CS470 Assignment 3

20180185 김해찬

Markov Decision Process [50 pts]

1. [15 pts] print out the transition probabilities to all the next states given

- a current state [3, 1] and a selected action “Down”,

```
[41] ###################################################################
     #####################   PLACE YOUR CODE HERE   ####################
     ###                                                             ###

     p = env.transition_model(int(env.twod_to_serial(np.array([3,1]))) ,2)
     ###################################################################
     print(p.reshape(env.grid_map_shape[::-1]).T)
```

```
[[0.     0.     0.     0.     0.     0.     0.     0.     0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.    ]
 [0.     0.0125 0.     0.     0.     0.     0.     0.     0.     0.    ]
 [0.0125 0.0125 0.0125 0.     0.     0.     0.     0.     0.     0.    ]
 [0.     0.95   0.     0.     0.     0.     0.     0.     0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.    ]]
```

- a current state [0, 6] and a selected action “Right”,

```
###################################################################
#####################   PLACE YOUR CODE HERE   ####################
###                                                             ###

p = env.transition_model(int(env.twod_to_serial(np.array([0,6]))) ,4)

###################################################################
print(p.reshape(env.grid_map_shape[::-1]).T)
```

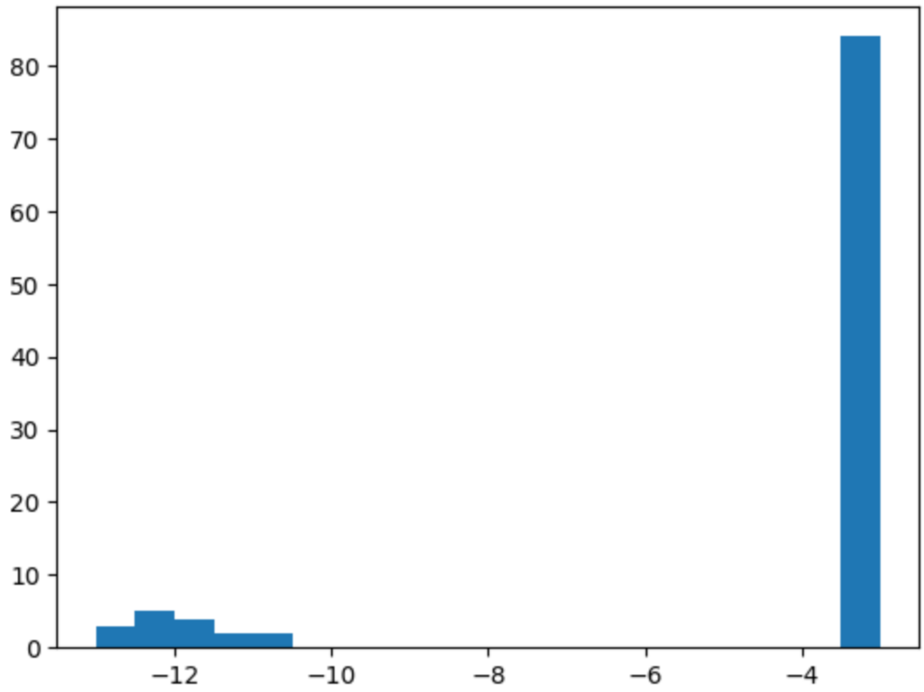
```
[[0.     0.     0.     0.     0.     0.0125 0.025  0.95   0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.0125 0.     0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.    ]]
```

- a current state [3, 5] and a selected action “Right”,

```
#####################################################################
#####################    PLACE YOUR CODE HERE    ####################
###                                                              ###

p = env.transition_model(int(env.twod_to_serial(np.array([3,5]))) ,4)

#####################################################################
print(p.reshape(env.grid_map_shape[::-1]).T)
```

```
[[0.     0.     0.     0.     0.     0.     0.     0.     0.     0.     ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.     ]
 [0.     0.     0.     0.     0.     0.0125 0.     0.     0.     0.     ]
 [0.     0.     0.     0.     0.0125 0.9625 0.     0.     0.     0.     ]
 [0.     0.     0.     0.     0.     0.0125 0.     0.     0.     0.     ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.     ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.     ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.     ]]
```

2. [5 pts] plot the resulting histogram of returns produced by a dummy policy in the IPython
notebook for 100 episodes,



3. [25 pts] attach your implemented code from transition model(), compute reward(), step(),
and is done() functions.

    - transition model()

```
####################################################################
####################   PLACE YOUR CODE HERE   ####################
###                                                              ###
### Instruction                                                  ###
### -----------                                                  ###
### - You have to fill out the "probs" variable following the   ###
###   the assignment-problem 1's transition model rules          ###
### - Hint: you have to find the next state based on the        ###
###         available action at the current state               ###
### - Hint: you need to use serial_to_twod & twod_to_serial      ###
###         functions                                            ###
###
### Example
### --------
### After some initializations...
### for action in availabe actions:
###     find a next state
###     fill out probs[next state]
###

# initialization or handling exceptions
action_probs = [(1-self.epsilon) if i == action else self.epsilon/4 for i in range(self.action_space.n)]

for new_action, prob in enumerate(action_probs):
    next_state_twod = self.serial_to_twod(state) + np.array(action_pos_dict[new_action])
    if next_state_twod[0] < 0 or next_state_twod[0] >= self.grid_map_shape[0] or next_state_twod[1] < 0 or next_state_twod[1] >= self.grid_map_shape[1]:
        next_state = state
    else:
        next_state = int(self.twod_to_serial(next_state_twod))
        if next_state in self.obstacles:
            next_state = state
    probs[next_state] += prob
###                                                              ###
###                                                              ###
####################################################################
####################################################################
```

- compute reward()

```
####################################################################
####################   PLACE YOUR CODE HERE   ####################
###                                                              ###
### Instruction                                                  ###
### -----------                                                  ###
### - fill out the reward variable                               ###
###                                                              ###

if next_state == self.terminal_state:
  reward = 5 - 0.1
elif next_state in self.traps:
  reward = -10 - 0.1
else:
  reward = -0.1

###                                                              ###
###                                                              ###
####################################################################
####################################################################
```

- done()

```
###################################################################
#####################   PLACE YOUR CODE HERE   ####################
###                                                            ###
### Instruction                                                ###
### -----------                                                ###
### - fill out the "done" variable                             ###

if next_state == self.terminal_state:
  done = True
elif next_state in self.traps:
  done = True
else:
  done = False

###                                                            ###
###################################################################
###################################################################
```

- step()

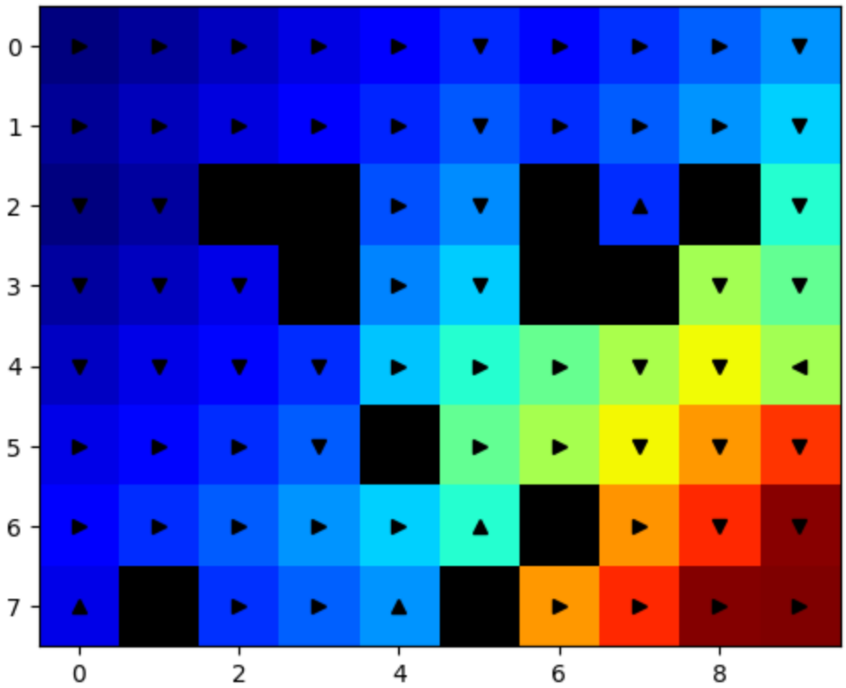
```
###################################################################
#####################   PLACE YOUR CODE HERE   ####################
###                                                            ###
### Instruction                                                ###
### -----------                                                ###
### - sample the next state considring the transition model    ###
### - then, compute "reward" and "done"                        ###
###   next_state = ...                                         ###
###
### Example                                                    ###
### -----------                                                ###
### next state = ?
### p = ?
### self.observation = ?
### reward = self.compute_reward(?, action, self.observation)  ###
### done = self.is_done(?, action, self.observation)           ###

state = self.observation
next_state = int(np.random.choice(self.observation_space.n, 1, p=probs))
p = probs[next_state]

self.observation = next_state
reward = self.compute_reward(state, action, self.observation)
done = self.is_done(state, action, self.observation)

###                                                            ###
###################################################################
###################################################################
```

2. 1 Value Iteration (VI) [30 pts]

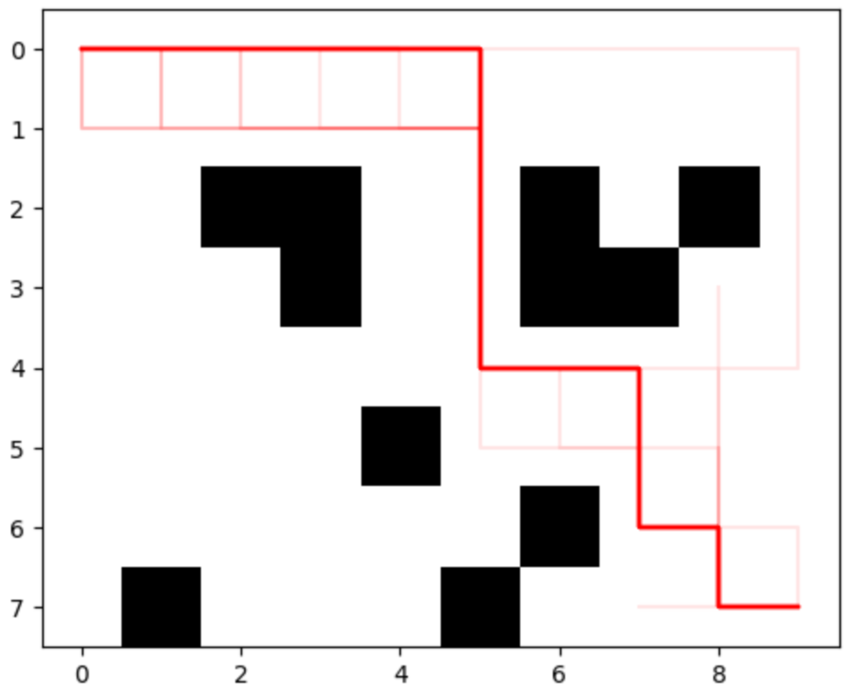
- write down the state values of the first 8 states of the gridworld environment1,

```
[ 7.92810146  8.7902002   8.06292376  9.10013195 10.27998951 11.59774961
 13.0168294  11.54909515]
```

- overlay the best action at each state based on the state-action values,



- plot the distribution of trajectories produced by the trained policy for 100 episodes, and



- attach your implemented code from value iteration() and get action() functions on your

  report.

  - value iteration()

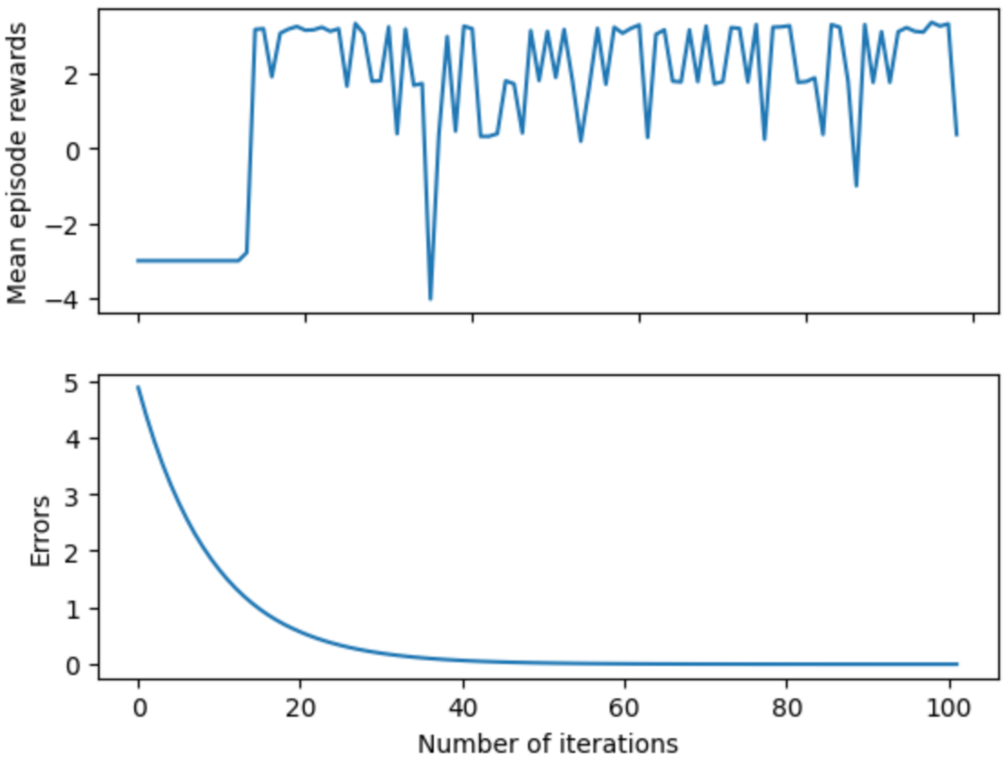
```
################################################################
#####################   PLACE YOUR CODE HERE   ################
###                                                          ###
### Instruction                                              ###
### -----------                                              ###
### Fill out self.V by using the VI algorithm                ###
###
### Example                                                  ###
### -----------                                              ###
### Initializations
### For a in available actions
###   For s' in available next states
###     Compute reward(s,a,s'), etc
###     Compute action value
### Fill out "self.V" at the current state using the action values

for action in range(env.action_space.n):
    for next_state in range(env.observation_space.n):
        Q[action] += env.transition_model(state, action)[next_state] * (env.compute_reward(state, action, next_state) + self.discount_factor * self.V[next_state])
max_error = max(max_error, np.abs(np.max(Q) - self.V[state]))
self.V[state] = np.max(Q)

###                                                          ###
################################################################
################################################################
```

  - get action()

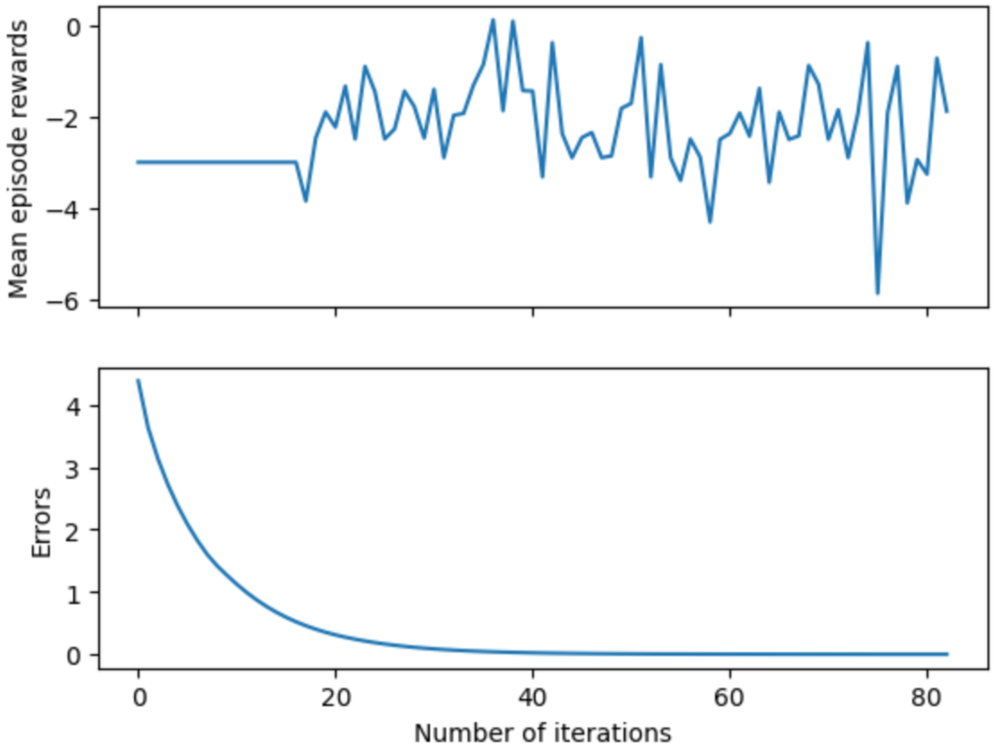
```python
###################################################################
#####################   PLACE YOUR CODE HERE   ####################
###                                                             ###
### Instruction                                                 ###
### -----------                                                 ###
### Find the best "action" that maximize Q value                ###
###                                                             ###
### Example                                                     ###
### -----------                                                 ###
### Compute Q values when you apply each action                 ###
### For a in available actions
###   For s' in next states
###     Q[a] = ?
###
### action = ?

for action in range(env.action_space.n):
    for next_state in range(env.observation_space.n):
        Q[action] += env.transition_model(state, action)[next_state] * (env.compute_reward(state, action, next_state) + self.discount_factor * self.V[next_state])

highest_value = np.max(Q)
available_actions = [i for i, value in enumerate(Q) if value == highest_value]
action = int(np.random.choice(available_actions, 1))

###                                                             ###
###################################################################
###################################################################
```

2.2 Comparison under different transition models [20 pts]

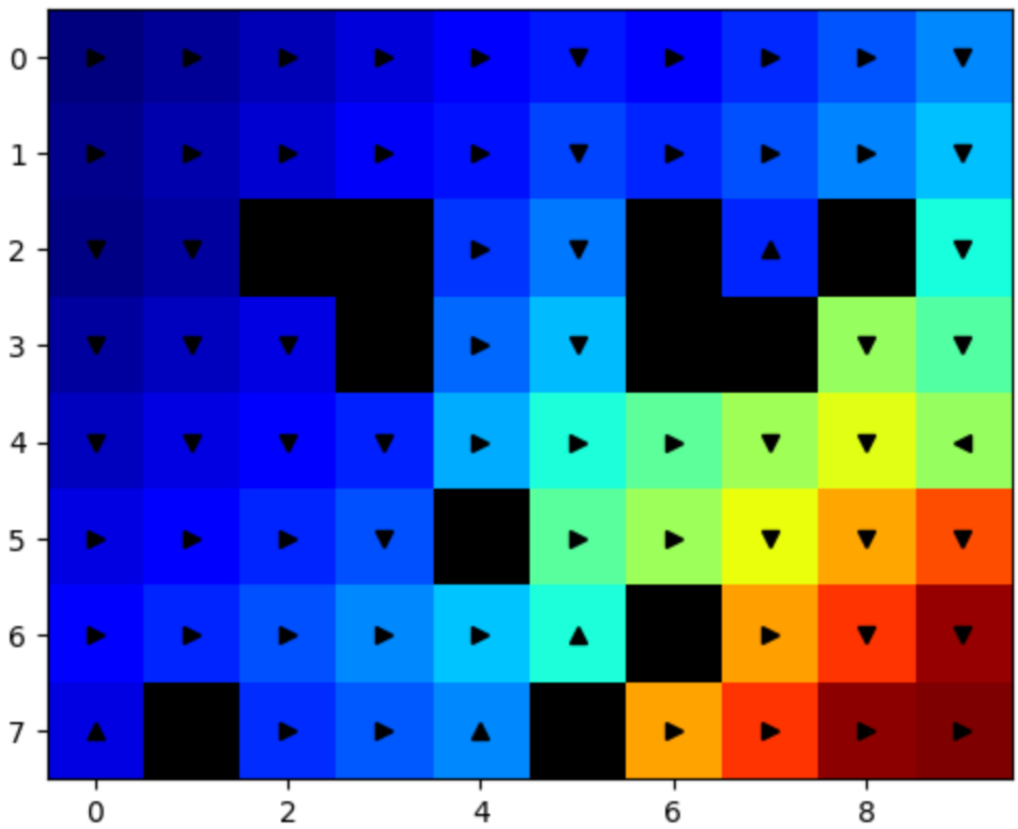
- plot the expected returns per ε value with respect to the number of iterations until convergence (two graphs or one unified graph),
    - ε = 0.1



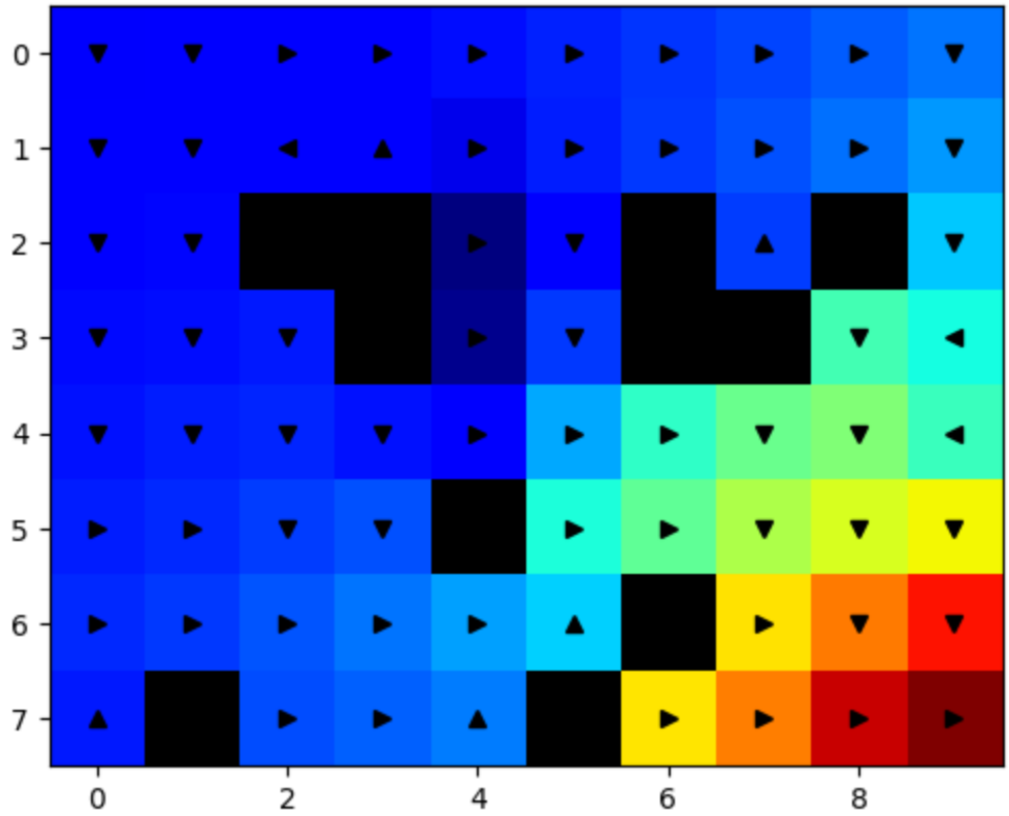
    - ε = 0.4



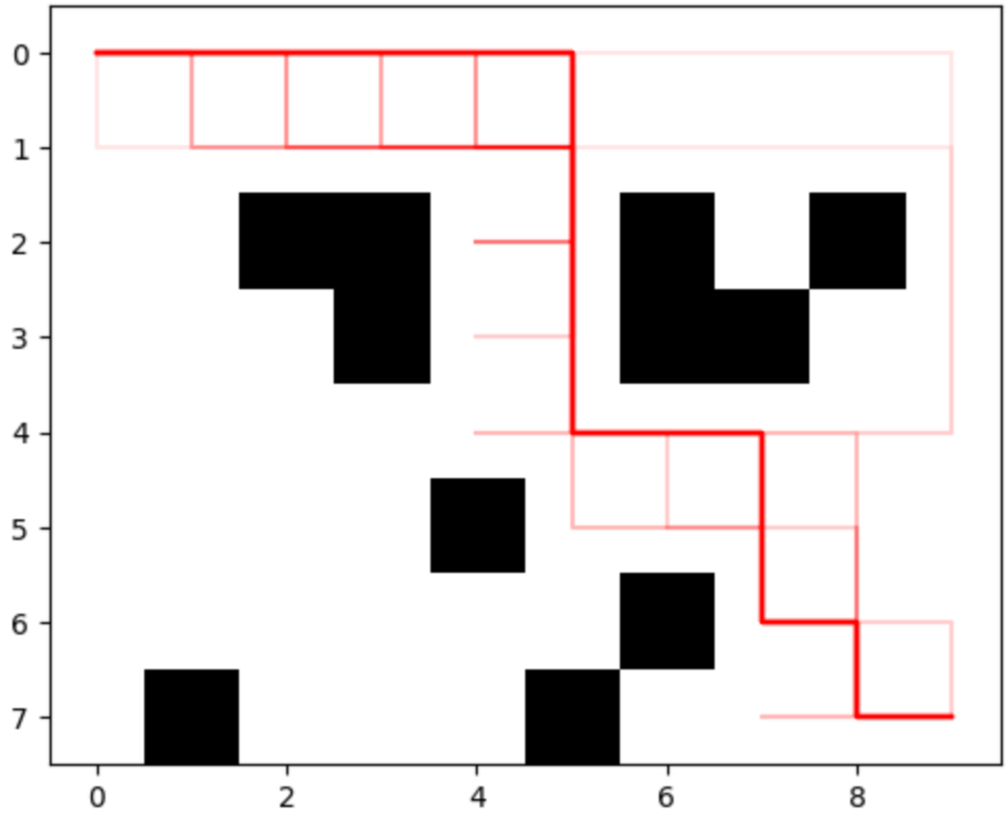
- overlaying the best actions at each state per ε value (two visualizations),
    - ε = 0.1



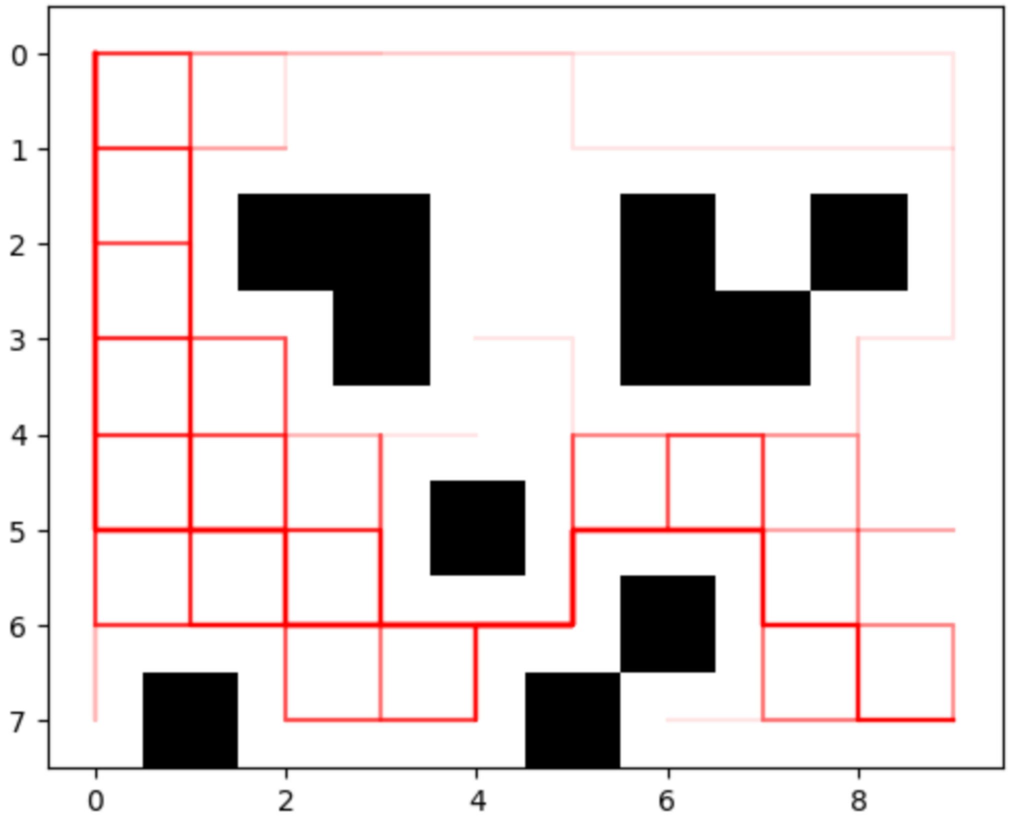
- ε = 0.4



- plot the distribution of trajectories produced by the trained policy for 100 episodes per ε value (two visualizations), and
    - ε = 0.1



- $\varepsilon = 0.4$



- compare/analyze the effect of different ε based on the above results.

  As shown in the two plots of trajectories for each epsilon value, there are some unnessesary paths where epslion is 0.4. This is because eplison represents the level of uncertainty that the agent do exactly chosen action, so a lager eplison may cause the agent to take unexpected actions with high probability. In addition, for high epsilon values, it is not guaranteed to reach the maximum reward consistently as the Value iteration progresses.