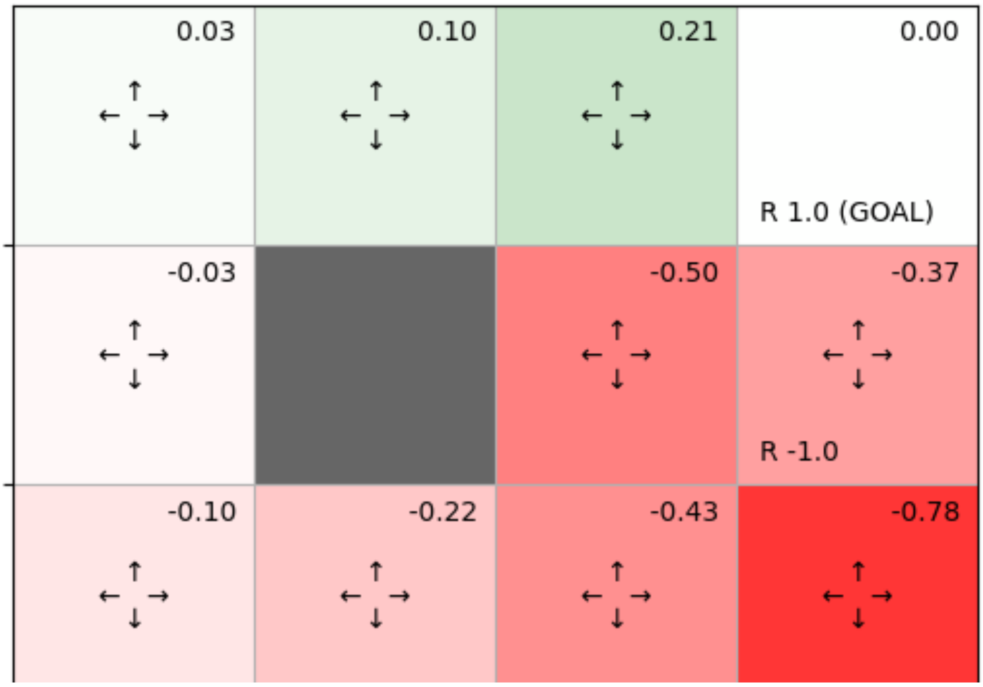


Policy iteration- 정책반복법

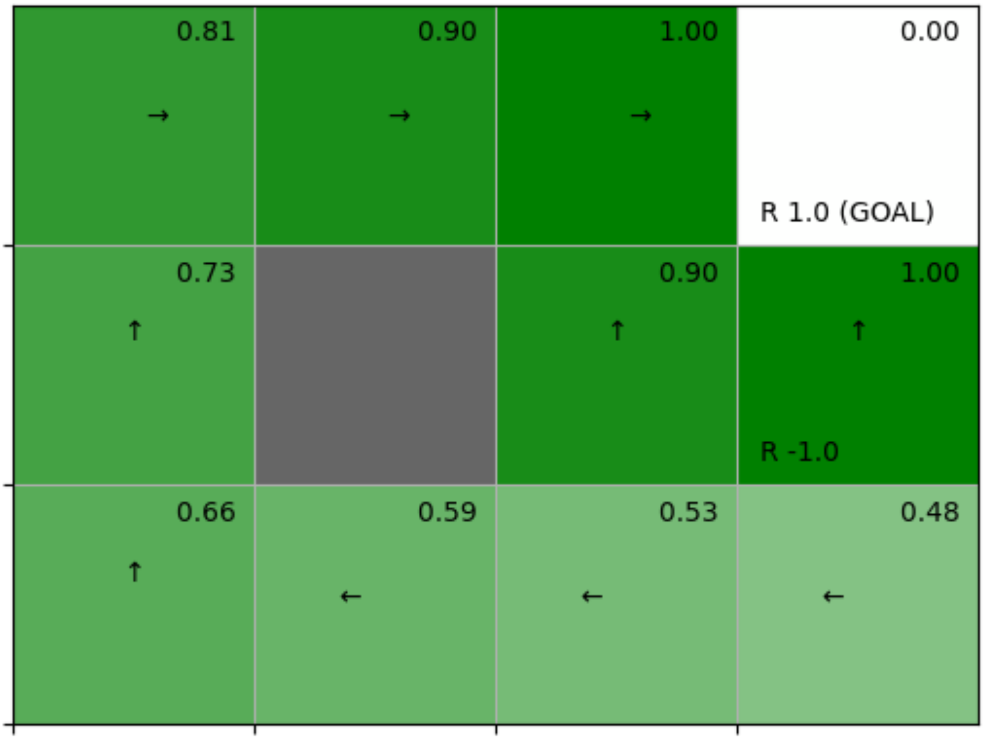
In [ ]:

```
from collections import defaultdict


def argmax(d):
    max_value = max(d.values())
    max_key = 0
    for key, value in d.items():
        if value == max_value:
            max_key = key
    return max_key
```

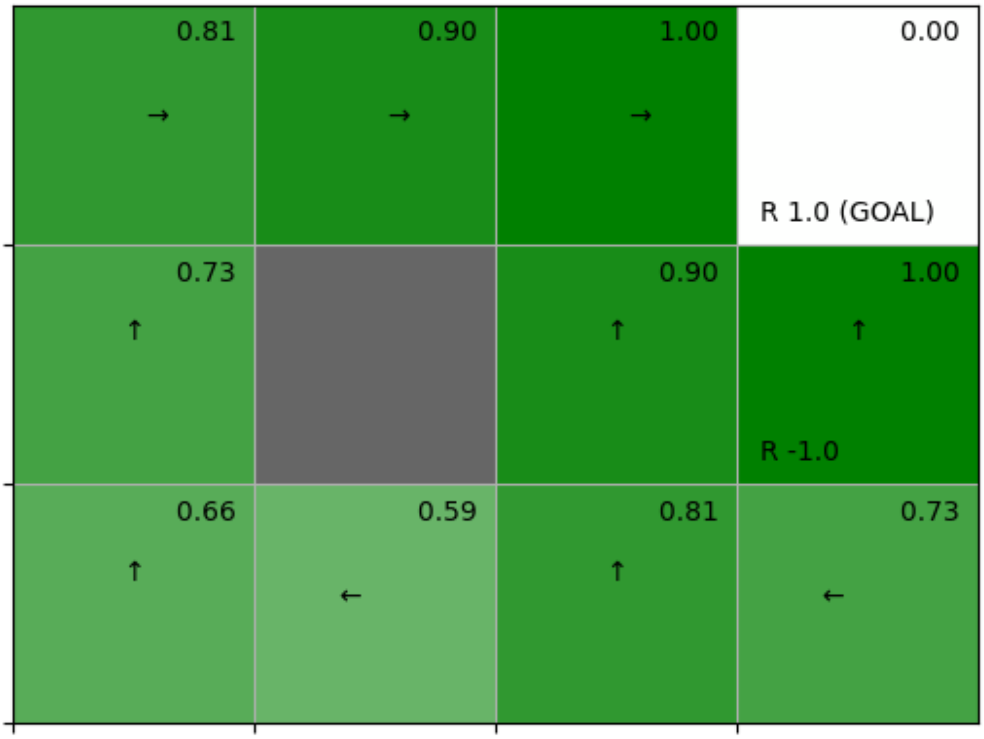
```
def greedy_policy(V, env, gamma):
    pi = {}

    for state in env.states():
        action_values = {}

        for action in env.actions():
            next_state = env.next_state(state, action)
            r = env.reward(state, action, next_state)
            value = r + gamma * V[next_state]
            action_values[action] = value

        max_action = argmax(action_values)
        action_probs = {0: 0, 1: 0, 2: 0, 3: 0}
        action_probs[max_action] = 1.0
        pi[state] = action_probs
    return pi


def policy_iter(env, gamma, threshold=0.001, is_render=True):
    pi = defaultdict(lambda: {0: 0.25, 1: 0.25, 2: 0.25, 3: 0.25})
    V = defaultdict(lambda: 0)

    while True:
        V = policy_eval(pi, V, env, gamma, threshold)  # 평가
        new_pi = greedy_policy(V, env, gamma)          # 개선

        if is_render:
            env.render_v(V, pi)

        if new_pi == pi:  # 갱신 여부 확인
            break
        pi = new_pi

    return pi


if __name__ == '__main__':
    env = GridWorld()
    gamma = 0.9
    pi = policy_iter(env, gamma)
```



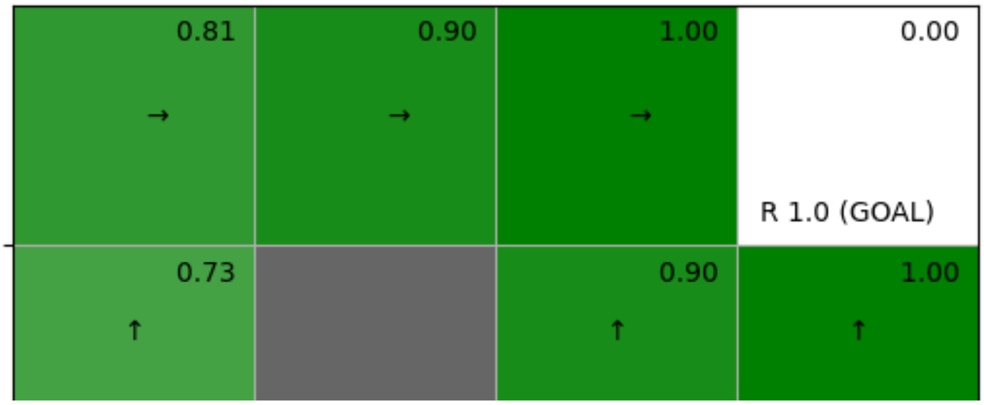
0.03 ↑ ← → ↓	0.10 ↑ ← → ↓	0.21 ↑ ← → ↓	0.00 R 1.0 (GOAL)
-0.03 ↑ ← → ↓		-0.50 ↑ ← → ↓	-0.37 ↑ ← → ↓ R -1.0
-0.10 ↑ ← → ↓	-0.22 ↑ ← → ↓	-0.43 ↑ ← → ↓	-0.78 ↑ ← → ↓



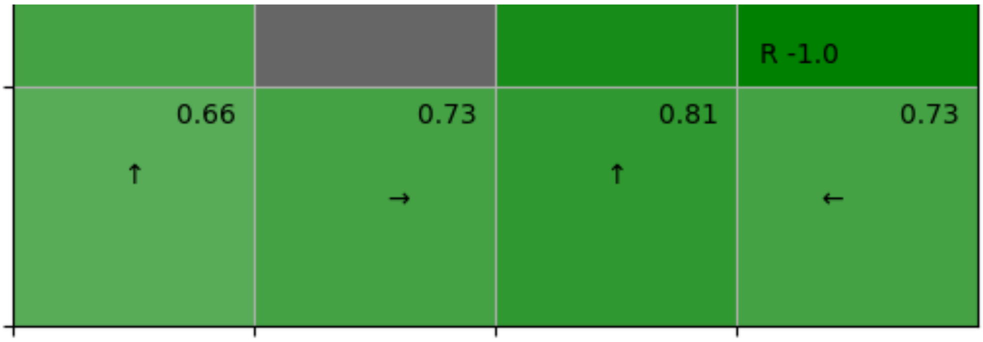
0.81
→
0.90
→
1.00
→
0.00
R 1.0 (GOAL)
0.73
↑
0.90
↑
1.00
↑
R -1.0
0.66
↑
0.59
←
0.53
←
0.48
←



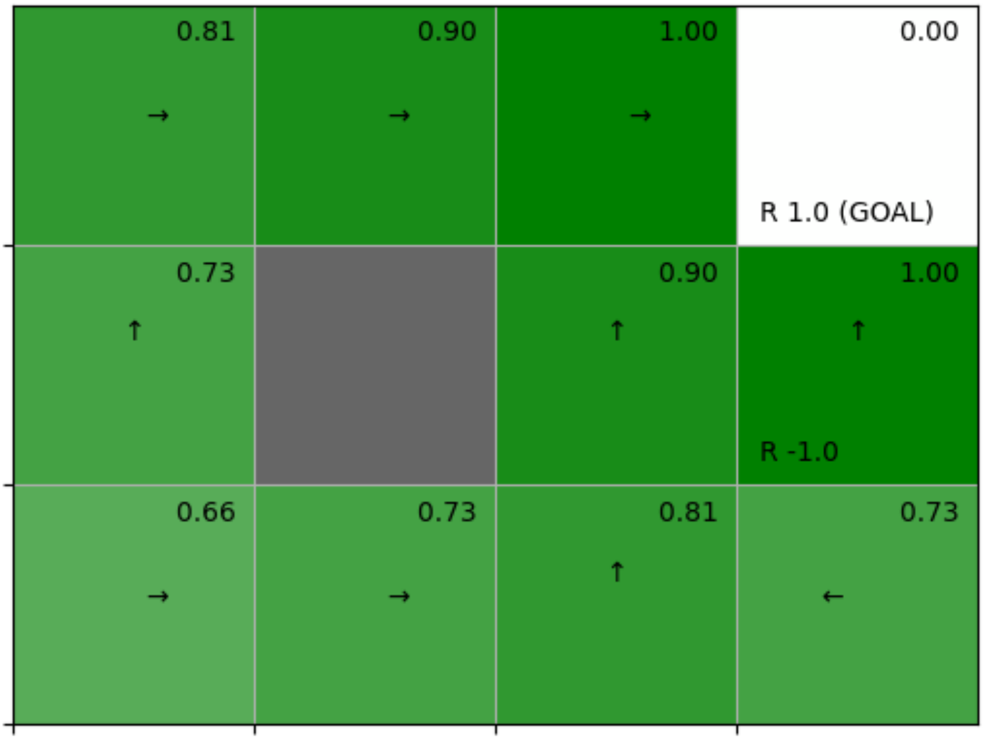
0.81
→
0.90
→
1.00
→
0.00
R 1.0 (GOAL)
0.73
↑
0.90
↑
1.00
↑
R -1.0
0.66
↑
0.59
←
0.81
↑
0.73
←



0.81
→
0.90
→
1.00
→
0.00
R 1.0 (GOAL)
0.73
↑
0.90
↑
1.00
↑



			R -1.0
0.66 ↑	0.73 →	0.81 ↑	0.73 ←



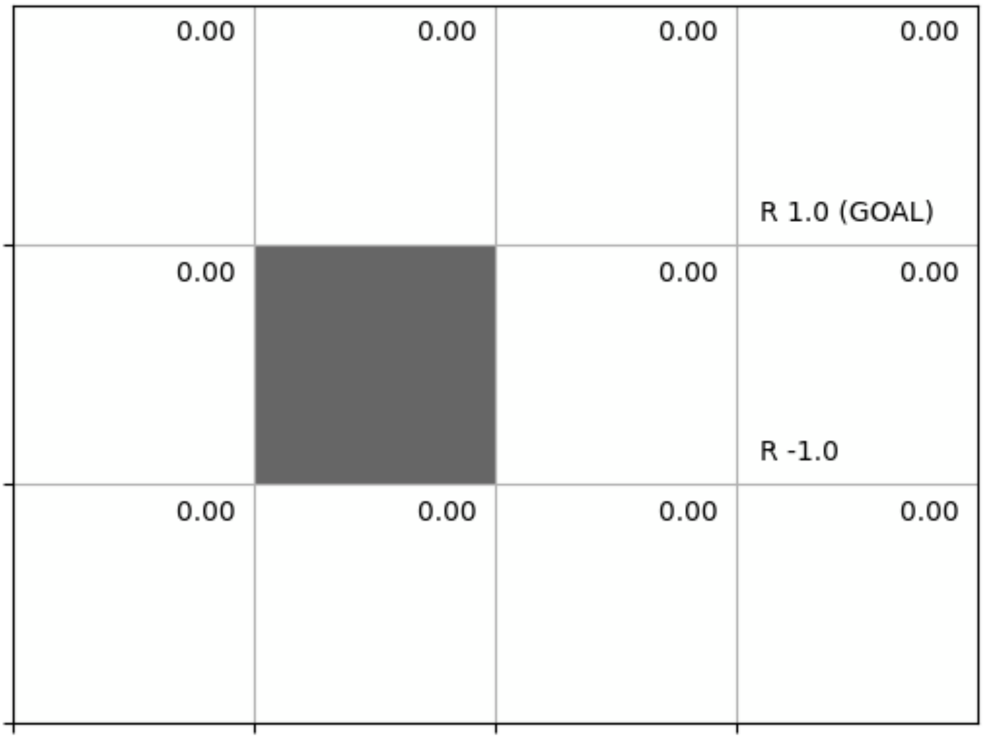
0.81 →	0.90 →	1.00 →	0.00 R 1.0 (GOAL)
0.73 ↑		0.90 ↑	1.00 ↑ R -1.0
0.66 →	0.73 →	0.81 ↑	0.73 ←

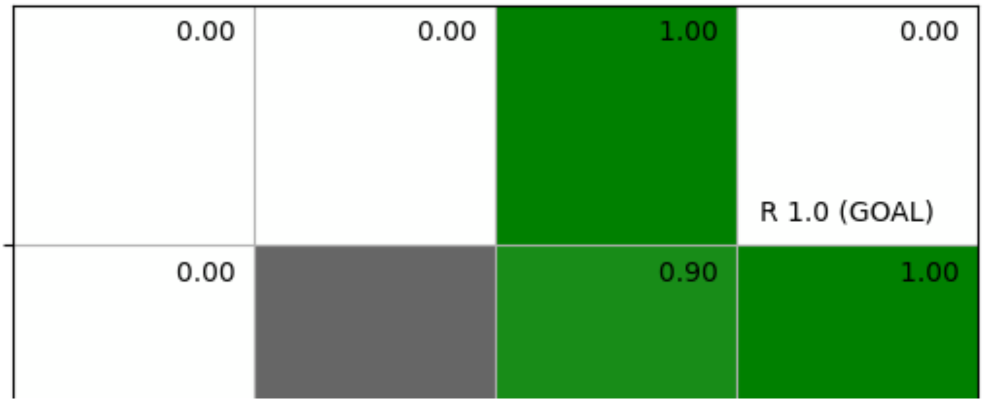
Value iteration - 가치반복법

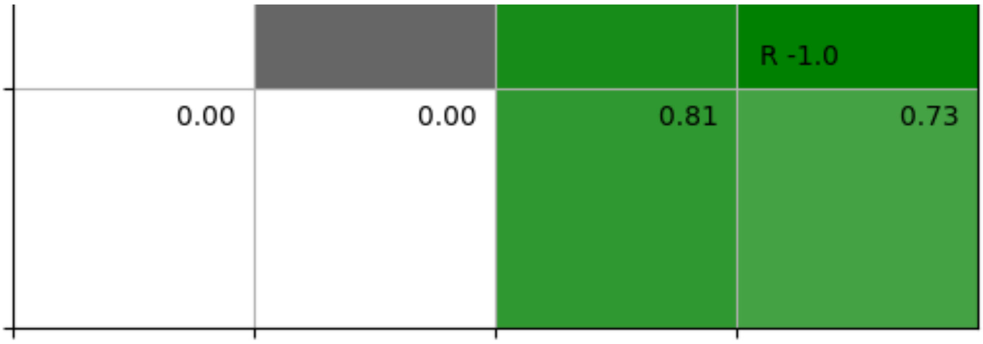
In [ ]:

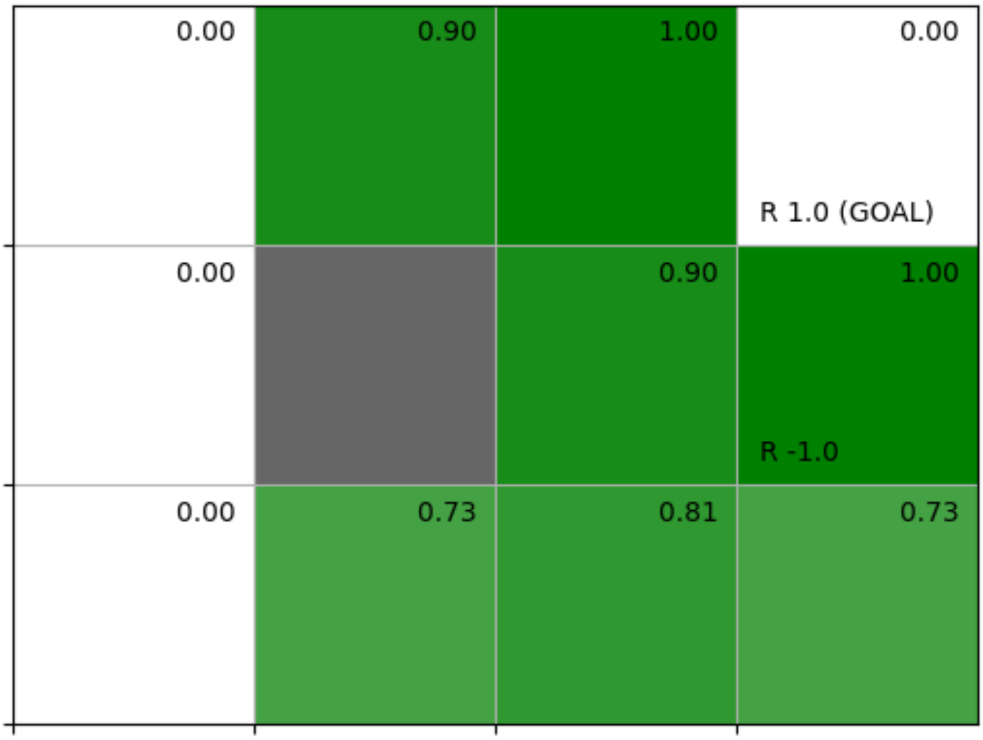
```
from collections import defaultdict


def value_iter_onestep(V, env, gamma):
```

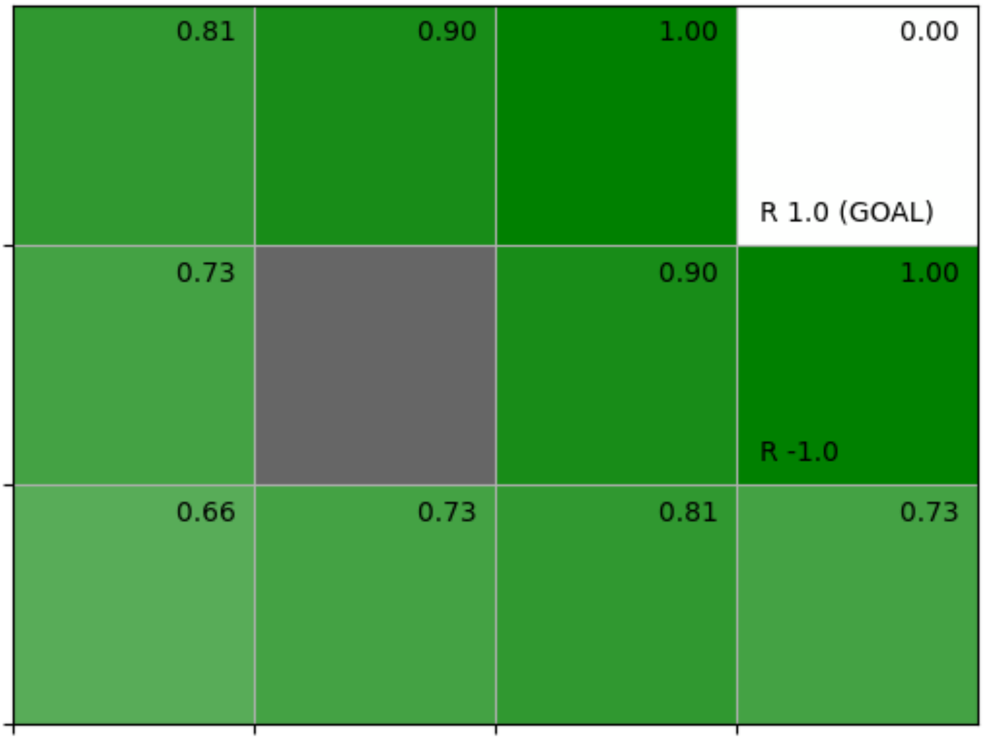
```
        action_values = []
        for action in env.actions():  # 모든 행동에 차례로 접근
            next_state = env.next_state(state, action)
            r = env.reward(state, action, next_state)
            value = r + gamma * V[next_state]  # 새로운 가치 함수
            action_values.append(value)

        V[state] = max(action_values)  # 최댓값 추출
    return V


def value_iter(V, env, gamma, threshold=0.001, is_render=True):
    while True:
        if is_render:
            env.render_v(V)
        old_V = V.copy()  # 갱신 전 가치 함수
        V = value_iter_onestep(V, env, gamma)

        # 갱신된 양의 최댓값 구하기
        delta = 0
        for state in V.keys():
            t = abs(V[state] - old_V[state])
            if delta < t:
                delta = t

        # 임계값과 비교
        if delta < threshold:
            break
    return V


if __name__ == '__main__':
    V = defaultdict(lambda: 0)
    env = GridWorld()
    gamma = 0.9

    V = value_iter(V, env, gamma)  # 최적 가치 함수 찾기

    pi = greedy_policy(V, env, gamma)  # 최적 정책 찾기
    env.render_v(V, pi)
```



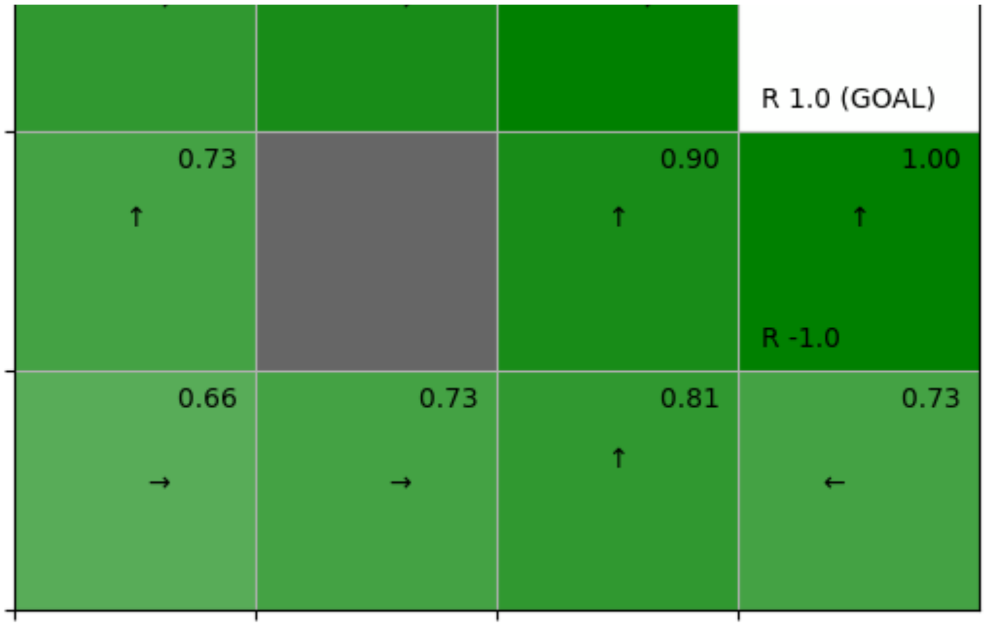



R -1.0
0.00
0.00
0.81
0.73


0.00
0.90
1.00
0.00
R 1.0 (GOAL)
0.00
0.90
1.00
R -1.0
0.00
0.73
0.81
0.73


0.81
0.90
1.00
0.00
R 1.0 (GOAL)
0.73
0.90
1.00
R -1.0
0.66
0.73
0.81
0.73

0.81
0.90
1.00
0.00



R 1.0 (GOAL)
0.73
↑
0.90
↑
1.00
↑
R -1.0
0.66
→
0.73
→
0.81
↑
0.73
←