지능시스템

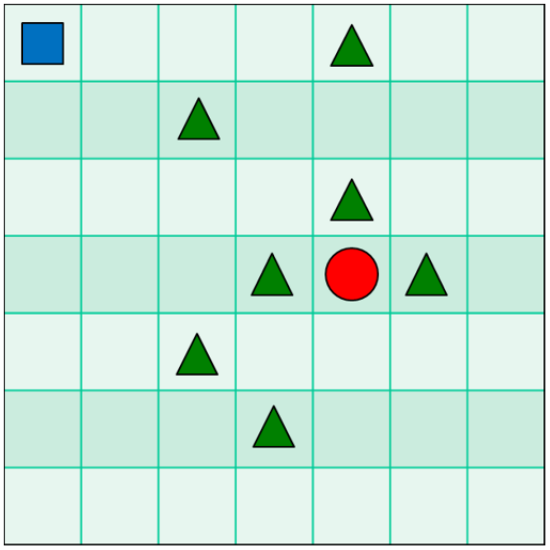
과제4

2019305059

이현수

숙제 4 (Dynamic Programming)

- 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)



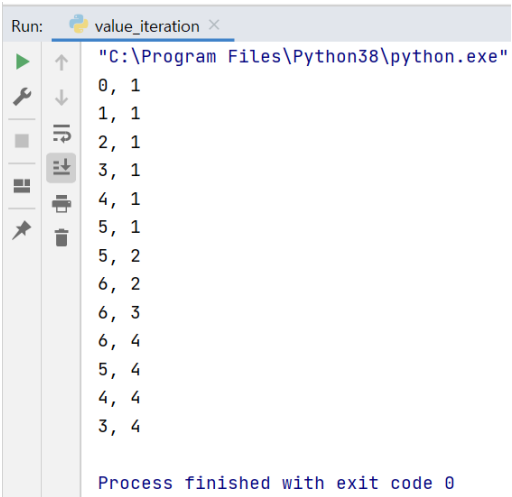
Rewards

▲ 상태로 가는 행동: −1

● 상태로 가는 행동: +1

- Iteration회수를 5, 10, 20으로 실행해 각 상태에서의 value table값과 action list를 결과로 제시함.

<- Action list를 출력할때의 좌표값 위치



Action list는 위 사진과 같이 출력됨.

●Policy Iteration

<수정한 코드>

environment.py

```python
import tkinter as tk
from tkinter import Button
import time
import numpy as np
from PIL import ImageTk, Image

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 = []


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(3, 4, "R : 1.0")
        self.text_reward(0, 4, "R : -1.0")
        self.text_reward(1, 2, "R : -1.0")
        self.text_reward(4, 2, "R : -1.0")
        self.text_reward(3, 3, "R : -1.0")
        self.text_reward(5, 3, "R : -1.0")
        self.text_reward(2, 4, "R : -1.0")
        self.text_reward(3, 5, "R : -1.0")

```

5*5에서 7*7로 변경

문제에서 주어진대로 보상을 텍스트로 출력

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

        canvas.pack()

        return canvas
```

문제에서 주어진대로 이미지 삽입

```python
    def rectangle_move(self, action,x,y):
        base_action = np.array([0, 0])
        location = self.find_rectangle()
        self.render()
        if action == 0 and location[0] > 0:  # 상
            base_action[1] -= UNIT
            print(str(x-1)+', '+str(y))
        elif action == 1 and location[0] < HEIGHT - 1:  # 하
            base_action[1] += UNIT
            print(str(x + 1) + ', ' + str(y))
        elif action == 2 and location[1] > 0:  # 좌
            base_action[0] -= UNIT
            print(str(x) + ', ' + str(y-1))
        elif action == 3 and location[1] < WIDTH - 1:  # 우
            base_action[0] += UNIT
            print(str(x) + ', ' + str(y+1))
        # move agent
        self.canvas.move(self.rectangle, base_action[0], base_action[1])

    def find_rectangle(self):
        temp = self.canvas.coords(self.rectangle)
        x = (temp[0] / 100) - 0.5
        y = (temp[1] / 100) - 0.5
        return int(y), int(x)

```

Rectangle_move 메소드에 x, y 매개변수 추가

Up으로 움직이면 (x-1), y 출력

Down으로 움직이면 (x+1), y 출력

Left으로 움직이면 x, (y-1) 출력

Right으로 움직이면 x, (y+1) 출력

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

        if policy[0] > 0:  # up
            origin_x, origin_y = 50 + (UNIT * row), 10 + (UNIT * col)
            self.arrows.append(self.canvas.create_image(origin_x, origin_y,
                                                        image=self.up))
        if policy[1] > 0:  # down
            origin_x, origin_y = 50 + (UNIT * row), 90 + (UNIT * col)
            self.arrows.append(self.canvas.create_image(origin_x, origin_y,
                                                        image=self.down))
        if policy[2] > 0:  # left
            origin_x, origin_y = 10 + (UNIT * row), 50 + (UNIT * col)
            self.arrows.append(self.canvas.create_image(origin_x, origin_y,
                                                        image=self.left))
        if policy[3] > 0:  # right
            origin_x, origin_y = 90 + (UNIT * row), 50 + (UNIT * col)
            self.arrows.append(self.canvas.create_image(origin_x, origin_y,
                                                        image=self.right))

```

col==2, row==2 ➔ col==3, row==4

```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
        self.reward[1][2] = -1
        self.reward[0][4] = -1
        self.reward[4][2] = -1
        self.reward[3][3] = -1
        self.reward[3][5] = -1
        self.reward[2][4] = -1
        self.reward[5][3] = -1
        self.all_state = []

        for x in range(WIDTH):
            for y in range(HEIGHT):
                state = [x, y]
                self.all_state.append(state)

```

문제에서 주어진대로 보상을 입력

policy_iteration.py

```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] = []
        # 감가율
        self.discount_factor = 0.9

    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]] = round(value, 2)

        self.value_table = next_value_table
```

self.policy_table[2][2] → self.policy_table[3][4]

[2, 2] → [3, 4]

```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
            value = -99999
            max_index = []
            # 반환할 정책 초기화
            result = [0.0, 0.0, 0.0, 0.0]

            # 모든 행동에 대해서 [보상 + (감가율 * 다음 상태 가치함수)] 계산
            for index, action in enumerate(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)
                temp = reward + self.discount_factor * next_value

                # 받을 보상이 최대인 행동의 index(최대가 복수라면 모두)를 추출
                if temp == value:
                    max_index.append(index)
                elif temp > value:
                    value = temp
                    max_index.clear()
                    max_index.append(index)

            # 행동의 확률 계산
            prob = 1 / len(max_index)

            for index in max_index:
                result[index] = prob

            next_policy[state[0]][state[1]] = result

        self.policy_table = next_policy

```

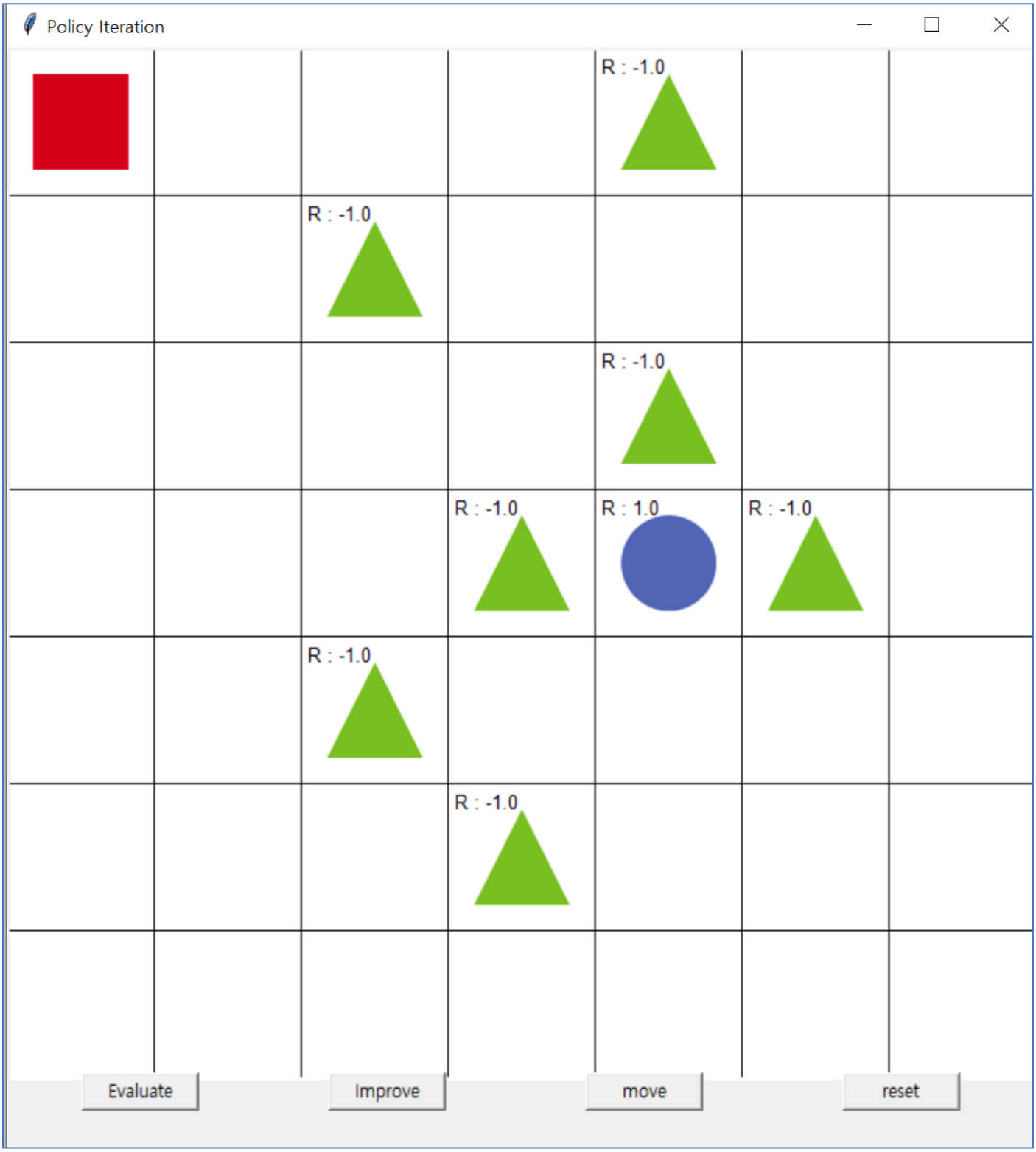
[2, 2] → [3, 4]

```python
    # 상태에 따른 정책 반환
    def get_policy(self, state):
        if state == [3, 4]:
            return 0.0
        return self.policy_table[state[0]][state[1]]

```

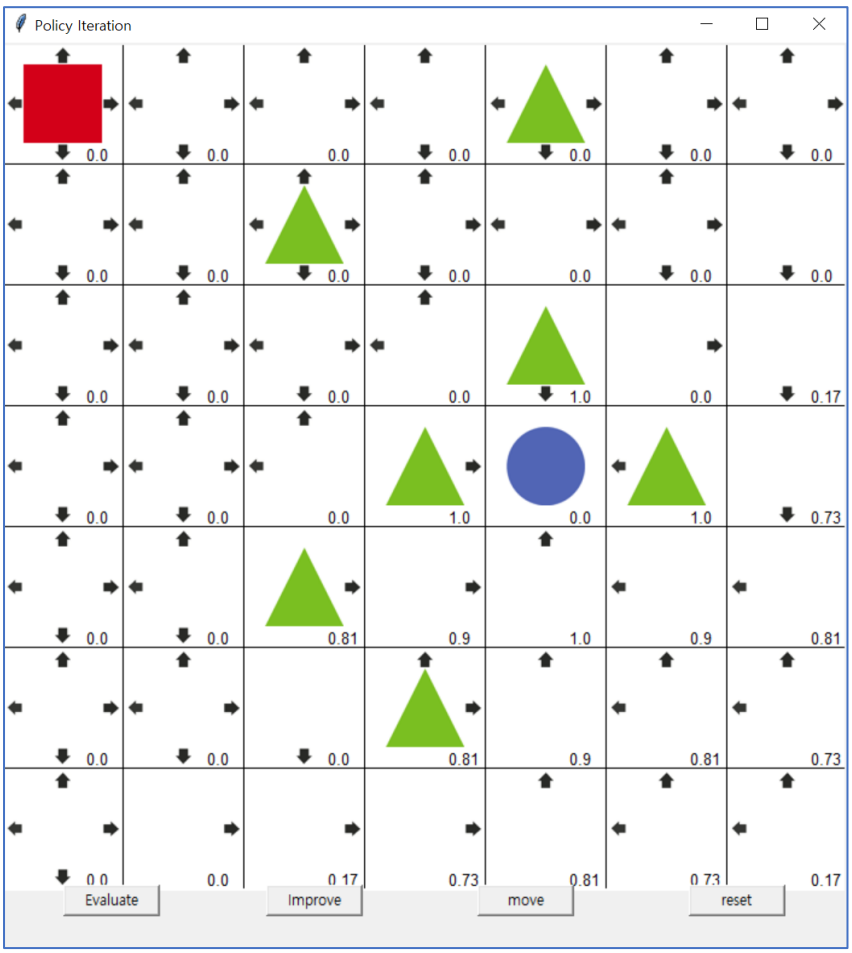
[2, 2] → [3, 4]

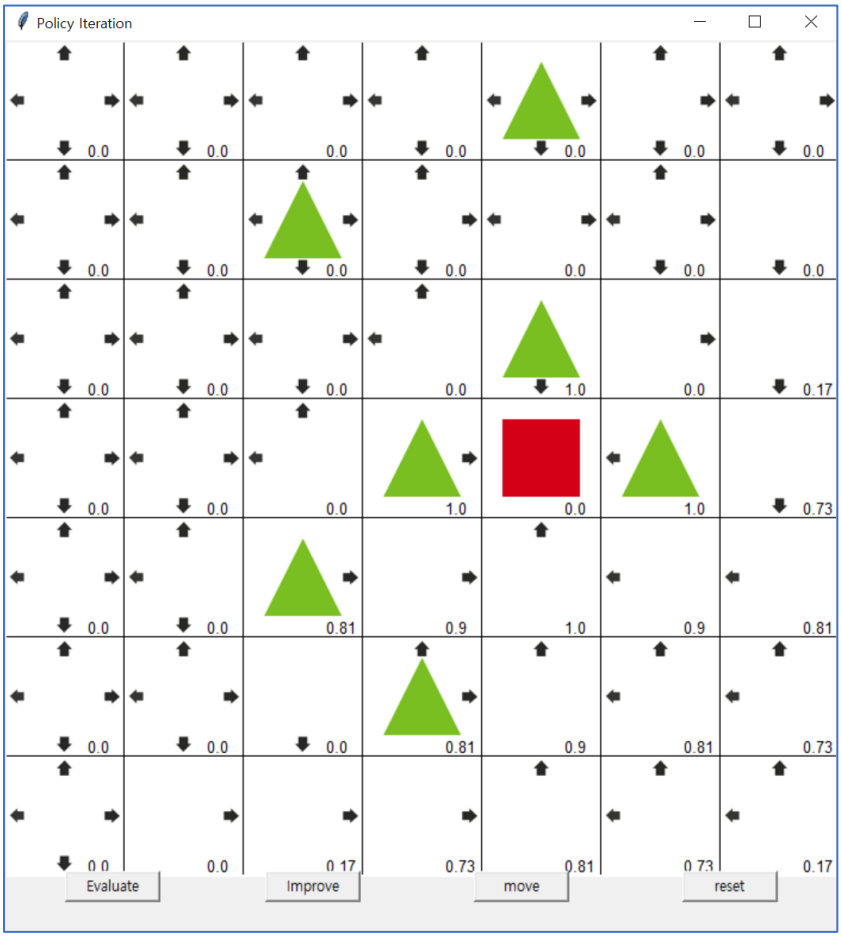
<실행>

Iteration회수 = 1 일 때 value table



Iteration회수 = 5 일 때 value table

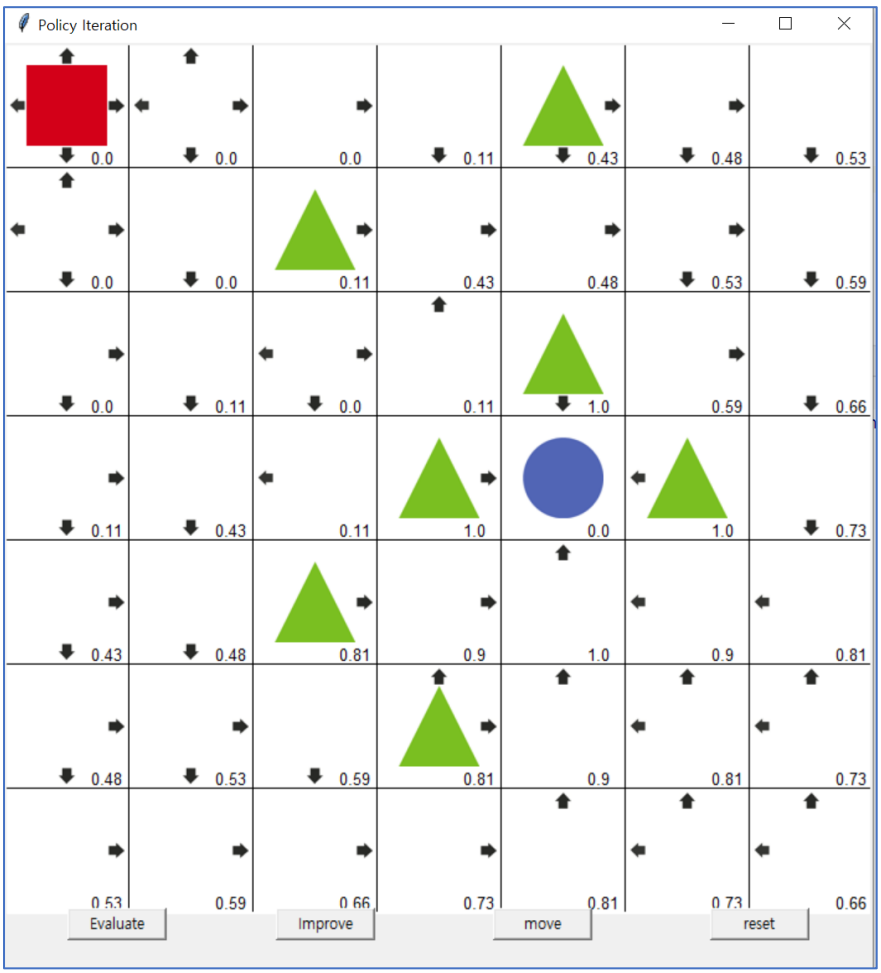


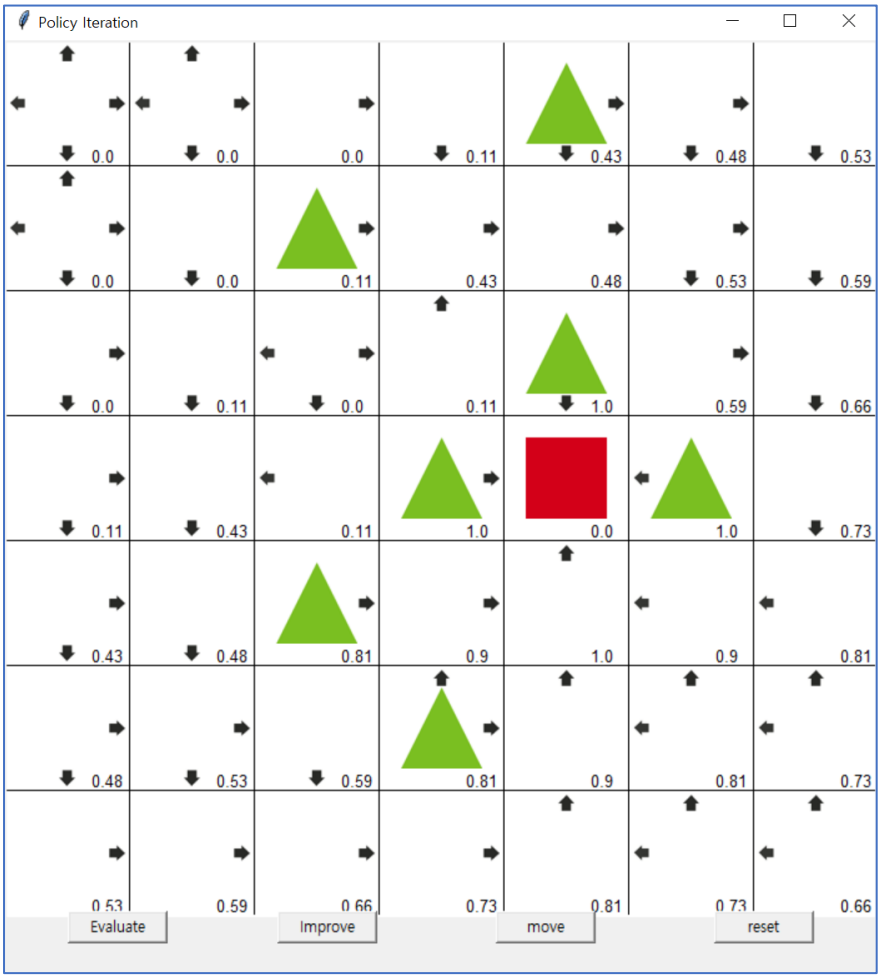


Iteration회수 = 5 일 때 action list

1, 0
0, 0
0, 1
0, 2
0, 3
1, 3
1, 4
1, 3
2, 3
1, 3
2, 3
1, 3
0, 3
0, 2
0, 1
1, 1
1, 0
0, 0
1, 0
2, 0
3, 0
3, 1
4, 1
5, 1
6, 1
6, 2
6, 3
6, 4
5, 4
4, 4
3, 4

Iteration회수 = 10 일 때 value table





Iteration회수 = 10 일 때 action list

0, 1

0, 0

1, 0

0, 0

1, 0

2, 0

3, 0

3, 1

4, 1

5, 1

6, 1

6, 2

6, 3

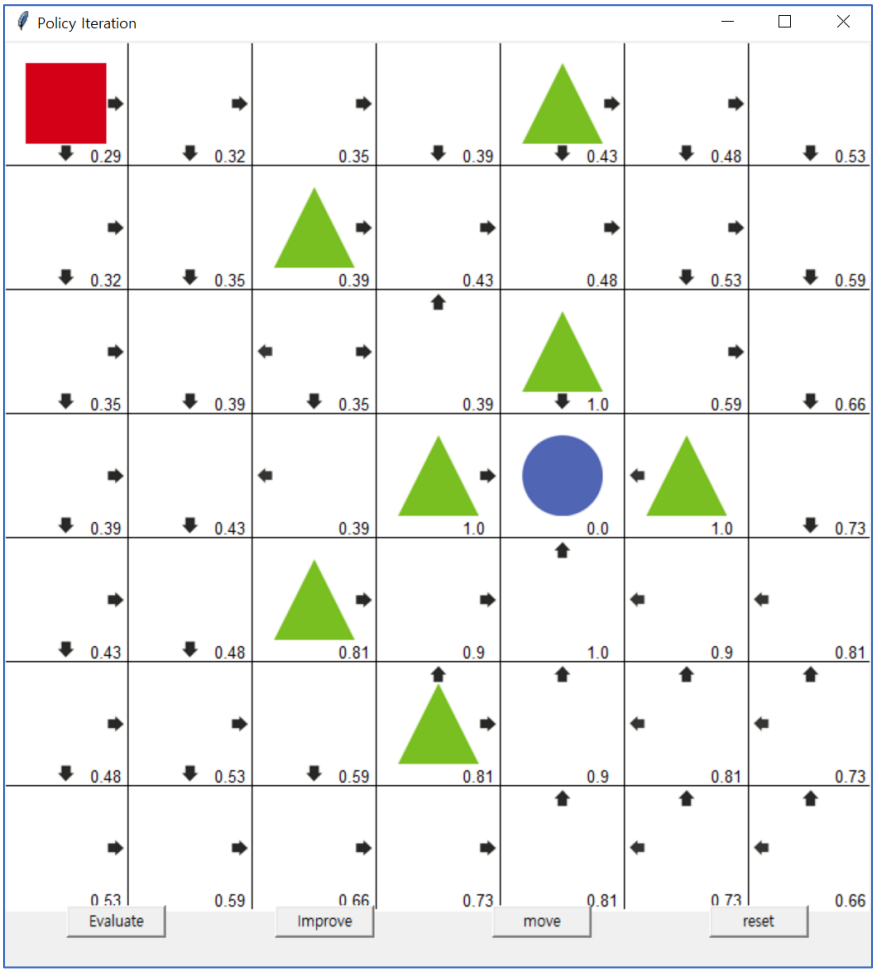
6, 4

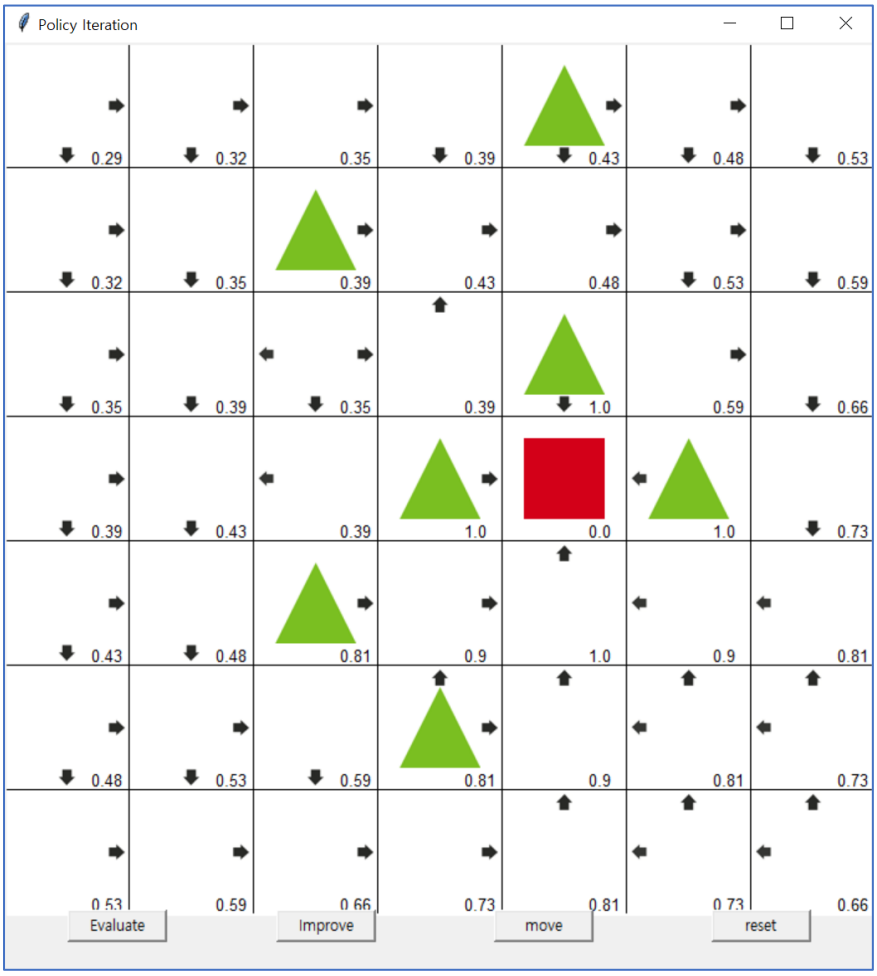
5, 4

4, 4

3, 4

Iteration회수 = 20 일 때 value table





Iteration회수 = 20 일 때 action list

0, 1

1, 1

2, 1

3, 1

4, 1

5, 1

6, 1

6, 2

6, 3

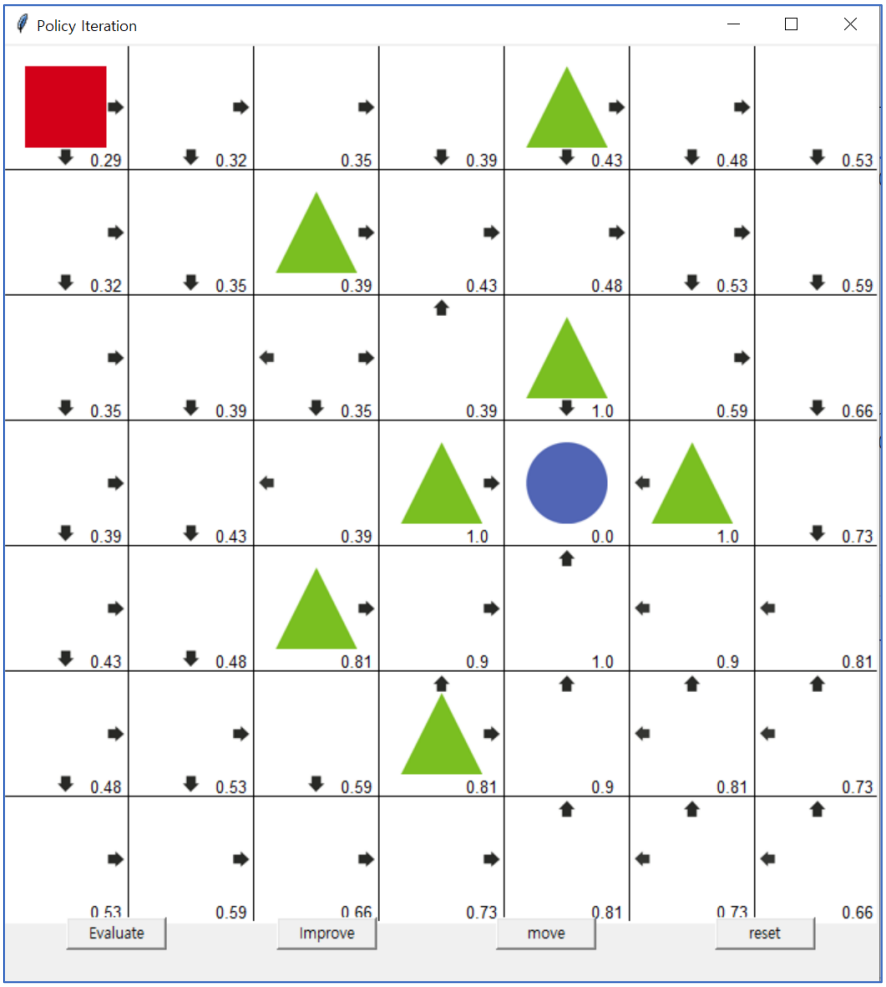
6, 4

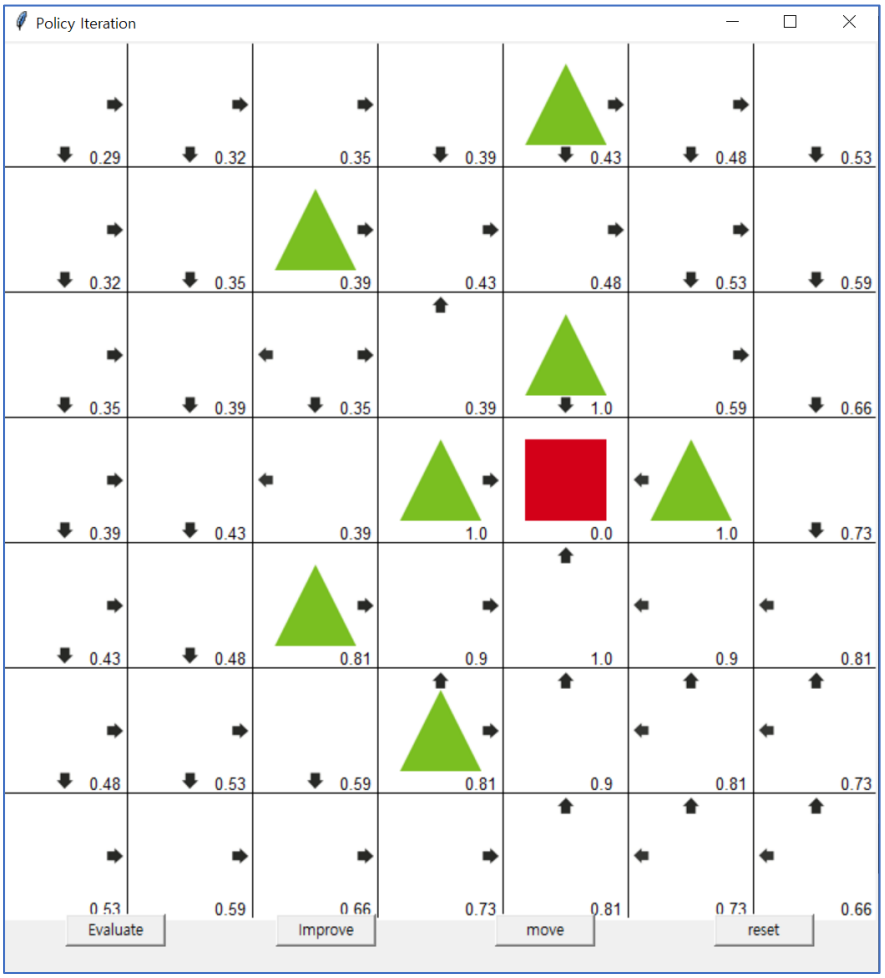
5, 4

4, 4

3, 4

Iteration회수 = 50 일 때 value table





Iteration회수 = 50 일 때 action list

0, 1

0, 2

0, 3

1, 3

1, 4

1, 5

1, 6

2, 6

3, 6

4, 6

4, 5

4, 4

3, 4

●Value Iteration

<수정한 코드>

environment.py

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

5*5에서 7*7로 변경

```python
class GraphicDisplay(tk.Tk):
    def __init__(self, value_iteration):
        super(GraphicDisplay, self).__init__()
        self.title('Value Iteration')
        self.geometry('{0}x{1}'.format(HEIGHT * UNIT, HEIGHT * UNIT + 50))
        self.texts = []
        self.arrows = []
        self.env = Env()
        self.agent = value_iteration
        self.iteration_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")
        self.text_reward(0, 4, "R : -1.0")
        self.text_reward(1, 2, "R : -1.0")
        self.text_reward(4, 2, "R : -1.0")
        self.text_reward(3, 3, "R : -1.0")
        self.text_reward(5, 3, "R : -1.0")
        self.text_reward(2, 4, "R : -1.0")
        self.text_reward(3, 5, "R : -1.0")
```

문제에서 주어진대로 보상을 텍스트로 출력

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

        canvas.pack()

        return canvas
```

문제에서 주어진대로 이미지 삽입

```python
    def rectangle_move(self, action,x,y):
        base_action = np.array([0, 0])
        location = self.find_rectangle()
        self.render()
        if action == 0 and location[0] > 0:  # up
            base_action[1] -= UNIT
            print(str(x-1)+', '+str(y))
        elif action == 1 and location[0] < HEIGHT - 1:  # down
            base_action[1] += UNIT
            print(str(x + 1) + ', ' + str(y))
        elif action == 2 and location[1] > 0:  # left
            base_action[0] -= UNIT
            print(str(x) + ', ' + str(y-1))
        elif action == 3 and location[1] < WIDTH - 1:  # right
            base_action[0] += UNIT
            print(str(x) + ', ' + str(y+1))

        self.canvas.move(self.rectangle, base_action[0],
                         base_action[1])  # move agent

```

Rectangle_move 메소드에 x, y 매개변수 추가

Up으로 움직이면 (x-1), y 출력

Down으로 움직이면 (x+1), y 출력

Left으로 움직이면 x, (y-1) 출력

Right으로 움직이면 x, (y+1) 출력

```python
    def draw_one_arrow(self, col, row, action):
        if col == 3 and row == 4:
            return
        if action == 0:  # up
            origin_x, origin_y = 50 + (UNIT * row), 10 + (UNIT * col)
            self.arrows.append(self.canvas.create_image(origin_x, origin_y,
                                                        image=self.up))
        elif action == 1:  # down
            origin_x, origin_y = 50 + (UNIT * row), 90 + (UNIT * col)
            self.arrows.append(self.canvas.create_image(origin_x, origin_y,
                                                        image=self.down))
        elif action == 3:  # right
            origin_x, origin_y = 90 + (UNIT * row), 50 + (UNIT * col)
            self.arrows.append(self.canvas.create_image(origin_x, origin_y,
                                                        image=self.right))
        elif action == 2:  # left
            origin_x, origin_y = 10 + (UNIT * row), 50 + (UNIT * col)
            self.arrows.append(self.canvas.create_image(origin_x, origin_y,
                                                        image=self.left))

```

col==2, row==2 ➔ col==3, row==4

```python
class Env:
    def __init__(self):
        self.transition_probability = TRANSITION_PROB
        self.width = WIDTH  # Width of Grid World
        self.height = HEIGHT  # Height of GridWorld
        self.reward = [[0] * WIDTH for _ in range(HEIGHT)]
        self.possible_actions = POSSIBLE_ACTIONS
        self.reward[3][4] = 1
        self.reward[1][2] = -1
        self.reward[0][4] = -1
        self.reward[4][2] = -1
        self.reward[3][3] = -1
        self.reward[3][5] = -1
        self.reward[2][4] = -1
        self.reward[5][3] = -1
        self.all_state = []

```

문제에서 주어진대로 보상을 입력

value_iteration.py

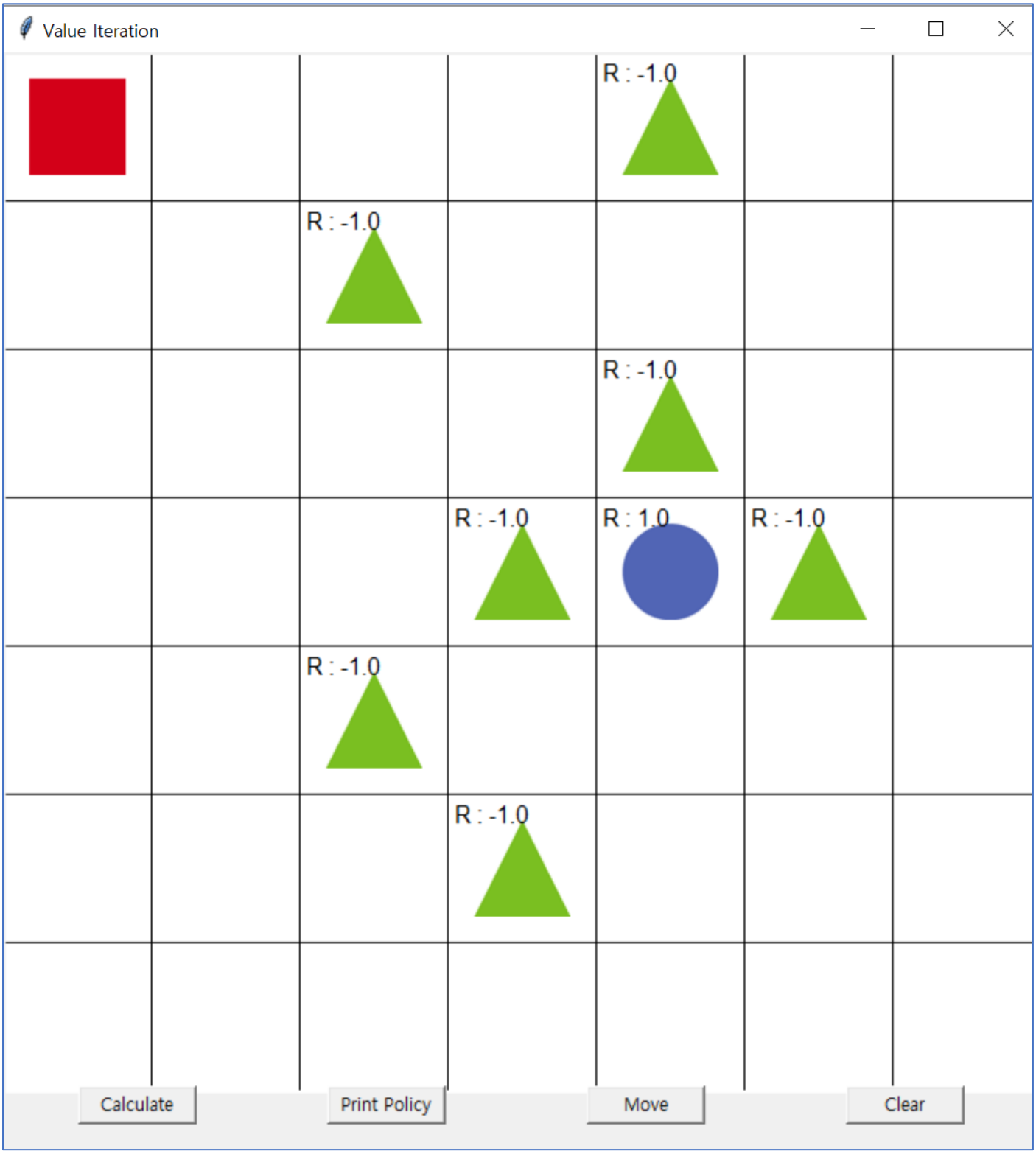
```python
    def value_iteration(self):
        next_value_table = [[0.0] * self.env.width for _ in
                            range(self.env.height)]
        for state in self.env.get_all_states():
            if state == [3, 4]:
                next_value_table[state[0]][state[1]] = 0.0
                continue
            # 가치 함수를 위한 빈 리스트
            value_list = []

            # 가능한 모든 행동에 대해 계산
            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_list.append((reward + self.discount_factor * next_value))
            # 최댓값을 다음 가치 함수로 대입
            next_value_table[state[0]][state[1]] = round(max(value_list), 2)
        self.value_table = next_value_table

    # 현재 가치 함수로부터 행동을 반환
    def get_action(self, state):
        action_list = []
        max_value = -99999

        if state == [3, 4]:
            return []

        # 모든 행동에 대해 큐함수 (보상 + (감가율 * 다음 상태 가치함수))를 계산
        # 최대 큐 함수를 가진 행동(복수일 경우 여러 개)을 반환
        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 = (reward + self.discount_factor * next_value)

            if value > max_value:
                action_list.clear()
                action_list.append(action)
                max_value = value
            elif value == max_value:
                action_list.append(action)

        return action_list
```

[2, 2] → [3, 4]

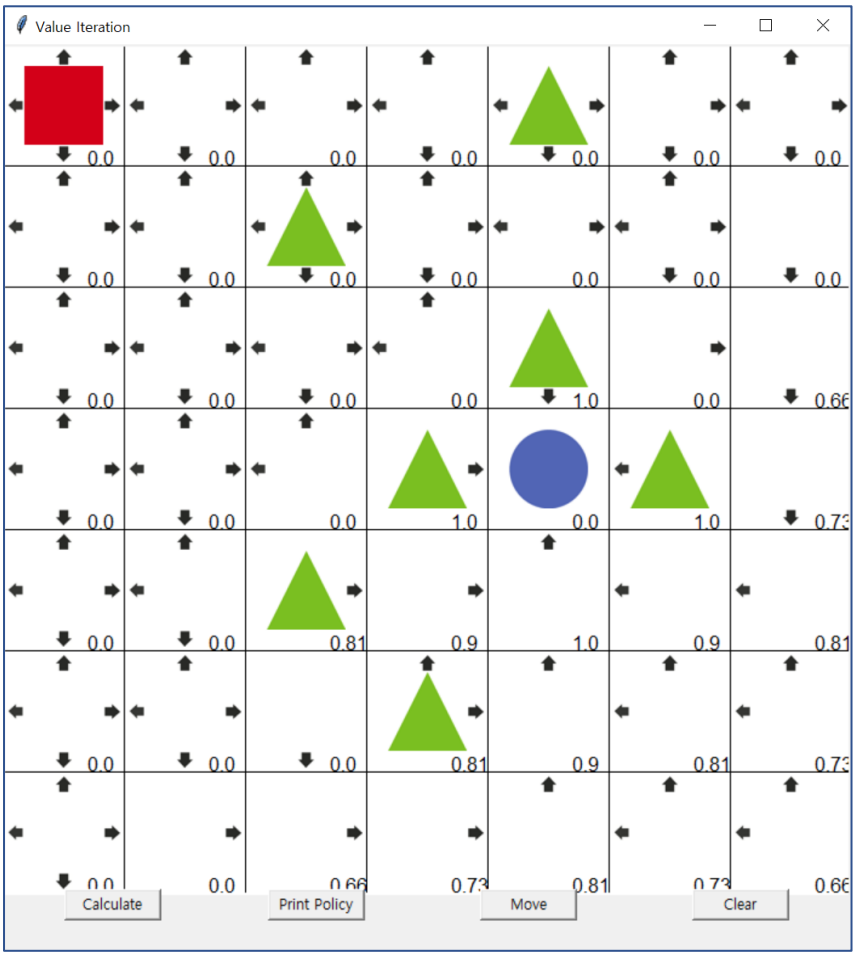
[2, 2] → [3, 4]

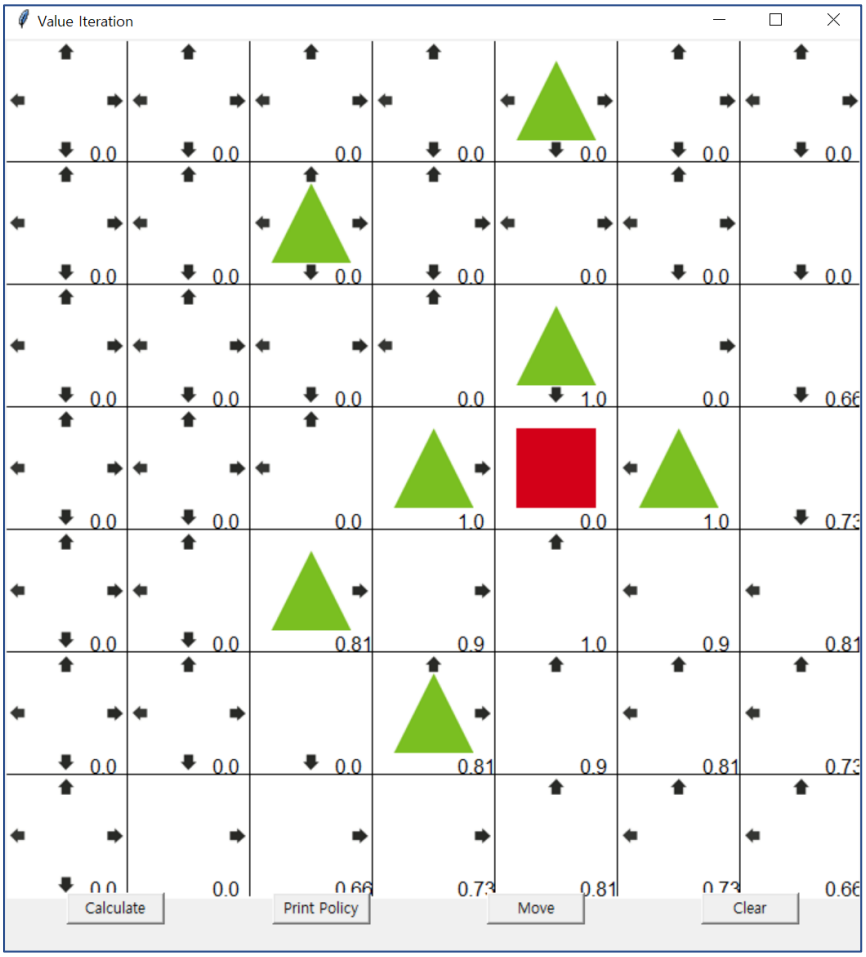
<실행>

Iteration회수 = 0 일 때 value table



Iteration회수 = 5 일 때 value table





Iteration회수 = 5 일 때 action list

0, 1

0, 0

1, 0

0, 0

0, 1

0, 2

0, 3

1, 3

1, 4

1, 5

1, 4

1, 3

1, 4

1, 5

1, 6

2, 6

3, 6

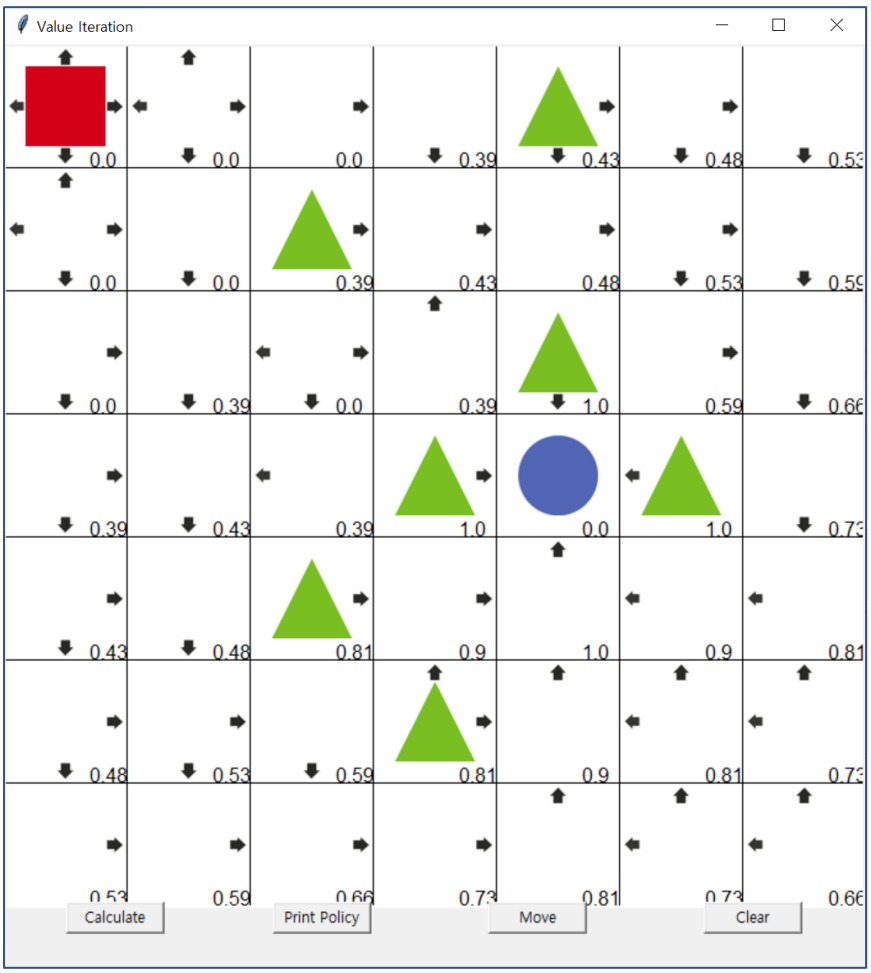
4, 6

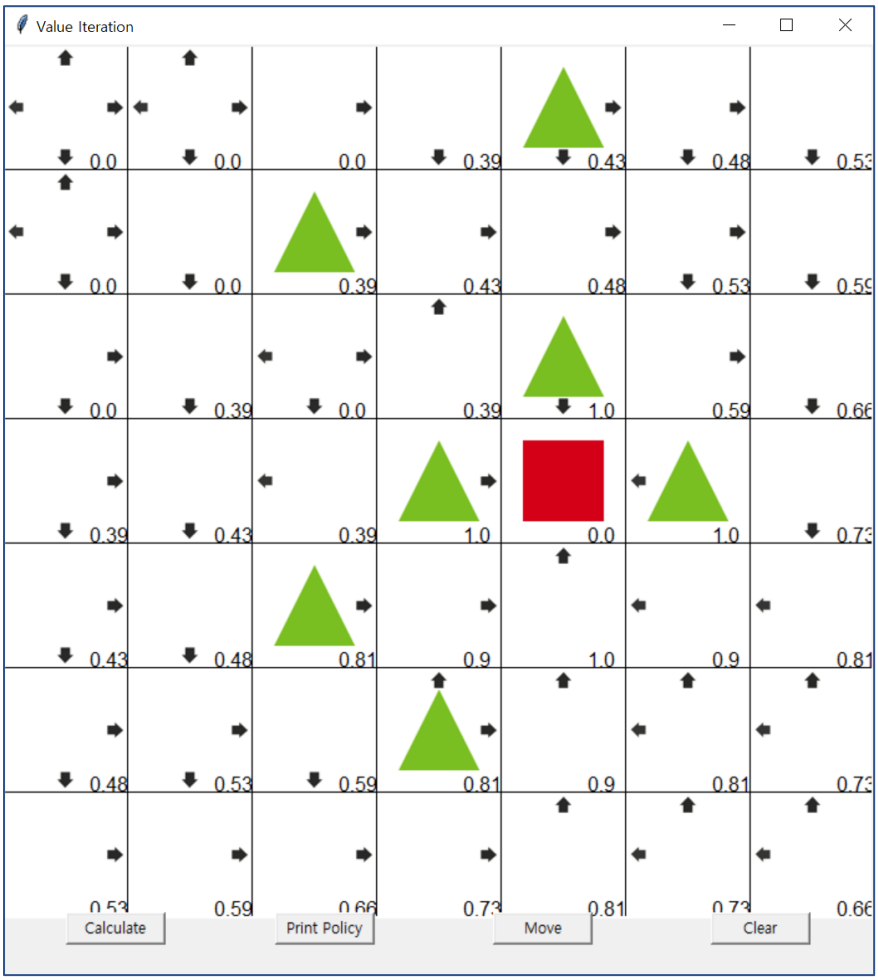
4, 5

4, 4

3, 4

Iteration회수 = 10 일 때 value table





Iteration회수 = 10 일 때 action list

1, 0

2, 0

2, 1

3, 1

4, 1

5, 1

6, 1

6, 2

6, 3

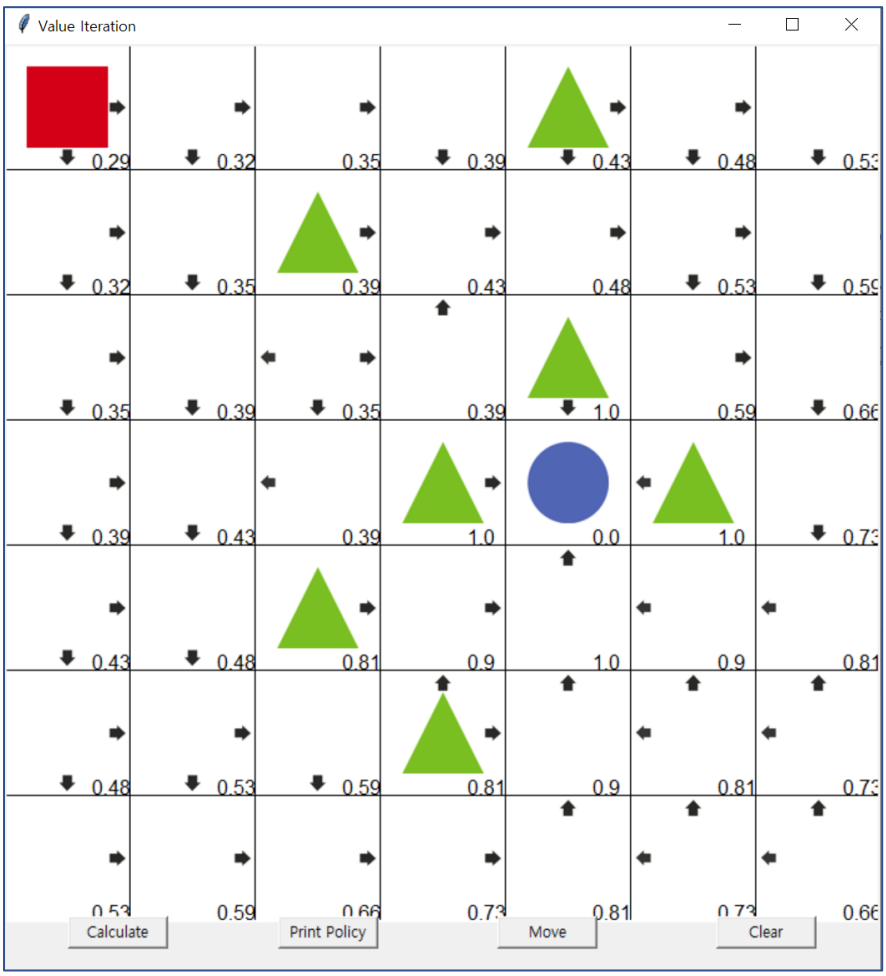
6, 4

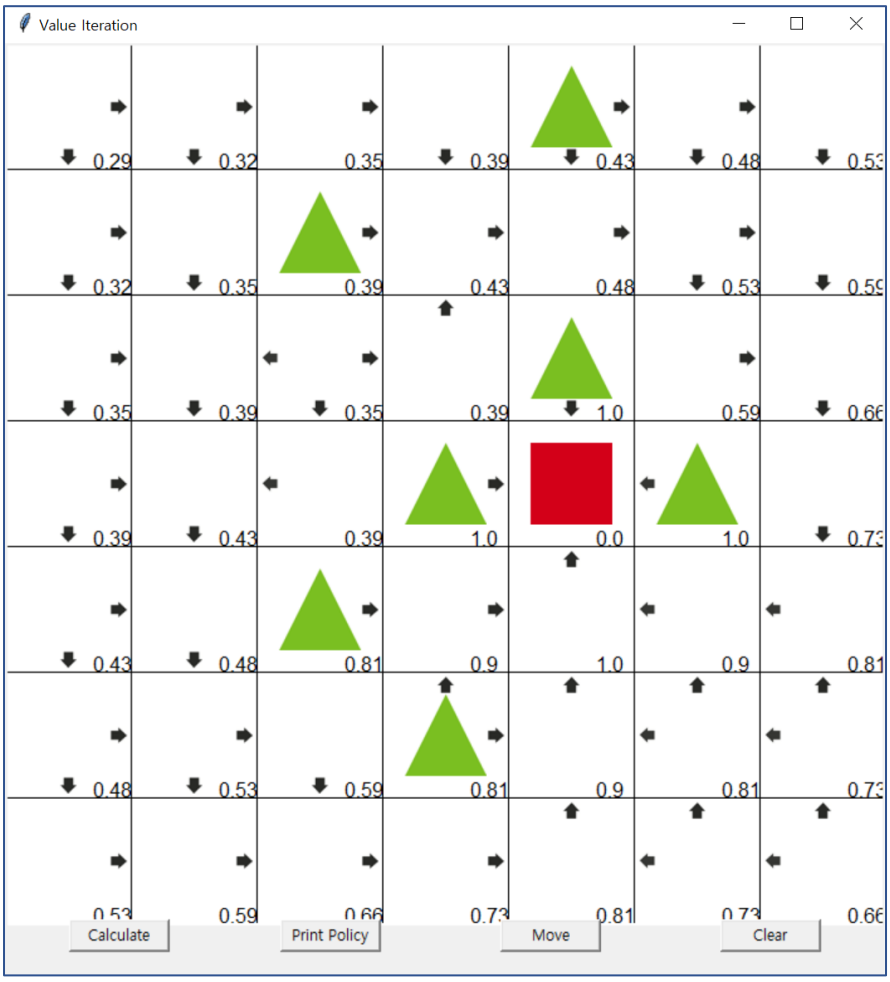
5, 4

4, 4

3, 4

Iteration회수 = 20 일 때 value table





Iteration회수 = 20 일 때 action list

0, 1

1, 1

2, 1

3, 1

4, 1

5, 1

5, 2

6, 2

6, 3

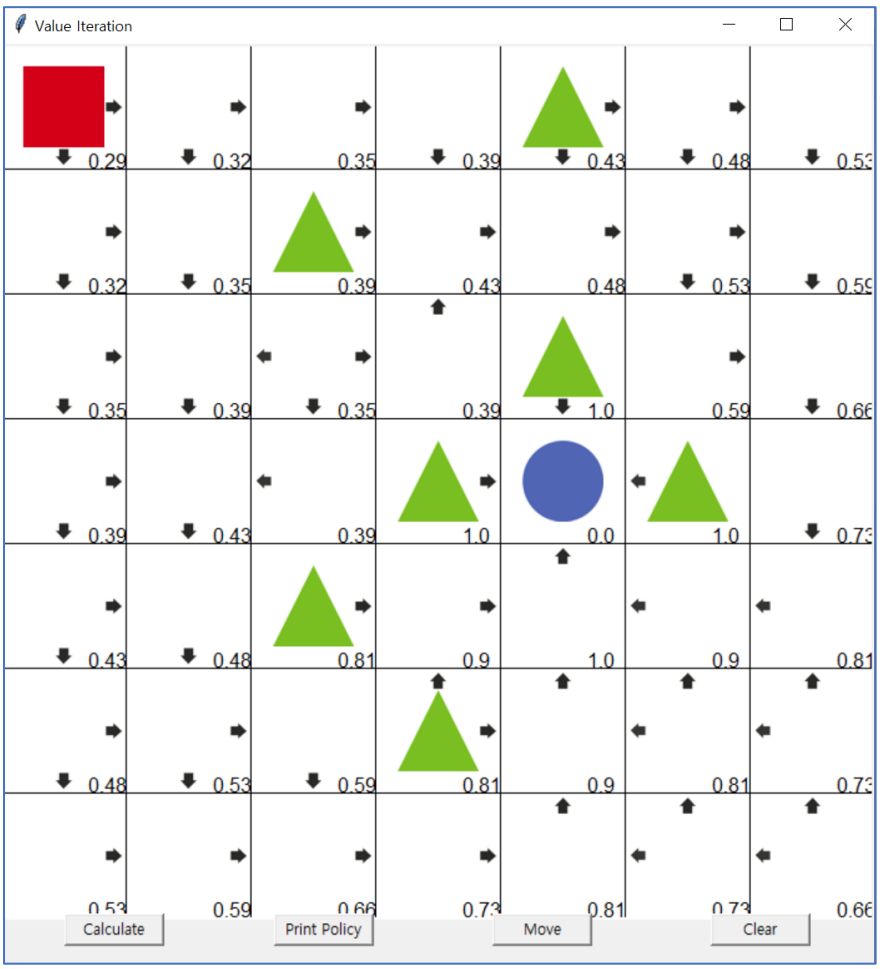
6, 4

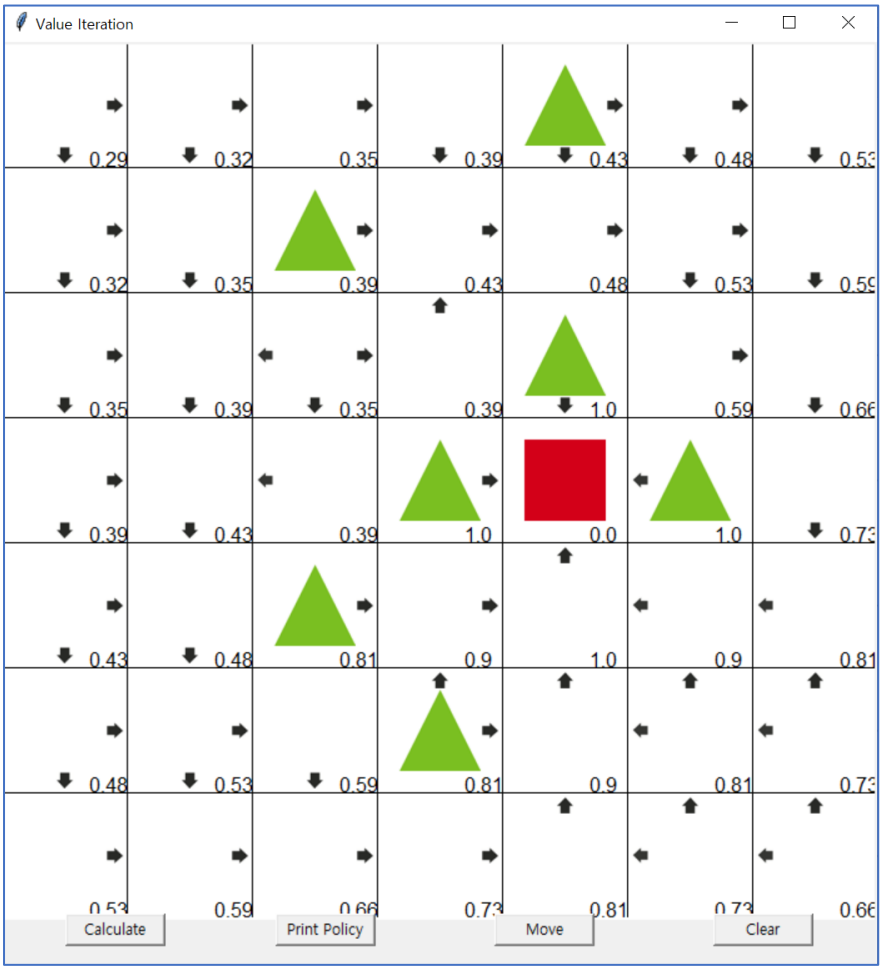
5, 4

4, 4

3, 4

Iteration회수 = 50 일 때 value table





Iteration회수 = 50 일 때 action list

0, 1

1, 1

2, 1

3, 1

4, 1

5, 1

5, 2

6, 2

6, 3

6, 4

5, 4

4, 4

3, 4