REPORT

“ Dynamic Programming
Policy Iteration, Value Iteration ”

과 목 명	지능 시스템
담당교수	조영완 교수님
학 과	컴퓨터공학과
학 번	2016305078
이 름	최영환
제 출 일	2021.05.21

- 다음의 7×7 Grid map에서 최종 도착 목표 지점이 $(3,4)$이고 ▲로 표시된 위치에 장애물이 있을 때, 출발점 ■에서 목표지점을 찾아 가는 행동을 Dynamic Programming의 Policy Iteration과 Value Iteration을 이용해서 구하시오. Iteration 회수(예를 들어, $k = 0, 5, 10, 20, 50$ 등)에 따른 각 상태에서의 value table 값과 action list를 결과로 제시하시오. (제출: 5월 21일 6시까지, Google Classroom)

< 해결 방법 >

▶ 교재의 예제 코드를 활용하였음.

▶ Policy Iteration 과 Value Iteration 파일을 교재의 Github 에서 다운로드 받아 사용하였음.

< 교재 코드와의 차이점 >

▶ 교재에서는 5 X 5 Grid map 을 사용하였으나, 이번 과제는 7 X 7 Grid map 을 사용하였음.

▶ hazard(장애물) 과 목표지점이 다름.

▶ 교재의 예제 코드 수정을 통하여 위 차이점들을 적용하여 해결하였음.

1. Policy_Iteration/environment.py 파일 수정사항

```
PhotoImage = ImageTk.PhotoImage
UNIT = 100  # 픽셀 수
HEIGHT = 5  # 그리드월드 세로
WIDTH = 5  # 그리드월드 가로
TRANSITION_PROB = 1
POSSIBLE_ACTIONS = [0, 1, 2, 3]  # 좌, 우, 상, 하
ACTIONS = [(-1, 0), (1, 0), (0, -1), (0, 1)]  # 좌표로 나타낸 행동
REWARDS = []
```

```
PhotoImage = ImageTk.PhotoImage
UNIT = 100  # 픽셀 수
HEIGHT = 7  # 그리드월드 세로
WIDTH = 7  # 그리드월드 가로
TRANSITION_PROB = 1
POSSIBLE_ACTIONS = [0, 1, 2, 3]  # 좌, 우, 상, 하
ACTIONS = [(-1, 0), (1, 0), (0, -1), (0, 1)]  # 좌표로 나타낸 행동
REWARDS = []
```

▶ Grid World 의 가로 세로를 5 X 5 에서 7 X 7로 변경하였음.

(1) GraphicDisplay 클래스 수정사항

```
class GraphicDisplay(tk.Tk):
    def __init__(self, agent):
        super(GraphicDisplay, self).__init__()
        self.title('Policy Iteration')
        self.geometry('{0}x{1}'.format(HEIGHT * UNIT, HEIGHT * UNIT + 50))
        self.texts = []
        self.arrows = []
        self.env = Env()
        self.agent = agent
        self.evaluation_count = 0
        self.improvement_count = 0
        self.is_moving = 0
        (self.up, self.down, self.left, self.right), self.shapes = self.load_images()
        self.canvas = self._build_canvas()
        self.text_reward(2, 2, "R : 1.0")
        self.text_reward(1, 2, "R : -1.0")
        self.text_reward(2, 1, "R : -1.0")
```

```
class GraphicDisplay(tk.Tk):
    def __init__(self, agent):
        self.k = 0
        print(f'k = {self.k}')
        super(GraphicDisplay, self).__init__()
        self.title('Policy Iteration')
        self.geometry('{0}x{1}'.format(HEIGHT * UNIT, HEIGHT * UNIT + 50))
        self.texts = []
        self.arrows = []
        self.env = Env()
        self.agent = agent
        self.evaluation_count = 0
        self.improvement_count = 0
        self.is_moving = 0
        (self.up, self.down, self.left, self.right), self.shapes = self.load_images()
        self.canvas = self._build_canvas()
        self.text_reward(3, 4, "R : 1.0")       # 마침상태 보상
        self.text_reward(0, 4, "R : -1.0")      # 장애물 보상
        self.text_reward(1, 2, "R : -1.0")      # 장애물 보상
        self.text_reward(2, 4, "R : -1.0")      # 장애물 보상
        self.text_reward(3, 3, "R : -1.0")      # 장애물 보상
        self.text_reward(3, 5, "R : -1.0")      # 장애물 보상
        self.text_reward(4, 2, "R : -1.0")      # 장애물 보상
        self.text_reward(5, 3, "R : -1.0")      # 장애물 보상
```

▶ Policy Evaluation 횟수 관측을 위해 변수 k를 추가하였음.

▶ 실행 시, k = 0 을 콘솔창에 출력함.

▶ Grid map 에서의 보상의 크기를 텍스트로 출력해주는 코드를 문제 요구사항에 맞도록 수정하였음.

```
# 캔버스에 이미지 추가
self.rectangle = canvas.create_image(50, 50, image=self.shapes[0])
canvas.create_image(250, 150, image=self.shapes[1])
canvas.create_image(150, 250, image=self.shapes[1])
canvas.create_image(250, 250, image=self.shapes[2])
```

```
# 캔버스에 이미지 추가
self.rectangle = canvas.create_image(50, 50, image=self.shapes[0])
canvas.create_image(450, 350, image=self.shapes[2])     # 마침상태(동그라미)
canvas.create_image(450, 50, image=self.shapes[1])      # 장애물(세모)
canvas.create_image(250, 150, image=self.shapes[1])     # 장애물(세모)
canvas.create_image(450, 250, image=self.shapes[1])     # 장애물(세모)
canvas.create_image(350, 350, image=self.shapes[1])     # 장애물(세모)
canvas.create_image(550, 350, image=self.shapes[1])     # 장애물(세모)
canvas.create_image(250, 450, image=self.shapes[1])     # 장애물(세모)
canvas.create_image(350, 550, image=self.shapes[1])     # 장애물(세모)
```

▶ Grid map 에서의 hazard 와 목표 지점의 이미지와 좌표 값을 문제 요구사항에 맞도록 수정하였음.

```
def reset(self):
    if self.is_moving == 0:
        self.evaluation_count = 0
        self.improvement_count = 0
        for i in self.texts:
            self.canvas.delete(i)

        for i in self.arrows:
            self.canvas.delete(i)
        self.agent.value_table = [[0.0] * WIDTH for _ in range(HEIGHT)]
        self.agent.policy_table = ([[[0.25, 0.25, 0.25, 0.25]] * WIDTH
                                    for _ in range(HEIGHT)])
        self.agent.policy_table[2][2] = []
        x, y = self.canvas.coords(self.rectangle)
        self.canvas.move(self.rectangle, UNIT / 2 - x, UNIT / 2 - y)
```

```
def reset(self):
    self.k = 0
    print('==========================================================================')
    print(f'k = {self.k}')
    if self.is_moving == 0:
        self.evaluation_count = 0
        self.improvement_count = 0
        for i in self.texts:
            self.canvas.delete(i)

        for i in self.arrows:
            self.canvas.delete(i)
        self.agent.value_table = [[0.0] * WIDTH for _ in range(HEIGHT)]
        self.agent.policy_table = ([[[0.25, 0.25, 0.25, 0.25]] * WIDTH
                                    for _ in range(HEIGHT)])
        self.agent.policy_table[3][4] = []
        x, y = self.canvas.coords(self.rectangle)
        self.canvas.move(self.rectangle, UNIT / 2 - x, UNIT / 2 - y)
```

▶ reset 이 제대로 수행되었는지 확인을 위해 변수 k의 값을 초기화하고, 출력문을 추가하였음.

▶ reset 시 agent의 목표 지점을 문제 요구사항에 맞도록 수정하였음.

```
def draw_one_arrow(self, col, row, policy):
    if col == 2 and row == 2:
        return
```

```
def draw_one_arrow(self, col, row, policy):
    if col == 3 and row == 4:
        return
```

▶ Grid map 에서 action을 표시하는 화살표가 표시되지 않을 지점(목표 지점)을 문제 요구사항에 맞도록 수정하였음.

```
def evaluate_policy(self):
    self.evaluation_count += 1
    for i in self.texts:
        self.canvas.delete(i)
    self.agent.policy_evaluation()
    self.print_value_table(self.agent.value_table)
```

```
def evaluate_policy(self):
    self.k += 1
    print(f'k = {self.k}')
    self.evaluation_count += 1
    for i in self.texts:
        self.canvas.delete(i)
    self.agent.policy_evaluation()
    self.print_value_table(self.agent.value_table)
```

▶ policy evaluation 횟수 측정을 위해 변수 k의 값을 1 씩 증가시키고, 이를 콘솔에 출력하도록 출력문 코드를 추가하였음.

(2) Env 클래스 수정사항

```
class Env:
    def __init__(self):
        self.transition_probability = TRANSITION_PROB
        self.width = WIDTH
        self.height = HEIGHT
        self.reward = [[0] * WIDTH for _ in range(HEIGHT)]
        self.possible_actions = POSSIBLE_ACTIONS
        self.reward[2][2] = 1  # (2,2) 좌표 동그라미 위치에 보상 1
        self.reward[1][2] = -1  # (1,2) 좌표 세모 위치에 보상 -1
        self.reward[2][1] = -1  # (2,1) 좌표 세모 위치에 보상 -1
        self.all_state = []
```

```
class Env:
    def __init__(self):
        self.transition_probability = TRANSITION_PROB
        self.width = WIDTH
        self.height = HEIGHT
        self.reward = [[0] * WIDTH for _ in range(HEIGHT)]
        self.possible_actions = POSSIBLE_ACTIONS
        self.reward[3][4] = 1   # (3,4) 좌표 동그라미 위치에 보상 1
        self.reward[0][4] = -1  # (0,4) 좌표 세모 위치에 보상 -1
        self.reward[1][2] = -1  # (1,2) 좌표 세모 위치에 보상 -1
        self.reward[2][4] = -1  # (2,4) 좌표 세모 위치에 보상 -1
        self.reward[3][3] = -1  # (3,3) 좌표 세모 위치에 보상 -1
        self.reward[3][5] = -1  # (3,5) 좌표 세모 위치에 보상 -1
        self.reward[4][2] = -1  # (4,2) 좌표 세모 위치에 보상 -1
        self.reward[5][3] = -1  # (5,3) 좌표 세모 위치에 보상 -1
        self.all_state = []
```

▶ Grid map 에서의 reward(보상) 값이 저장된 위치를 문제 요구사항에 맞도록 수정하였음.

2. Policy_Iteration/policy_iteration.py 파일 수정사항

```python
class PolicyIteration:
    def __init__(self, env):
        # 환경에 대한 객체 선언
        self.env = env
        # 가치함수를 2차원 리스트로 초기화
        self.value_table = [[0.0] * env.width for _ in range(env.height)]
        # 상 하 좌 우 동일한 확률로 정책 초기화
        self.policy_table = [[[0.25, 0.25, 0.25, 0.25]] * env.width
                             for _ in range(env.height)]
        # 마침 상태의 설정
        self.policy_table[2][2] = []
        # 할인율
        self.discount_factor = 0.9
```

```python
class PolicyIteration:
    def __init__(self, env):
        # 환경에 대한 객체 선언
        self.env = env
        # 가치함수를 2차원 리스트로 초기화
        self.value_table = [[0.0] * env.width for _ in range(env.height)]
        # 상 하 좌 우 동일한 확률로 정책 초기화
        self.policy_table = [[[0.25, 0.25, 0.25, 0.25]] * env.width
                             for _ in range(env.height)]
        # 마침 상태의 설정
        self.policy_table[3][4] = []
        # 할인율
        self.discount_factor = 0.9
```

```python
# 벨만 기대 방정식을 통해 다음 가치함수를 계산하는 정책 평가
def policy_evaluation(self):
    # 다음 가치함수 초기화
    next_value_table = [[0.00] * self.env.width
                        for _ in range(self.env.height)]

    # 모든 상태에 대해서 벨만 기대방정식을 계산
    for state in self.env.get_all_states():
        value = 0.0
        # 마침 상태의 가치 함수 = 0
        if state == [2, 2]:
            next_value_table[state[0]][state[1]] = value
            continue

        # 벨만 기대 방정식
        for action in self.env.possible_actions:
            next_state = self.env.state_after_action(state, action)
            reward = self.env.get_reward(state, action)
            next_value = self.get_value(next_state)
            value += (self.get_policy(state)[action] *
                      (reward + self.discount_factor * next_value))

        next_value_table[state[0]][state[1]] = value

    self.value_table = next_value_table
```

```python
# 벨만 기대 방정식을 통해 다음 가치함수를 계산하는 정책 평가
def policy_evaluation(self):
    # 다음 가치함수 초기화
    next_value_table = [[0.00] * self.env.width
                        for _ in range(self.env.height)]

    # 모든 상태에 대해서 벨만 기대방정식을 계산
    for state in self.env.get_all_states():
        value = 0.0
        # 마침 상태의 가치 함수 = 0
        if state == [3, 4]:
            next_value_table[state[0]][state[1]] = value
            continue

        # 벨만 기대 방정식
        for action in self.env.possible_actions:
            next_state = self.env.state_after_action(state, action)
            reward = self.env.get_reward(state, action)
            next_value = self.get_value(next_state)
            value += (self.get_policy(state)[action] *
                      (reward + self.discount_factor * next_value))

        next_value_table[state[0]][state[1]] = value

    self.value_table = next_value_table
```

```python
# 현재 가치 함수에 대해서 탐욕 정책 발전
def policy_improvement(self):
    next_policy = self.policy_table
    for state in self.env.get_all_states():
        if state == [2, 2]:
            continue
```

```python
# 현재 가치 함수에 대해서 탐욕 정책 발전
def policy_improvement(self):
    next_policy = self.policy_table
    for state in self.env.get_all_states():
        if state == [3, 4]:
            continue
```

▶ 마침 상태를 문제 요구사항에 맞도록 수정하였음.

※ Policy Iteration 과 Value Iteration 의 수행 결과가 동일하므로, 마지막에 두 결과를 한번에 관측하도록 편집하였음.

3. Value_Iteration/environment.py 파일 수정사항

```
PhotoImage = ImageTk.PhotoImage
UNIT = 100  # 픽셀 수
HEIGHT = 5  # 그리드월드 세로
WIDTH = 5  # 그리드월드 가로
TRANSITION_PROB = 1
POSSIBLE_ACTIONS = [0, 1, 2, 3]  # 좌, 우, 상, 하
ACTIONS = [(-1, 0), (1, 0), (0, -1), (0, 1)]  # 좌표로 나타낸 행동
REWARDS = []
```

```
PhotoImage = ImageTk.PhotoImage
UNIT = 100  # 픽셀 수
HEIGHT = 7  # 그리드월드 세로
WIDTH = 7  # 그리드월드 가로
TRANSITION_PROB = 1
POSSIBLE_ACTIONS = [0, 1, 2, 3]  # 좌, 우, 상, 하
ACTIONS = [(-1, 0), (1, 0), (0, -1), (0, 1)]  # 좌표로 나타낸 행동
REWARDS = []
```

▶ Grid World 의 가로 세로를 5 X 5 에서 7 X 7로 변경하였음.

(1) GraphicDisplay 클래스 수정사항

```
class GraphicDisplay(tk.Tk):
    def __init__(self, agent):
        super(GraphicDisplay, self).__init__()
        self.title('Policy Iteration')
        self.geometry('{0}x{1}'.format(HEIGHT * UNIT, HEIGHT * UNIT + 50))
        self.texts = []
        self.arrows = []
        self.env = Env()
        self.agent = agent
        self.evaluation_count = 0
        self.improvement_count = 0
        self.is_moving = 0
        (self.up, self.down, self.left, self.right), self.shapes = self.load_images()
        self.canvas = self._build_canvas()
        self.text_reward(2, 2, "R : 1.0")
        self.text_reward(1, 2, "R : -1.0")
        self.text_reward(2, 1, "R : -1.0")
```

```
class GraphicDisplay(tk.Tk):
    def __init__(self, agent):
        self.k = 0
        print(f'k = {self.k}')
        super(GraphicDisplay, self).__init__()
        self.title('Policy Iteration')
        self.geometry('{0}x{1}'.format(HEIGHT * UNIT, HEIGHT * UNIT + 50))
        self.texts = []
        self.arrows = []
        self.env = Env()
        self.agent = agent
        self.evaluation_count = 0
        self.improvement_count = 0
        self.is_moving = 0
        (self.up, self.down, self.left, self.right), self.shapes = self.load_images()
        self.canvas = self._build_canvas()
        self.text_reward(3, 4, "R : 1.0")       # 마침상태 보상
        self.text_reward(0, 4, "R : -1.0")      # 장애물 보상
        self.text_reward(1, 2, "R : -1.0")      # 장애물 보상
        self.text_reward(2, 4, "R : -1.0")      # 장애물 보상
        self.text_reward(3, 3, "R : -1.0")      # 장애물 보상
        self.text_reward(3, 5, "R : -1.0")      # 장애물 보상
        self.text_reward(4, 2, "R : -1.0")      # 장애물 보상
        self.text_reward(5, 3, "R : -1.0")      # 장애물 보상
```

▶ Policy Evaluation 횟수 관측을 위해 변수 k를 추가하였음.

▶ 실행 시, k = 0 을 콘솔창에 출력함.

▶ Grid map 에서의 보상의 크기를 텍스트로 출력해주는 코드를 문제 요구사항에 맞도록 수정하였음.

```
# 캔버스에 이미지 추가
self.rectangle = canvas.create_image(50, 50, image=self.shapes[0])
canvas.create_image(250, 150, image=self.shapes[1])
canvas.create_image(150, 250, image=self.shapes[1])
canvas.create_image(250, 250, image=self.shapes[2])
```

```
# 캔버스에 이미지 추가
self.rectangle = canvas.create_image(50, 50, image=self.shapes[0])
canvas.create_image(450, 350, image=self.shapes[2])     # 마침상태(동그라미)
canvas.create_image(450, 50, image=self.shapes[1])      # 장애물(세모)
canvas.create_image(250, 150, image=self.shapes[1])     # 장애물(세모)
canvas.create_image(450, 250, image=self.shapes[1])     # 장애물(세모)
canvas.create_image(350, 350, image=self.shapes[1])     # 장애물(세모)
canvas.create_image(550, 350, image=self.shapes[1])     # 장애물(세모)
canvas.create_image(250, 450, image=self.shapes[1])     # 장애물(세모)
canvas.create_image(350, 550, image=self.shapes[1])     # 장애물(세모)
```

▶ Grid map 에서 action을 표시하는 화살표가 표시되지 않을 지점(목표 지점)을 문제 요구사항에 맞도록 수정하였음.

```
def draw_one_arrow(self, col, row, policy):
    if col == 2 and row == 2:
        return
```

```
def draw_one_arrow(self, col, row, policy):
    if col == 3 and row == 4:
        return
```

```python
def calculate_value(self):
    self.iteration_count += 1
    for i in self.texts:
        self.canvas.delete(i)
    self.agent.value_iteration()
    self.print_values(self.agent.value_table)
```

```python
def calculate_value(self):
    self.k += 1
    print(f'k = {self.k}')
    self.iteration_count += 1
    for i in self.texts:
        self.canvas.delete(i)
    self.agent.value_iteration()
    self.print_values(self.agent.value_table)
```

▶ Policy Iteration 때 와 동일하게 횟수 관측을 위하여 k의 값을 증가시키고, 이를 관측하는 출력문 코드를 삽입하였음.

(2) Env 클래스 수정사항

```python
class Env:
    def __init__(self):
        self.transition_probability = TRANSITION_PROB
        self.width = WIDTH
        self.height = HEIGHT
        self.reward = [[0] * WIDTH for _ in range(HEIGHT)]
        self.possible_actions = POSSIBLE_ACTIONS
        self.reward[2][2] = 1  # (2,2) 좌표 동그라미 위치에 보상 1
        self.reward[1][2] = -1  # (1,2) 좌표 세모 위치에 보상 -1
        self.reward[2][1] = -1  # (2,1) 좌표 세모 위치에 보상 -1
        self.all_state = []
```

```python
class Env:
    def __init__(self):
        self.transition_probability = TRANSITION_PROB
        self.width = WIDTH
        self.height = HEIGHT
        self.reward = [[0] * WIDTH for _ in range(HEIGHT)]
        self.possible_actions = POSSIBLE_ACTIONS
        self.reward[3][4] = 1  # (3,4) 좌표 동그라미 위치에 보상 1
        self.reward[0][4] = -1  # (0,4) 좌표 세모 위치에 보상 -1
        self.reward[1][2] = -1  # (1,2) 좌표 세모 위치에 보상 -1
        self.reward[2][4] = -1  # (2,4) 좌표 세모 위치에 보상 -1
        self.reward[3][3] = -1  # (3,3) 좌표 세모 위치에 보상 -1
        self.reward[3][5] = -1  # (3,5) 좌표 세모 위치에 보상 -1
        self.reward[4][2] = -1  # (4,2) 좌표 세모 위치에 보상 -1
        self.reward[5][3] = -1  # (5,3) 좌표 세모 위치에 보상 -1
        self.all_state = []
```

▶ Grid map 에서의 reward 값이 저장된 위치를 문제 요구사항에 맞도록 수정하였음.

4. Value_Iteration/value_iteration.py 파일 수정사항

```python
class ValueIteration:
    def __init__(self, env):
        # 환경에 대한 객체 선언
        self.env = env
        # 가치 함수를 2차원 리스트로 초기화
        self.value_table = [[0.0] * env.width for _ in range(env.height)]
        # 할인율
        self.discount_factor = 0.9

    # 벨만 최적 방정식을 통해 다음 가치 함수 계산
    def value_iteration(self):
        # 다음 가치함수 초기화
        next_value_table = [[0.0] * self.env.width
                            for _ in range(self.env.height)]

        # 모든 상태에 대해서 벨만 최적방정식을 계산
        for state in self.env.get_all_states():
            # 마침 상태의 가치 함수 = 0
            if state == [2, 2]:
                next_value_table[state[0]][state[1]] = 0.0
                continue
```

```python
class ValueIteration:
    def __init__(self, env):
        # 환경에 대한 객체 선언
        self.env = env
        # 가치 함수를 2차원 리스트로 초기화
        self.value_table = [[0.0] * env.width for _ in range(env.height)]
        # 할인율
        self.discount_factor = 0.9

    # 벨만 최적 방정식을 통해 다음 가치 함수 계산
    def value_iteration(self):
        # 다음 가치함수 초기화
        next_value_table = [[0.0] * self.env.width
                            for _ in range(self.env.height)]

        # 모든 상태에 대해서 벨만 최적방정식을 계산
        for state in self.env.get_all_states():
            # 마침 상태의 가치 함수 = 0
            if state == [3, 4]:
                next_value_table[state[0]][state[1]] = 0.0
                continue
```

```python
# 현재 가치 함수로부터 행동을 반환
def get_action(self, state):
    if state == [2, 2]:
        return []
```

```python
# 현재 가치 함수로부터 행동을 반환
def get_action(self, state):
    if state == [3, 4]:
        return []
```

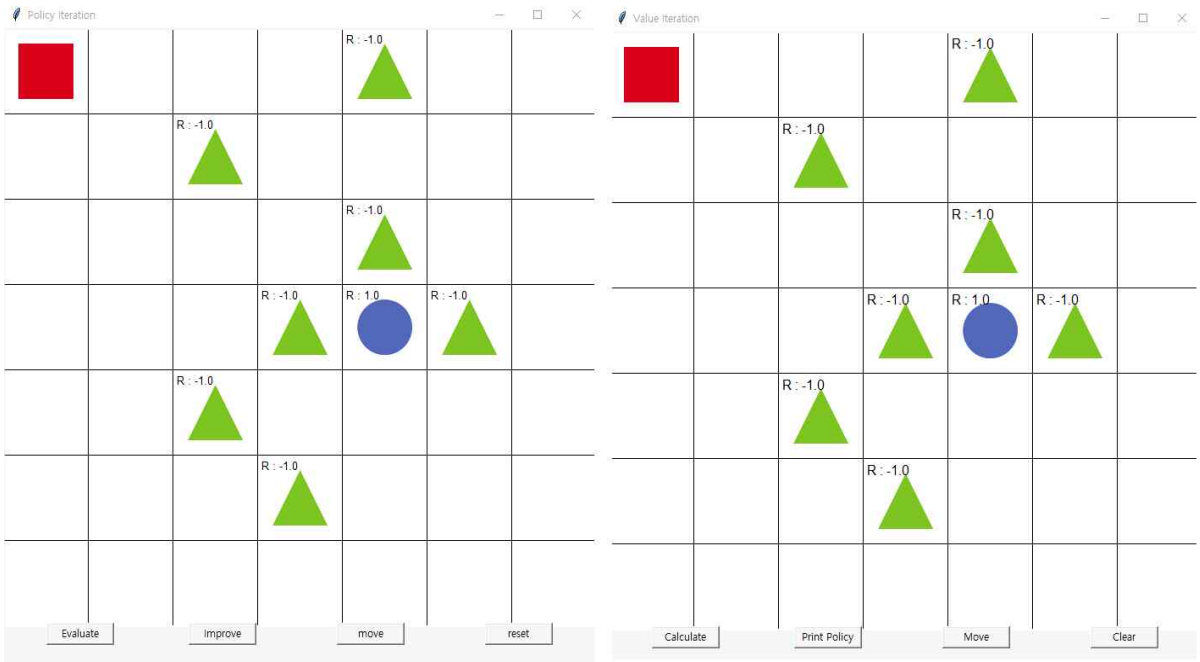
▶ 마침 상태를 문제 요구사항에 맞도록 수정하였음.

※ Policy Iteration 과 Value Iteration 의 수행 결과가 동일하므로, 마지막에 두 결과를 한번에 관측하도록 편집하였음.

< Policy Iteration 과 Value Iteration 수행 결과 화면 >

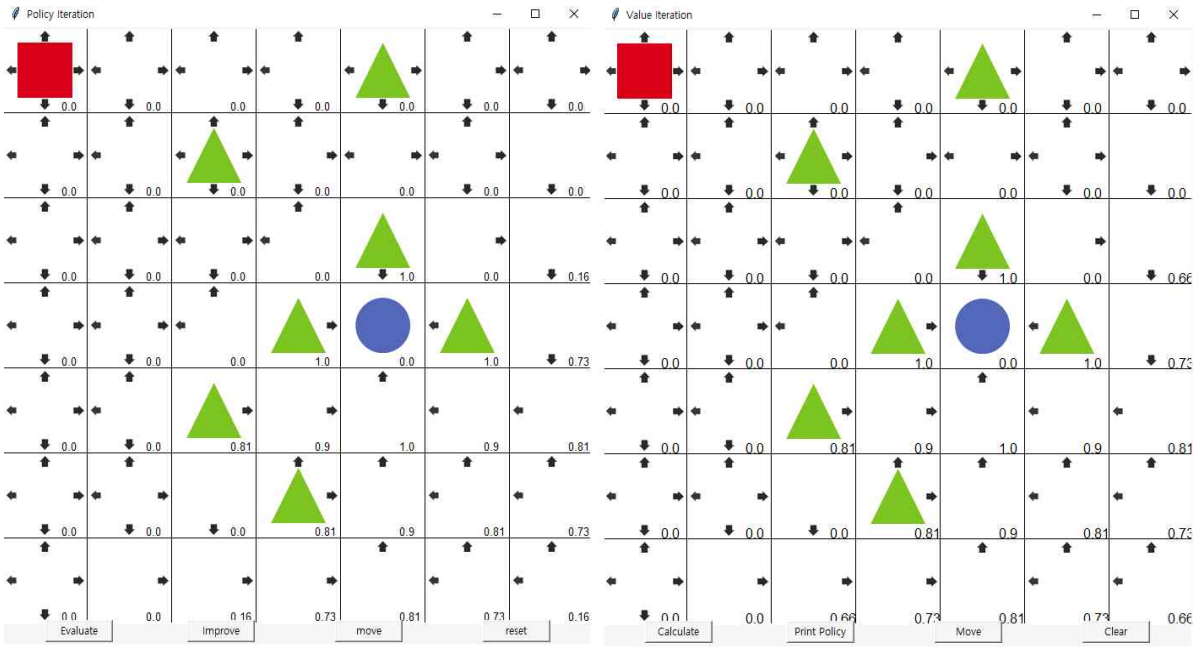
※ 좌측은 Policy Iteration, 우측은 Value Iteration 화면

1. k = 0



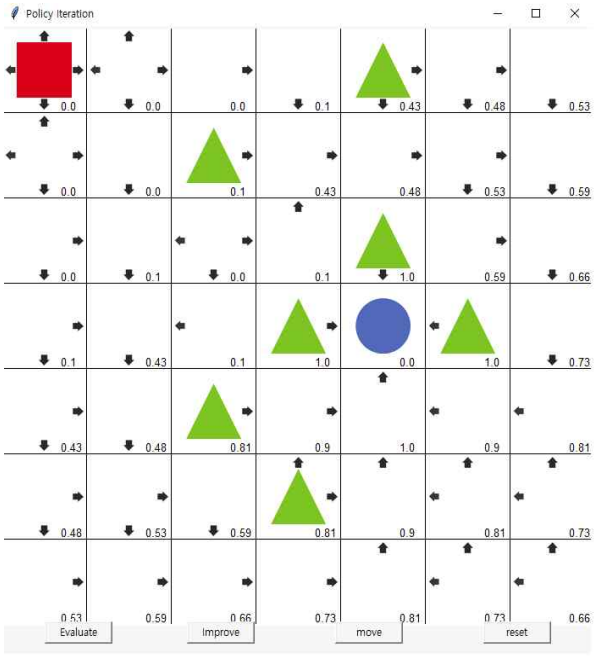
▶ hazard(세모) 와 reward(좌측 상단 텍스트) 의 값과, 각 위치가 정상적으로 출력되었으며, 요구사항과 일치함을 확인하였음.

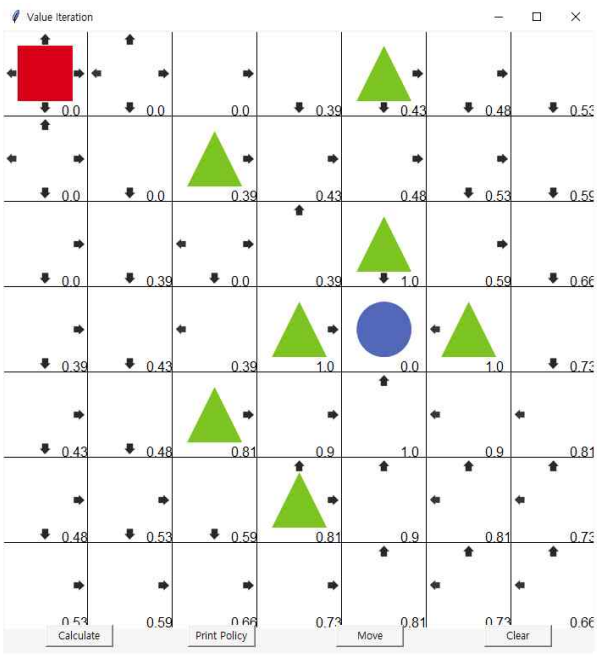
2. k = 5



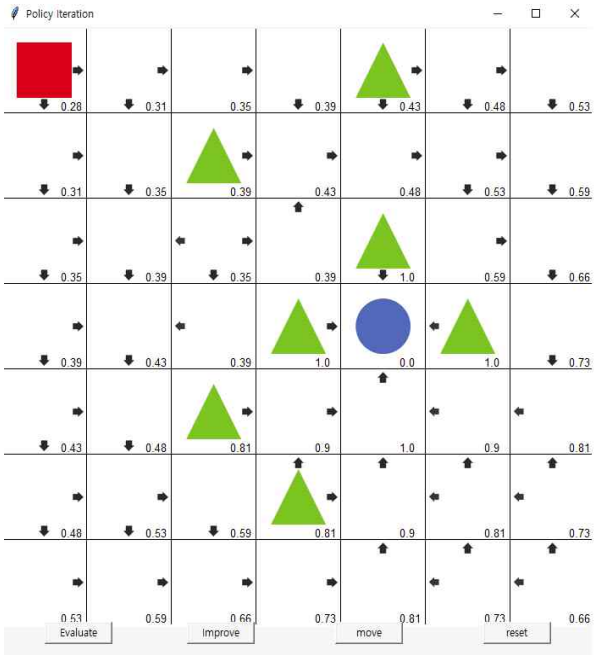
▶ 각 좌표에서의 화살표(행동)와 우측 하단 숫자(가치함수)의 값이 변화함을 관측하였음.

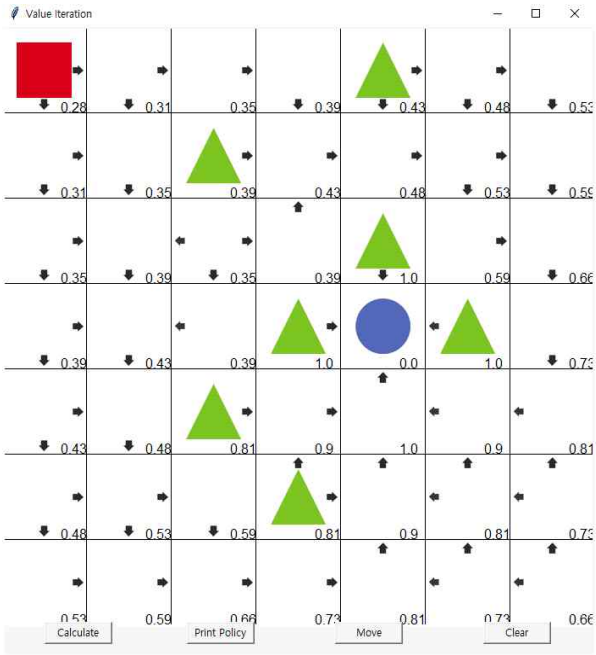
3. k = 10





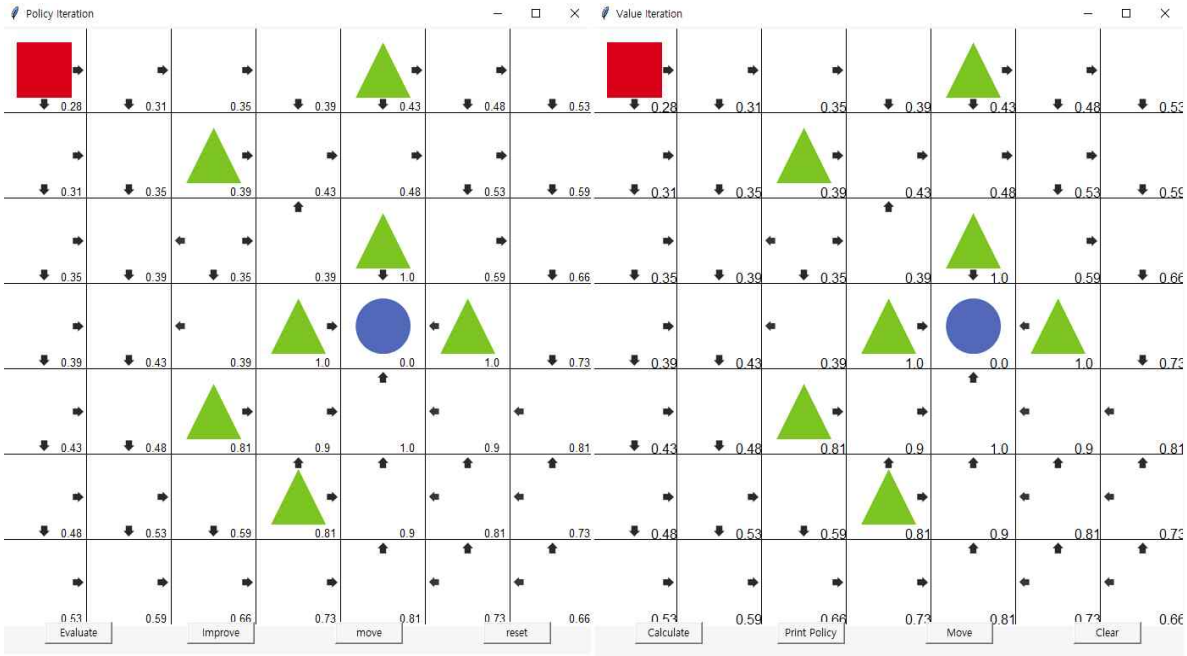
4. k = 20





▶ 각 좌표에서의 화살표(행동)와 우측 하단 숫자(가치함수)의 값이 변화함을 관측하였음.

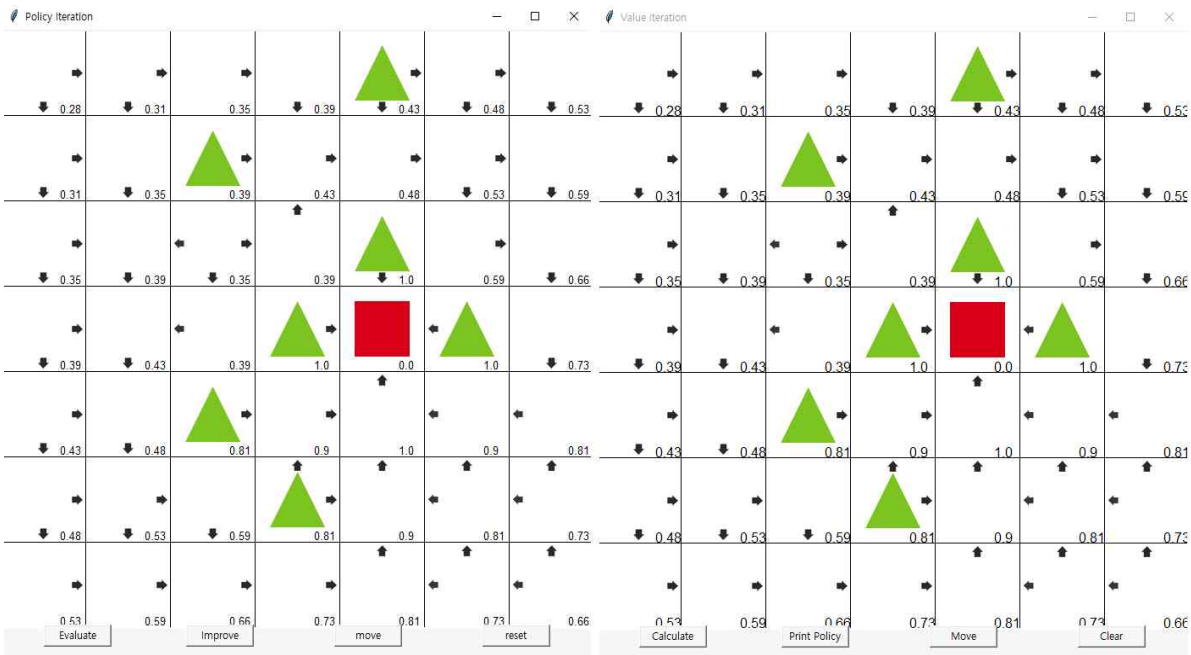
5. k = 50



▶ k = 50까지 수행한 결과. k = 20 이후로 변화가 없음을 관측하였음.

▶ 따라서, 20번 정책평가 및 개선 이후 최적의 정책을 찾았다고 생각함.

6. move 결과



▶ 위 화면과 같이 hazard(세모) 를 피해 목표 지점(동그라미)으로 잘 이동하였음을 확인함.

▶ 결과는 위와 항상 동일하지만, 경로는 경우에 따라 다름도 확인하였음.

▶ 이는 최초 상태에서 취하는 행동 두 가지(우, 하)의 다음 가치함수 값이 동일하기 때문이라고 생각함.