

# 지능시스템

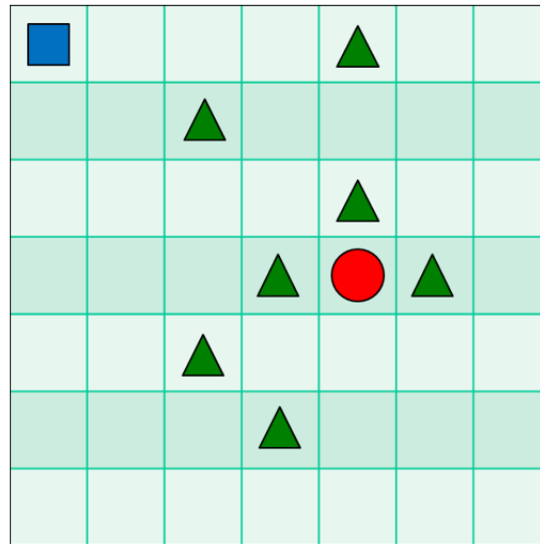
## 과제4

2019305059

이현수

## 숙제 4 (Dynamic Programming)

- 다음의  $7 \times 7$  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

```
Run: value_iteration x
"C:\Program Files\Python38\python.exe"
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
Process finished with exit code 0
```

Action list는 위 사진과 같이 출력됨.

# ● Policy Iteration

<수정한 코드>

environment.py

```
1 import tkinter as tk
2 from tkinter import Button
3 import time
4 import numpy as np
5 from PIL import ImageTk, Image
6
7 PhotoImage = ImageTk.PhotoImage
8 UNIT = 100 # 픽셀 수
9 HEIGHT = 7 # 그리드월드 세로
10 WIDTH = 7 # 그리드월드 가로
11 TRANSITION_PROB = 1
12 POSSIBLE_ACTIONS = [0, 1, 2, 3] # 좌, 우, 상, 하
13 ACTIONS = [(-1, 0), (1, 0), (0, -1), (0, 1)] # 좌표로 나타낸 행동
14 REWARDS = []
15
16
```

5\*5에서 7\*7로 변경

```
17 class GraphicDisplay(tk.Tk):
18     def __init__(self, agent):
19         super(GraphicDisplay, self).__init__()
20         self.title('Policy Iteration')
21         self.geometry('{0}x{1}'.format(HEIGHT * UNIT, HEIGHT * UNIT + 50))
22         self.texts = []
23         self.arrows = []
24         self.env = Env()
25         self.agent = agent
26         self.evaluation_count = 0
27         self.improvement_count = 0
28         self.is_moving = 0
29         (self.up, self.down, self.left, self.right), self.shapes = self.load_images()
30         self.canvas = self._build_canvas()
31         self.text_reward(3, 4, "R : 1.0")
32         self.text_reward(0, 4, "R : -1.0")
33         self.text_reward(1, 2, "R : -1.0")
34         self.text_reward(4, 2, "R : -1.0")
35         self.text_reward(3, 3, "R : -1.0")
36         self.text_reward(5, 3, "R : -1.0")
37         self.text_reward(2, 4, "R : -1.0")
38         self.text_reward(3, 5, "R : -1.0")
39
```

문제에서 주어진대로 보상을  
텍스트로 출력

```
72 # 캔버스에 이미지 추가
73 self.rectangle = canvas.create_image(50, 50, image=self.shapes[0])
74 canvas.create_image(450, 50, image=self.shapes[1])
75 canvas.create_image(250, 150, image=self.shapes[1])
76 canvas.create_image(450, 250, image=self.shapes[1])
77 canvas.create_image(350, 350, image=self.shapes[1])
78 canvas.create_image(550, 350, image=self.shapes[1])
79 canvas.create_image(250, 450, image=self.shapes[1])
80 canvas.create_image(350, 550, image=self.shapes[1])
81 canvas.create_image(450, 350, image=self.shapes[2])
82
83 canvas.pack()
84
85 return canvas
```

문제에서 주어진대로  
이미지 삽입

```

131 def rectangle_move(self, action, x, y):
132     base_action = np.array([0, 0])
133     location = self.find_rectangle()
134     self.render()
135     if action == 0 and location[0] > 0: # 상
136         base_action[1] -= UNIT
137         print(str(x-1)+' ', ' '+str(y))
138     elif action == 1 and location[0] < HEIGHT - 1: # 하
139         base_action[1] += UNIT
140         print(str(x + 1) + ' ', ' ' + str(y))
141     elif action == 2 and location[1] > 0: # 좌
142         base_action[0] -= UNIT
143         print(str(x) + ' ', ' ' + str(y-1))
144     elif action == 3 and location[1] < WIDTH - 1: # 우
145         base_action[0] += UNIT
146         print(str(x) + ' ', ' ' + str(y+1))
147     # move agent
148     self.canvas.move(self.rectangle, base_action[0], base_action[1])
149
150 def find_rectangle(self):
151     temp = self.canvas.coords(self.rectangle)
152     x = (temp[0] / 100) - 0.5
153     y = (temp[1] / 100) - 0.5
154     return int(y), int(x)
155

```

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

```

170 def draw_one_arrow(self, col, row, policy):
171     if col == 3 and row == 4:
172         return
173
174     if policy[0] > 0: # up
175         origin_x, origin_y = 50 + (UNIT * row), 10 + (UNIT * col)
176         self.arrows.append(self.canvas.create_image(origin_x, origin_y,
177                                                     image=self.up))
178     if policy[1] > 0: # down
179         origin_x, origin_y = 50 + (UNIT * row), 90 + (UNIT * col)
180         self.arrows.append(self.canvas.create_image(origin_x, origin_y,
181                                                     image=self.down))
182     if policy[2] > 0: # left
183         origin_x, origin_y = 10 + (UNIT * row), 50 + (UNIT * col)
184         self.arrows.append(self.canvas.create_image(origin_x, origin_y,
185                                                     image=self.left))
186     if policy[3] > 0: # right
187         origin_x, origin_y = 90 + (UNIT * row), 50 + (UNIT * col)
188         self.arrows.append(self.canvas.create_image(origin_x, origin_y,
189                                                     image=self.right))
190

```

col==2, row==2 → col==3, row==4

```


221 class Env:
222     def __init__(self):
223         self.transition_probability = TRANSITION_PROB
224         self.width = WIDTH
225         self.height = HEIGHT
226         self.reward = [[0] * WIDTH for _ in range(HEIGHT)]
227         self.possible_actions = POSSIBLE_ACTIONS
228         self.reward[3][4] = 1
229         self.reward[1][2] = -1
230         self.reward[0][4] = -1
231         self.reward[4][2] = -1
232         self.reward[3][3] = -1
233         self.reward[3][5] = -1
234         self.reward[2][4] = -1
235         self.reward[5][3] = -1
236         self.all_state = []
237
238     for x in range(WIDTH):
239         for y in range(HEIGHT):
240             state = [x, y]
241             self.all_state.append(state)
242

```


문제에서 주어진대로 보상을 입력

## policy\_iteration.py


```
6 class PolicyIteration:
7     def __init__(self, env):
8         # 환경에 대한 객체 선언
9         self.env = env
10        # 가치함수를 2차원 리스트로 초기화
11        self.value_table = [[0.0] * env.width for _ in range(env.height)]
12        # 상 하 좌 우 동일한 확률로 정책 초기화
13        self.policy_table = [[0.25, 0.25, 0.25, 0.25] * env.width
14                               for _ in range(env.height)]
15        # 마침 상태의 설정
16        self.policy_table[3][4] = []
17        # 감가율
18        self.discount_factor = 0.9
19
```

 self.policy\_table[2][2] → self.policy\_table[3][4]

```
20 def policy_evaluation(self):
21
22     # 다음 가치함수 초기화
23     next_value_table = [[0.00] * self.env.width
24                          for _ in range(self.env.height)]
25
26     # 모든 상태에 대해서 벨만 기대방정식을 계산
27     for state in self.env.get_all_states():
28         value = 0.0
29         # 마침 상태의 가치 함수 = 0
30         if state == [3, 4]:
31             next_value_table[state[0]][state[1]] = value
32             continue
33
34         # 벨만 기대 방정식
35         for action in self.env.possible_actions:
36             next_state = self.env.state_after_action(state, action)
37             reward = self.env.get_reward(state, action)
38             next_value = self.get_value(next_state)
39             value += (self.get_policy(state)[action] *
40                      (reward + self.discount_factor * next_value))
41
42         next_value_table[state[0]][state[1]] = round(value, 2)
43
44     self.value_table = next_value_table
45
```


 [2, 2] → [3, 4]

```

46 # 현재 가치 함수에 대해서 탐욕 정책 발전
47 def policy_improvement(self):
48     next_policy = self.policy_table
49     for state in self.env.get_all_states():
50         if state == [3, 4]:  [2, 2] → [3, 4]
51             continue
52         value = -99999
53         max_index = []
54         # 반환할 정책 초기화
55         result = [0.0, 0.0, 0.0, 0.0]
56
57         # 모든 행동에 대해서 [보상 + (감가율 * 다음 상태 가치함수)] 계산
58         for index, action in enumerate(self.env.possible_actions):
59             next_state = self.env.state_after_action(state, action)
60             reward = self.env.get_reward(state, action)
61             next_value = self.get_value(next_state)
62             temp = reward + self.discount_factor * next_value
63
64             # 받을 보상이 최대인 행동의 index(최대가 복수라면 모두)를 추출
65             if temp == value:
66                 max_index.append(index)
67             elif temp > value:
68                 value = temp
69                 max_index.clear()
70                 max_index.append(index)
71
72             # 행동의 확률 계산
73             prob = 1 / len(max_index)
74
75             for index in max_index:
76                 result[index] = prob
77
78         next_policy[state[0]][state[1]] = result
79
80     self.policy_table = next_policy
81

```

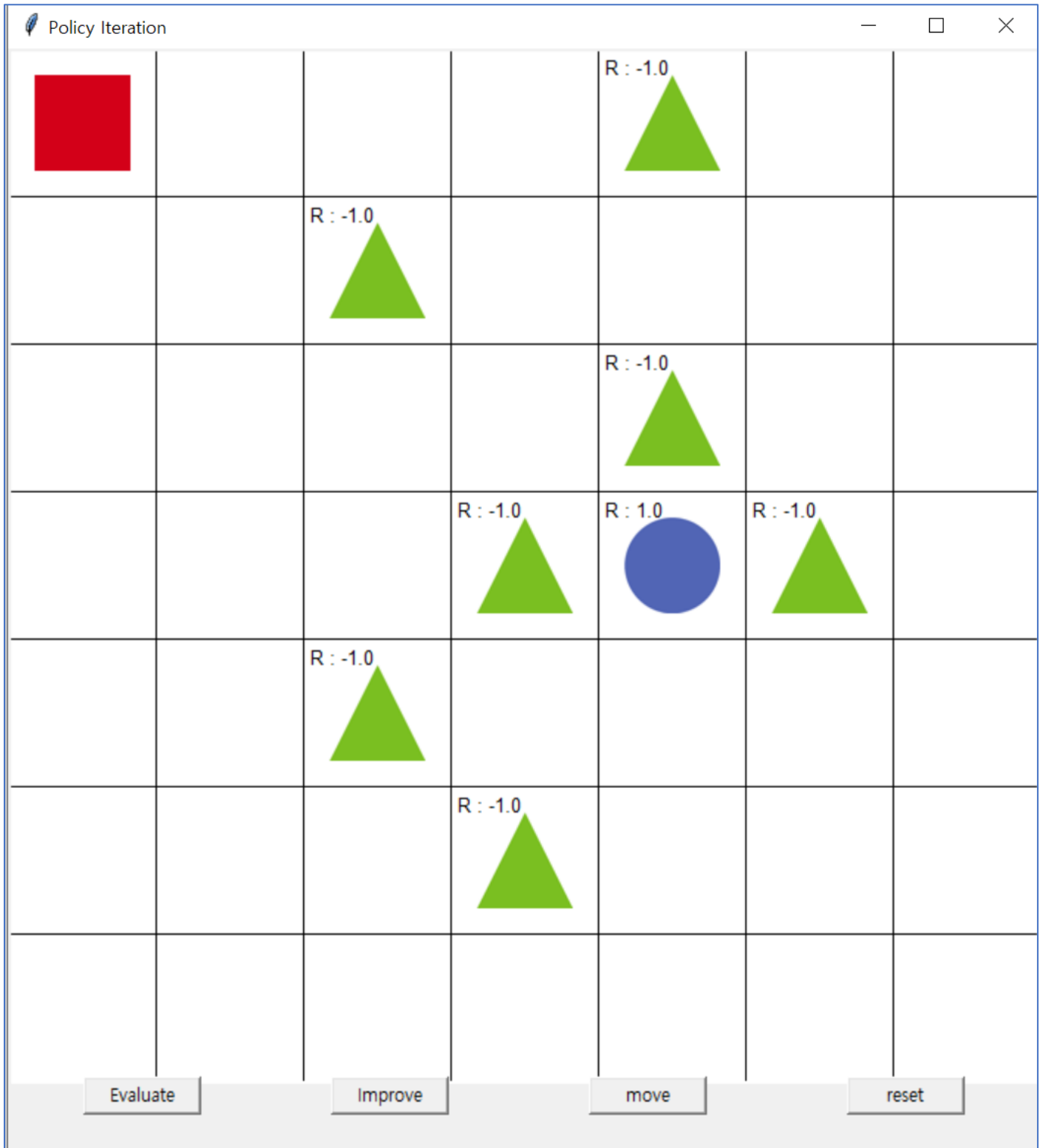
```

95 # 상태에 따른 정책 반환
96 def get_policy(self, state):
97     if state == [3, 4]:  [2, 2] → [3, 4]
98         return 0.0
99     return self.policy_table[state[0]][state[1]]
100

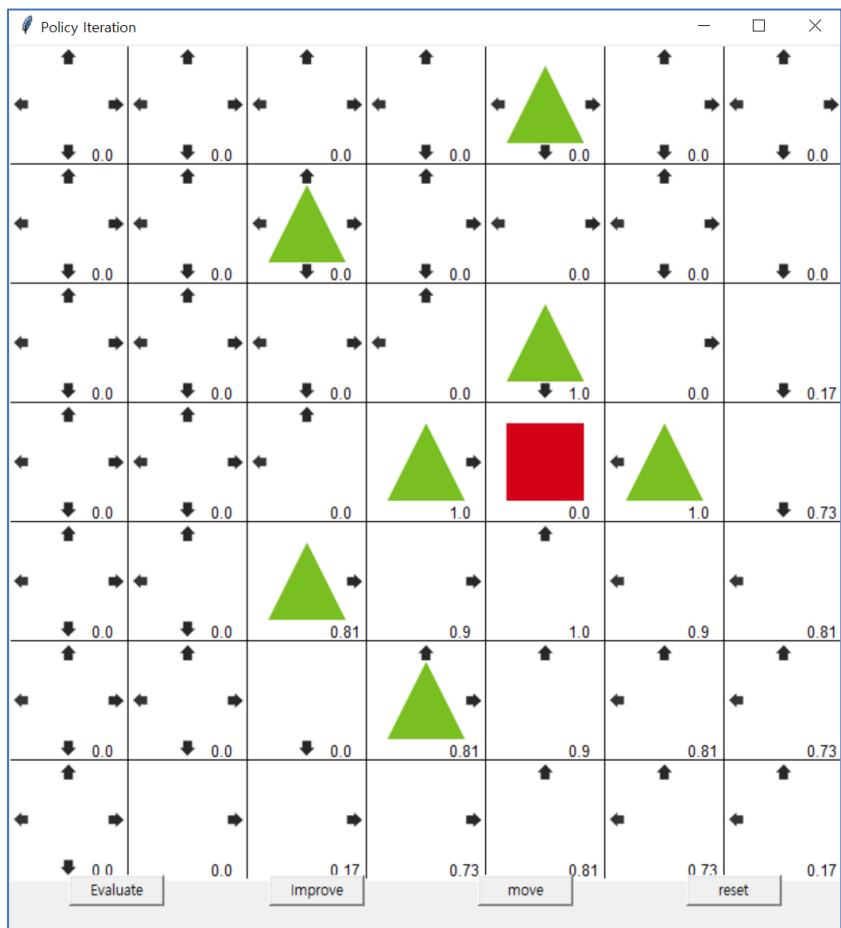
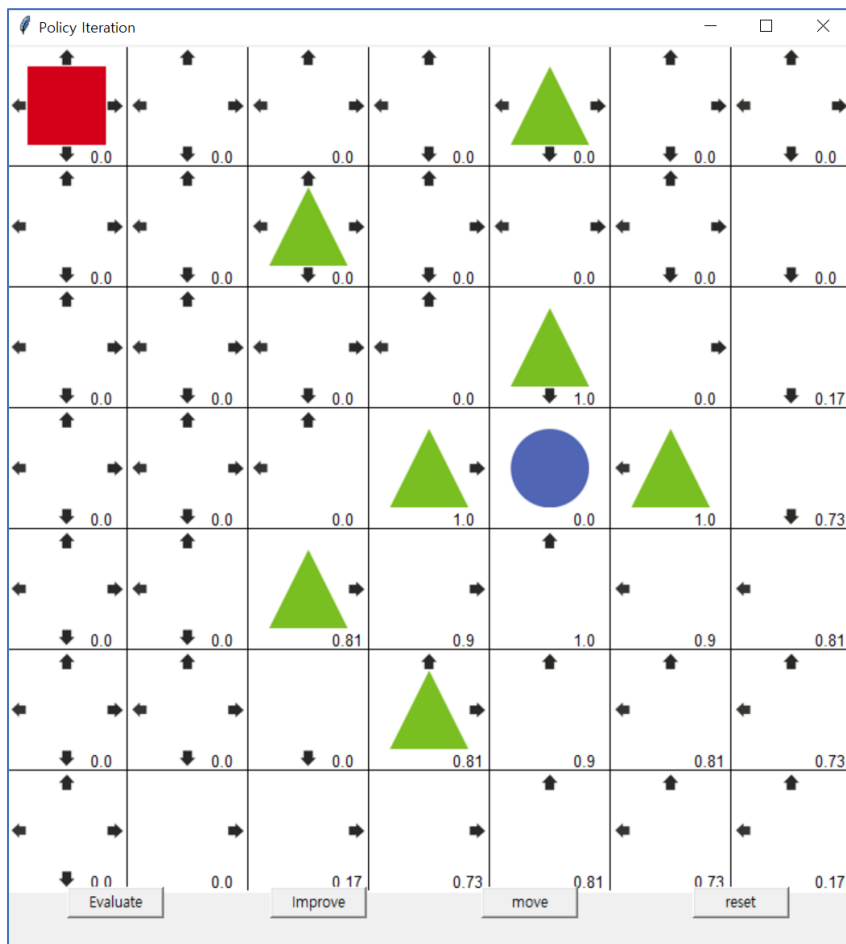
```

<실행>

Iteration회수 = 1 일 때 value table



## Iteration회수 = 5 일 때 value table

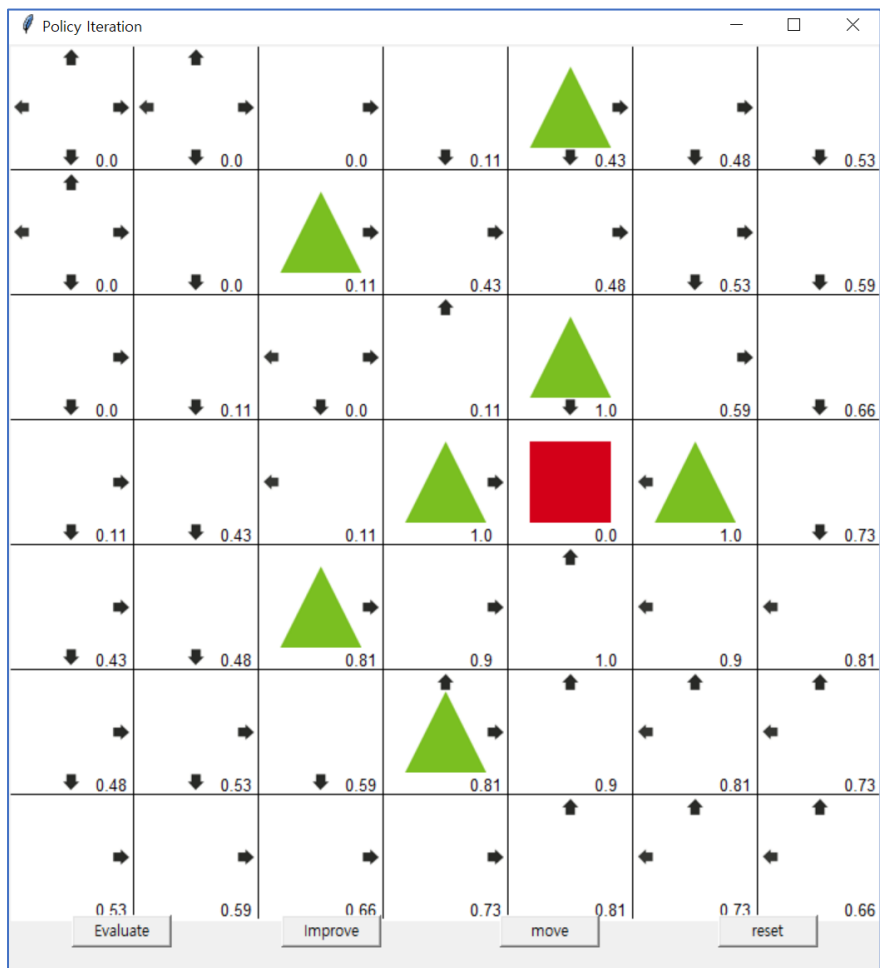
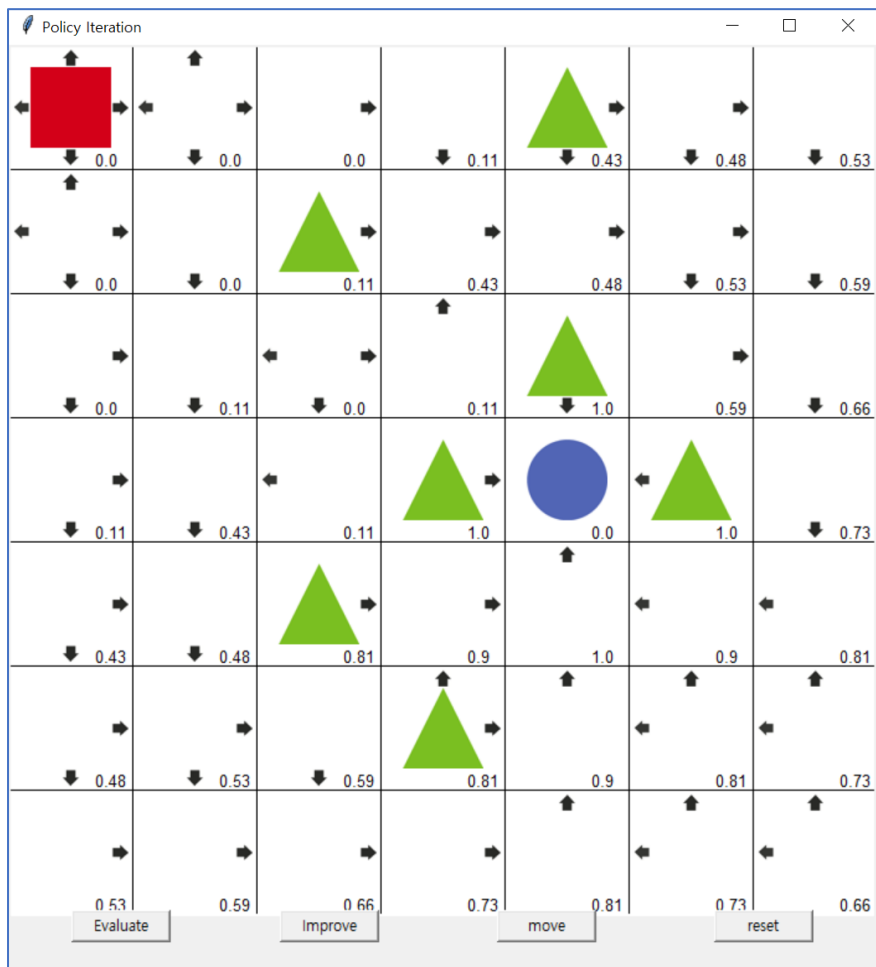




**Iteration회수 = 5 일 때 action list**

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

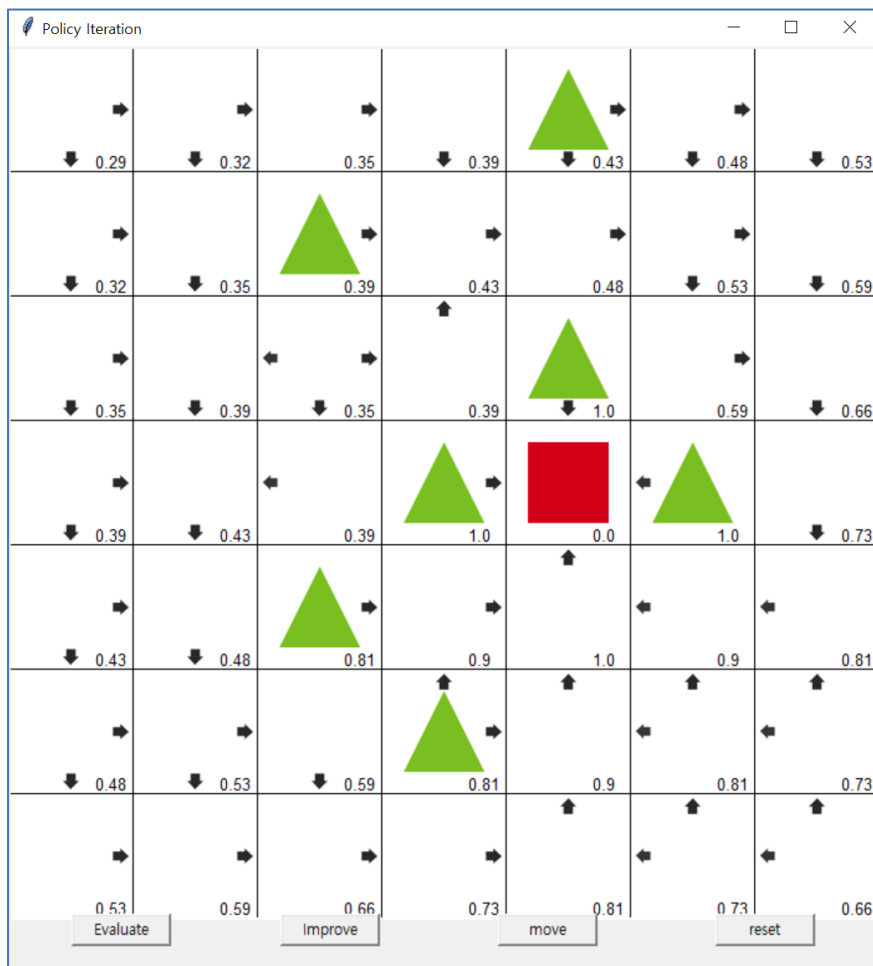
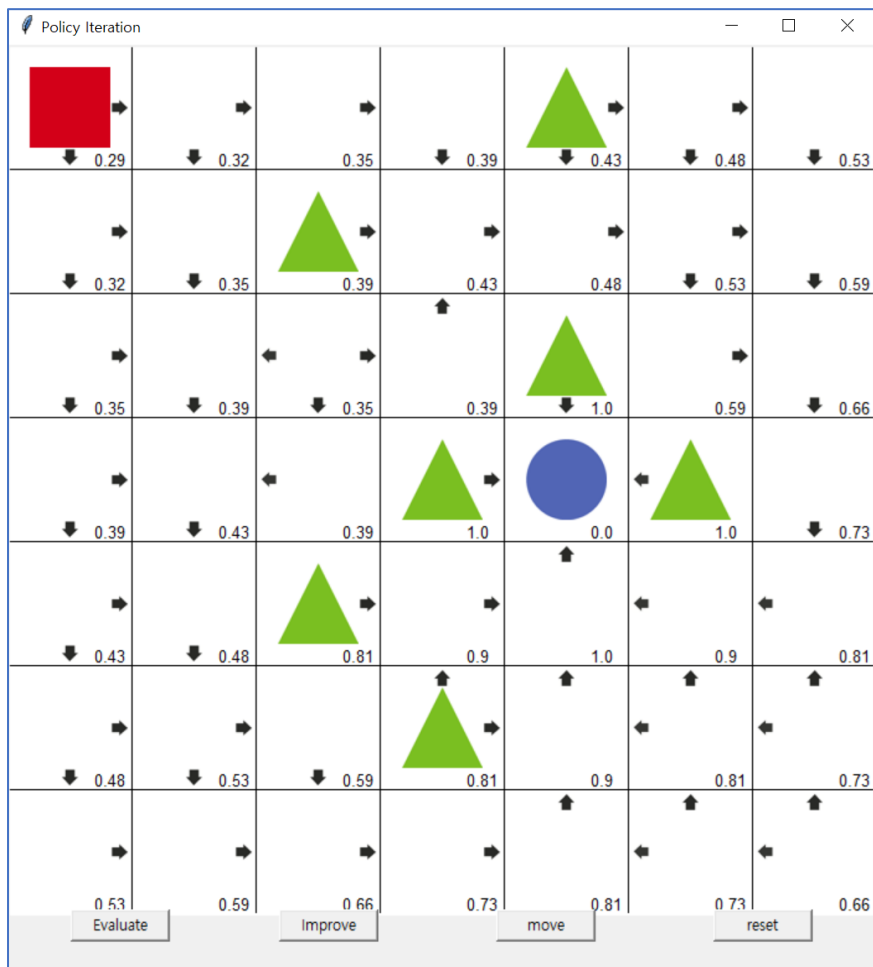
6, 4

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

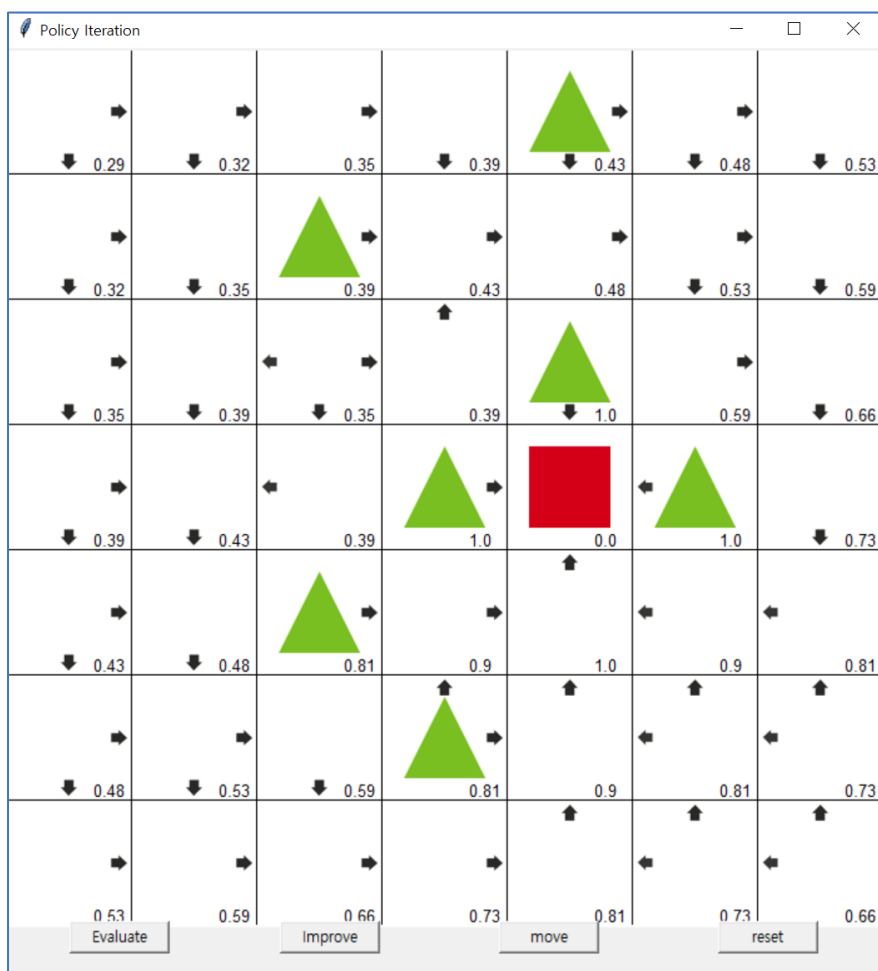
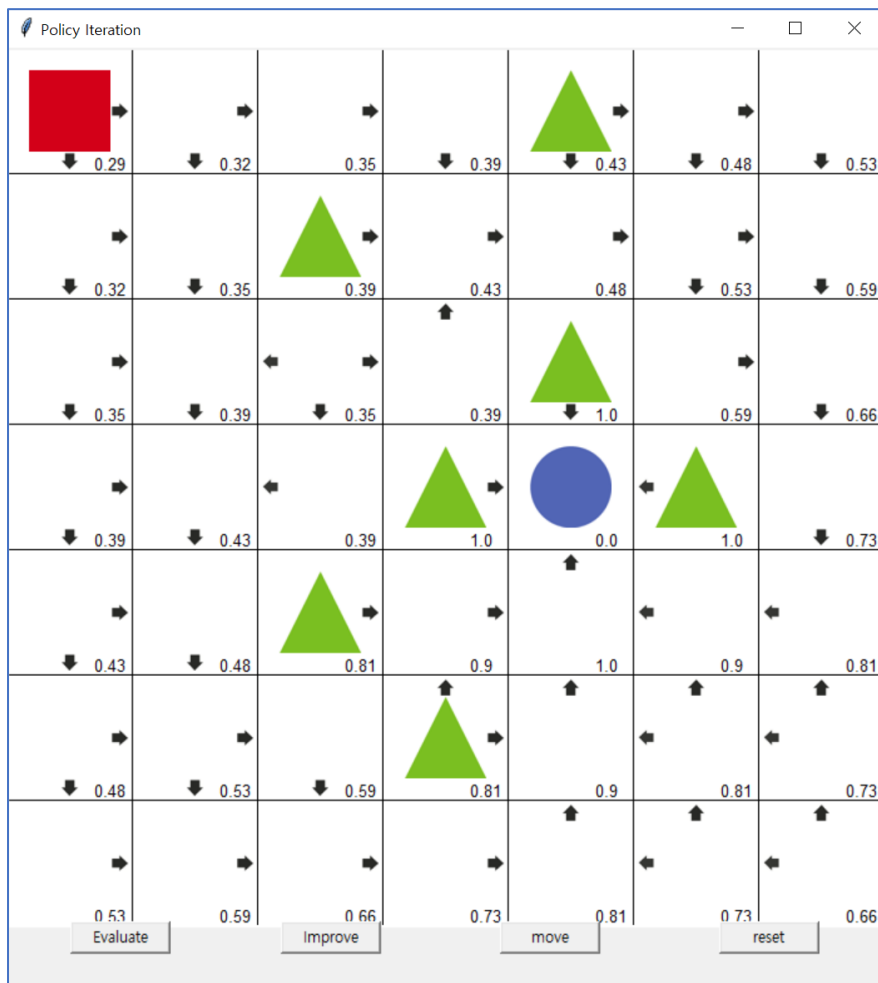
6, 4

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

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

5\*5에서 7\*7로 변경

```
17 class GraphicDisplay(tk.Tk):
18     def __init__(self, value_iteration):
19         super(GraphicDisplay, self).__init__()
20         self.title('Value Iteration')
21         self.geometry('{0}x{1}'.format(HEIGHT * UNIT, HEIGHT * UNIT + 50))
22         self.texts = []
23         self.arrows = []
24         self.env = Env()
25         self.agent = value_iteration
26         self.iteration_count = 0
27         self.improvement_count = 0
28         self.is_moving = 0
29         (self.up, self.down, self.left,
30          self.right), self.shapes = self.load_images()
31         self.canvas = self._build_canvas()
32         self.text_reward(3, 4, "R : 1.0")
33         self.text_reward(0, 4, "R : -1.0")
34         self.text_reward(1, 2, "R : -1.0")
35         self.text_reward(4, 2, "R : -1.0")
36         self.text_reward(3, 3, "R : -1.0")
37         self.text_reward(5, 3, "R : -1.0")
38         self.text_reward(2, 4, "R : -1.0")
39         self.text_reward(3, 5, "R : -1.0")
```

문제에서 주어진대로 보상을  
텍스트로 출력

```
77 # 캔버스에 이미지 추가
78 self.rectangle = canvas.create_image(50, 50, image=self.shapes[0])
79 canvas.create_image(450, 50, image=self.shapes[1])
80 canvas.create_image(250, 150, image=self.shapes[1])
81 canvas.create_image(450, 250, image=self.shapes[1])
82 canvas.create_image(350, 350, image=self.shapes[1])
83 canvas.create_image(550, 350, image=self.shapes[1])
84 canvas.create_image(250, 450, image=self.shapes[1])
85 canvas.create_image(350, 550, image=self.shapes[1])
86 canvas.create_image(450, 350, image=self.shapes[2])
87
88 canvas.pack()
89
90 return canvas
```

문제에서 주어진대로  
이미지 삽입



```

145 def rectangle_move(self, action, x, y):
146     base_action = np.array([0, 0])
147     location = self.find_rectangle()
148     self.render()
149     if action == 0 and location[0] > 0: # up
150         base_action[1] -= UNIT
151         print(str(x-1) + ', ' + str(y))
152     elif action == 1 and location[0] < HEIGHT - 1: # down
153         base_action[1] += UNIT
154         print(str(x + 1) + ', ' + str(y))
155     elif action == 2 and location[1] > 0: # left
156         base_action[0] -= UNIT
157         print(str(x) + ', ' + str(y-1))
158     elif action == 3 and location[1] < WIDTH - 1: # right
159         base_action[0] += UNIT
160         print(str(x) + ', ' + str(y+1))
161
162     self.canvas.move(self.rectangle, base_action[0],
163                     base_action[1]) # move agent
164

```

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

```

185 def draw_one_arrow(self, col, row, action):
186     if col == 3 and row == 4:
187         return
188     if action == 0: # up
189         origin_x, origin_y = 50 + (UNIT * row), 10 + (UNIT * col)
190         self.arrows.append(self.canvas.create_image(origin_x, origin_y,
191                                                     image=self.up))
192     elif action == 1: # down
193         origin_x, origin_y = 50 + (UNIT * row), 90 + (UNIT * col)
194         self.arrows.append(self.canvas.create_image(origin_x, origin_y,
195                                                     image=self.down))
196     elif action == 3: # right
197         origin_x, origin_y = 90 + (UNIT * row), 50 + (UNIT * col)
198         self.arrows.append(self.canvas.create_image(origin_x, origin_y,
199                                                     image=self.right))
200     elif action == 2: # left
201         origin_x, origin_y = 10 + (UNIT * row), 50 + (UNIT * col)
202         self.arrows.append(self.canvas.create_image(origin_x, origin_y,
203                                                     image=self.left))
204

```

col==2, row==2 → col==3, row==4

```

237 class Env:
238     def __init__(self):
239         self.transition_probability = TRANSITION_PROB
240         self.width = WIDTH # Width of Grid World
241         self.height = HEIGHT # Height of GridWorld
242         self.reward = [[0] * WIDTH for _ in range(HEIGHT)]
243         self.possible_actions = POSSIBLE_ACTIONS
244         self.reward[3][4] = 1
245         self.reward[1][2] = -1
246         self.reward[0][4] = -1
247         self.reward[4][2] = -1
248         self.reward[3][3] = -1
249         self.reward[3][5] = -1
250         self.reward[2][4] = -1
251         self.reward[5][3] = -1
252         self.all_state = []
253

```

문제에서 주어진대로 보상을 입력

## value\_iteration.py

```
15 def value_iteration(self):
16     next_value_table = [[0.0] * self.env.width for _ in
17                         range(self.env.height)]
18     for state in self.env.get_all_states():
19         if state == [3, 4]:
20             next_value_table[state[0]][state[1]] = 0.0
21             continue
22     # 가치 함수를 위한 빈 리스트
23     value_list = []
24
25     # 가능한 모든 행동에 대해 계산
26     for action in self.env.possible_actions:
27         next_state = self.env.state_after_action(state, action)
28         reward = self.env.get_reward(state, action)
29         next_value = self.get_value(next_state)
30         value_list.append((reward + self.discount_factor * next_value))
31     # 최댓값을 다음 가치 함수로 대입
32     next_value_table[state[0]][state[1]] = round(max(value_list), 2)
33     self.value_table = next_value_table
34
```

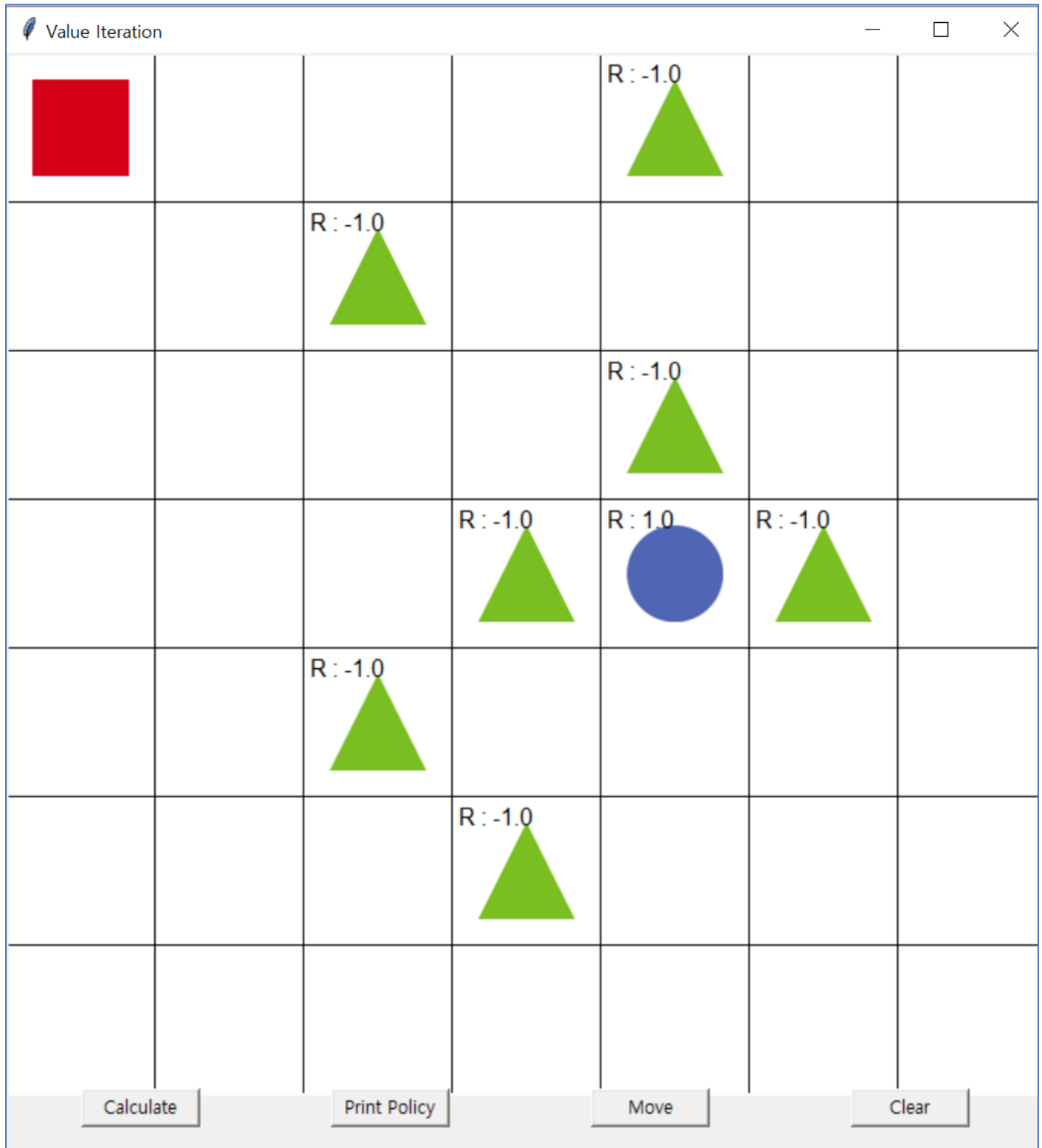
[2, 2] → [3, 4]

```
35 # 현재 가치 함수로부터 행동을 반환
36 def get_action(self, state):
37     action_list = []
38     max_value = -99999
39
40     if state == [3, 4]:
41         return []
42
43     # 모든 행동에 대해 큐함수 (보상 + (감가율 * 다음 상태 가치함수))를 계산
44     # 최대 큐 함수를 가진 행동(복수일 경우 여러 개)을 반환
45     for action in self.env.possible_actions:
46
47         next_state = self.env.state_after_action(state, action)
48         reward = self.env.get_reward(state, action)
49         next_value = self.get_value(next_state)
50         value = (reward + self.discount_factor * next_value)
51
52         if value > max_value:
53             action_list.clear()
54             action_list.append(action)
55             max_value = value
56         elif value == max_value:
57             action_list.append(action)
58
59     return action_list
60
```

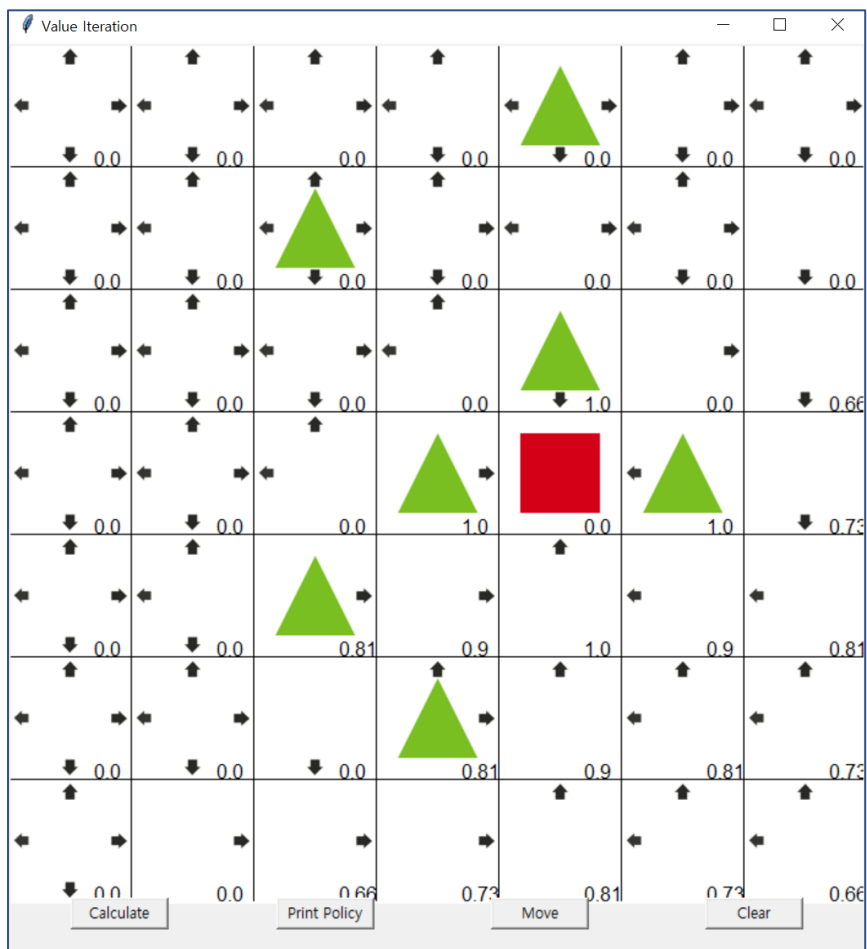
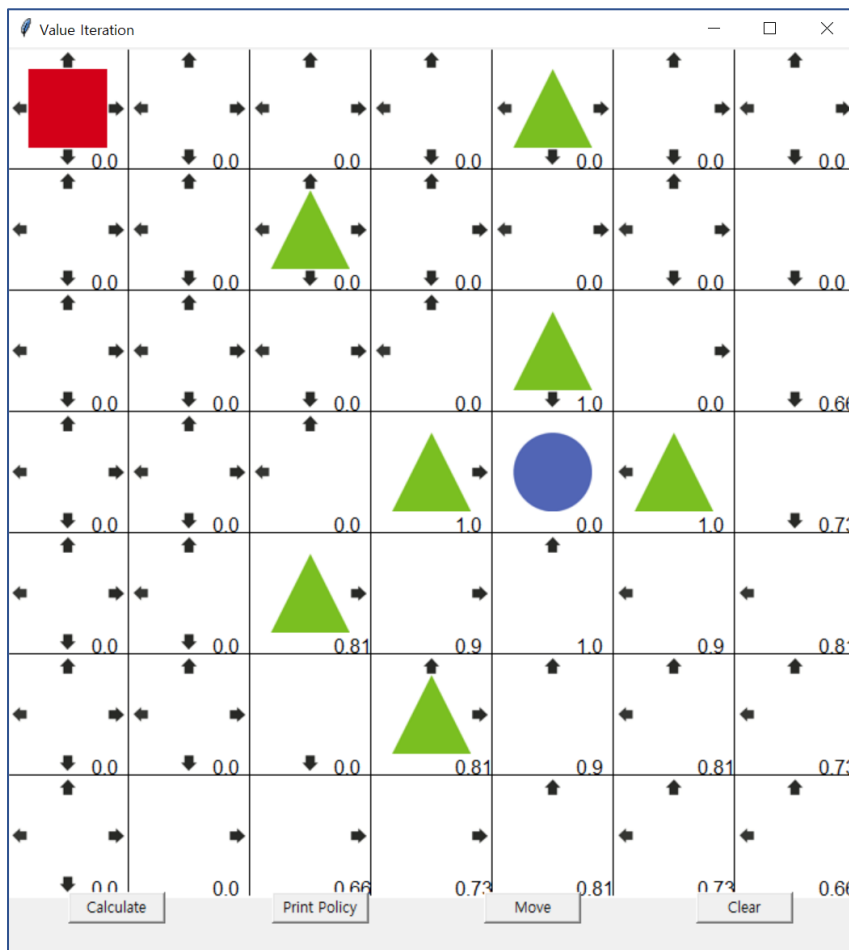
[2, 2] → [3, 4]

<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

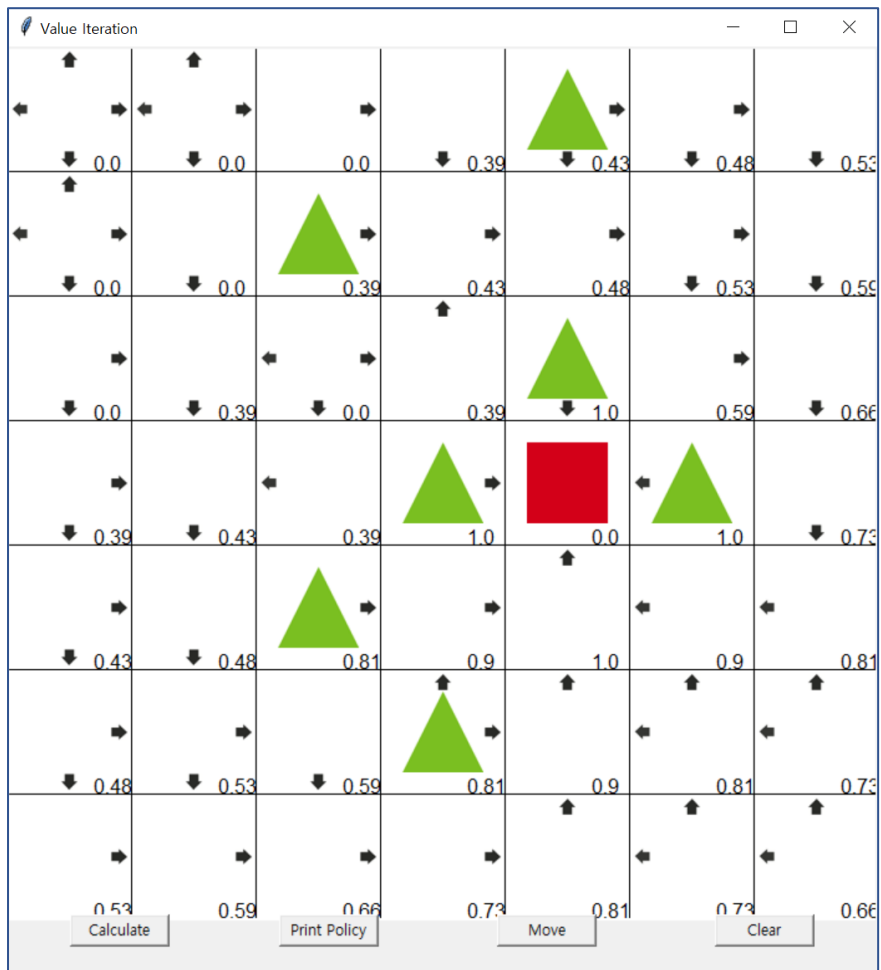
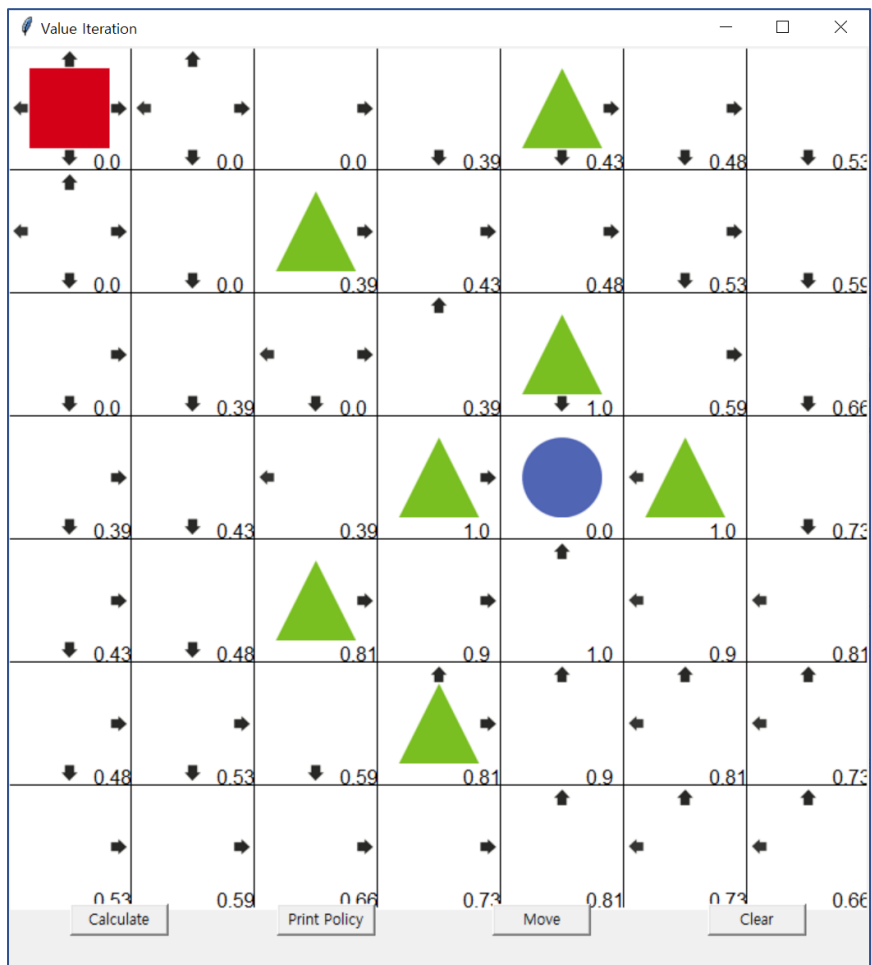
4, 6

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

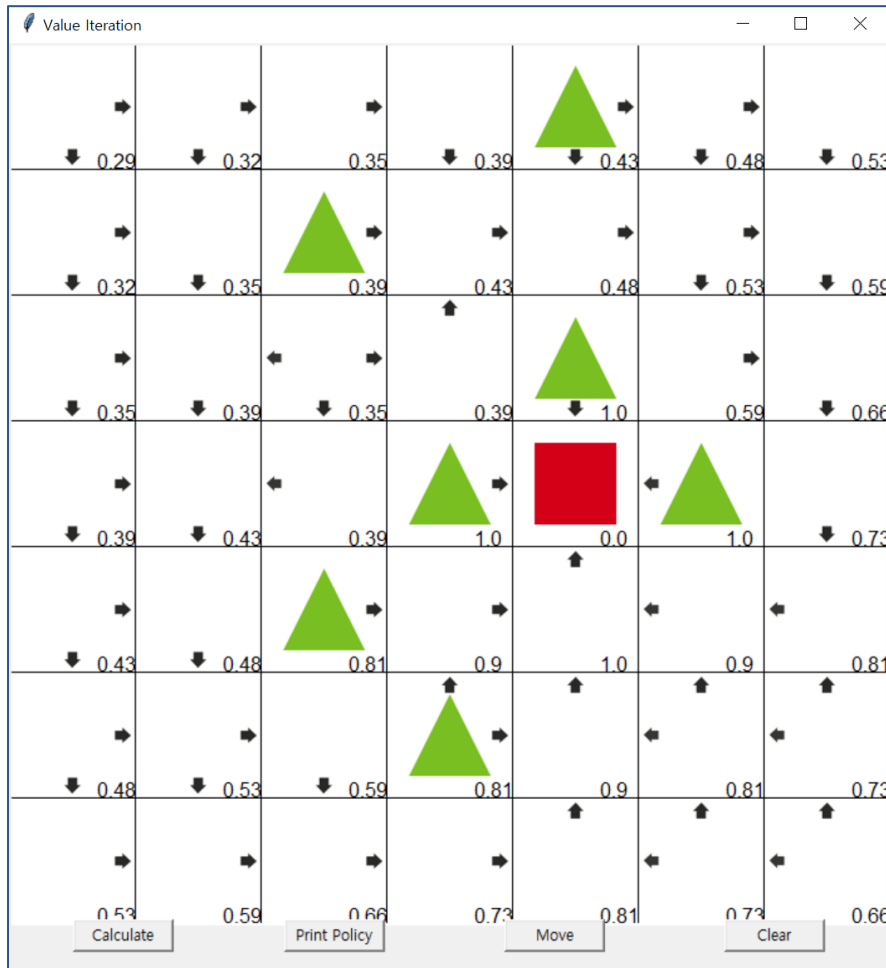
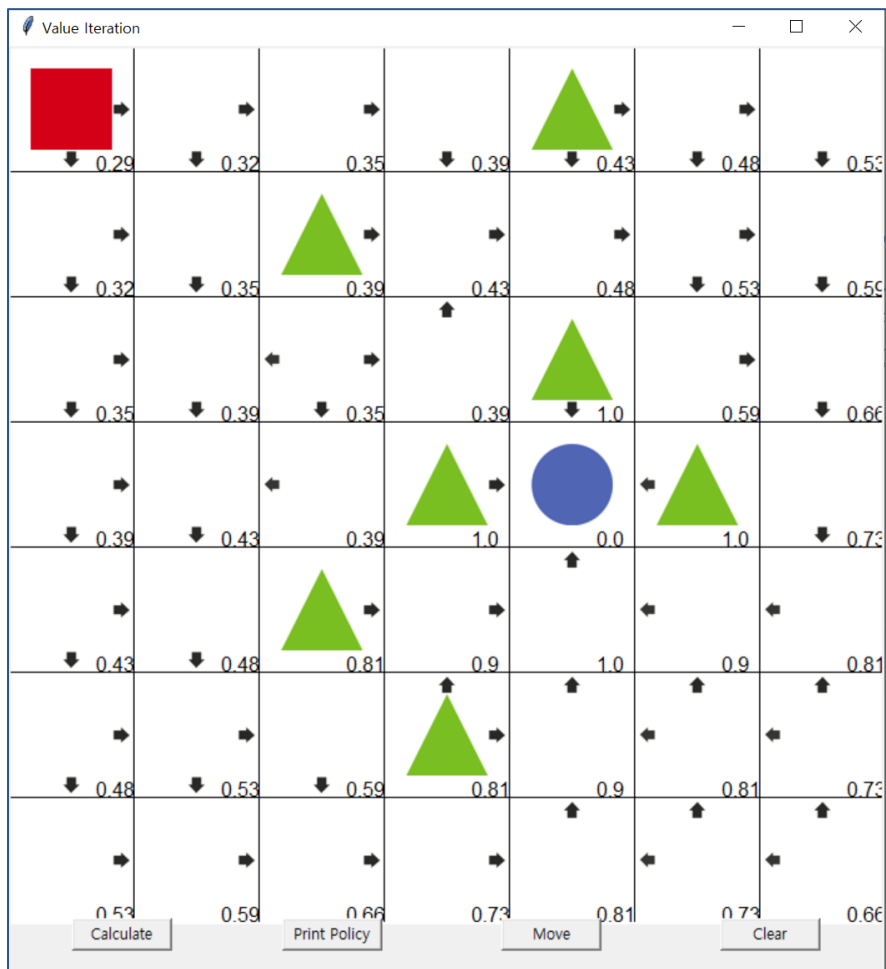
6, 4

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

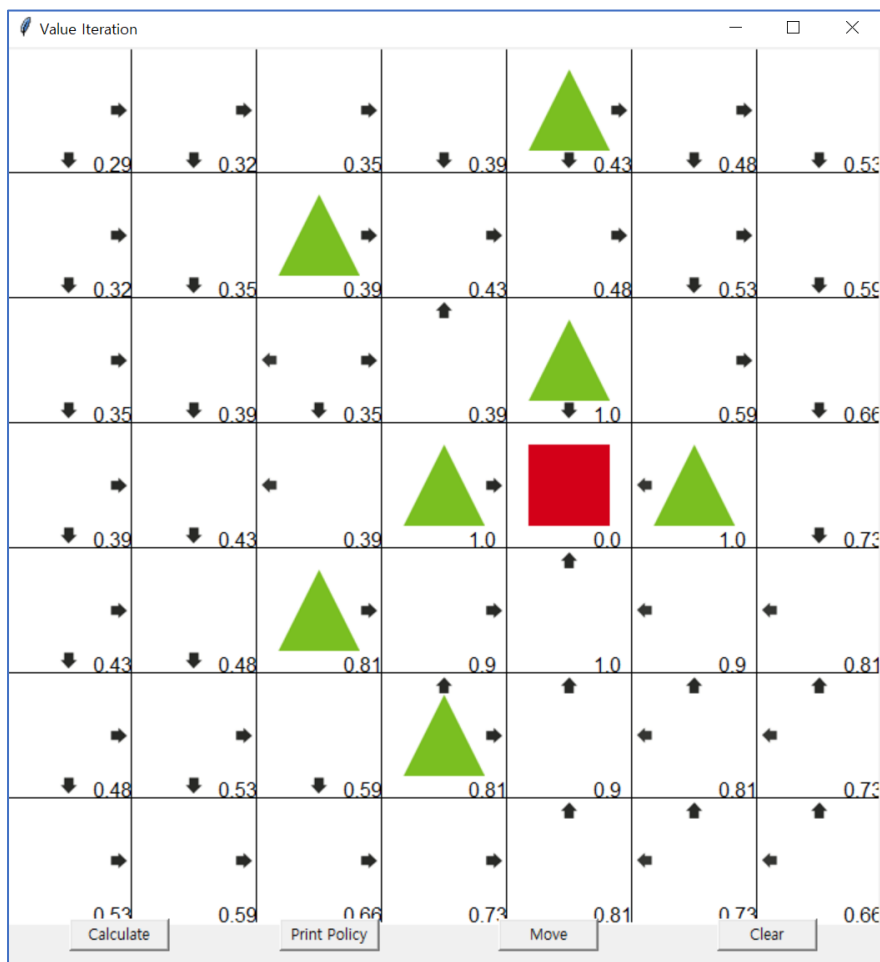
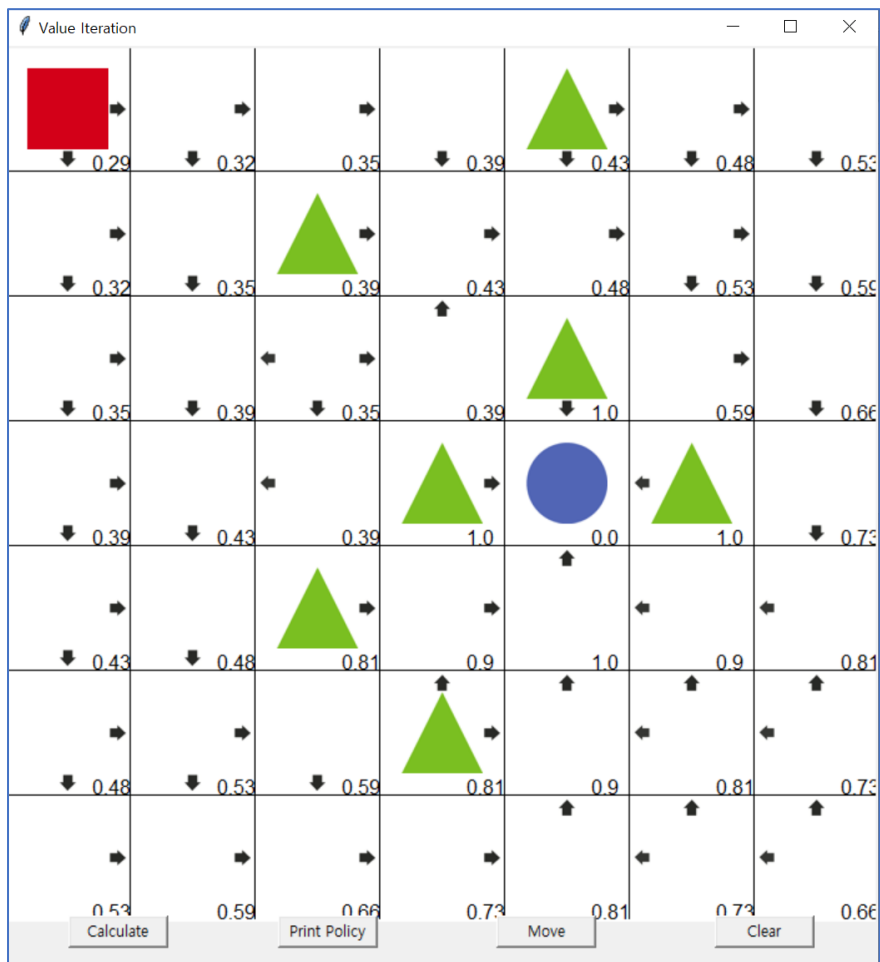
6, 4

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