Exercise 2

Amr Alkhashab - 905833
ELEC-E8125 - Reinforcement Learning

September 21, 2020

1 Question 1

The agent is the sailor donated with a brown "boat" and environment is the gridworld containing the light blue, rocks, dark blue and the target harbour.

2 Task 1

The Value iteration Function Code:

```
def value_interation(env, gamma, theta):
    V = np.zeros((env.w, env.h))
    for i in range(100):
        for w in range(env.w):
            for h in range(env.h):
                v = V[w, h]
                V[w, h] = bellman_optimality_update(env, V, w,
                   h, gamma)

     #Observe Value function for each iteration
        #env.clear_text()
        #env.draw_values(V)
        #env.render()
        #sleep(1)


$Greedy policy estimation
    pi = np.zeros((env.w, env.h))
    for w in range(env.w):
        for h in range(env.h):
            pi[w, h] = q_greedify_policy(env, V, w, h, gamma)
    return V, pi
```

The Optimality bellman equation used to update state value function for each state

```
def bellman_optimality_update(env, V, w, h, gamma):
    min = float('-inf')
    Vnext = 0
    actions = [env.UP, env.DOWN, env.RIGHT, env.LEFT]

#for each action calculate value and choose the max at the end
    for a in actions:
        transition = env.transitions[w, h, a]
        Vaction = 0

#loop over all state and ignore those with state = None (
   termination state) assigned to Zero.
        if (transition[0].state != None):
            for t in range(len(transition)):
                Vnext = V[transition[t].state[0], transition[t
                   ].state[1]]
                Vaction = Vaction + transition[t].prob * (
                   transition[t].reward + gamma * Vnext)
                #print(Vaction)
            if Vaction > min:
                min = Vaction
        else:
            min = 0
    return min
```

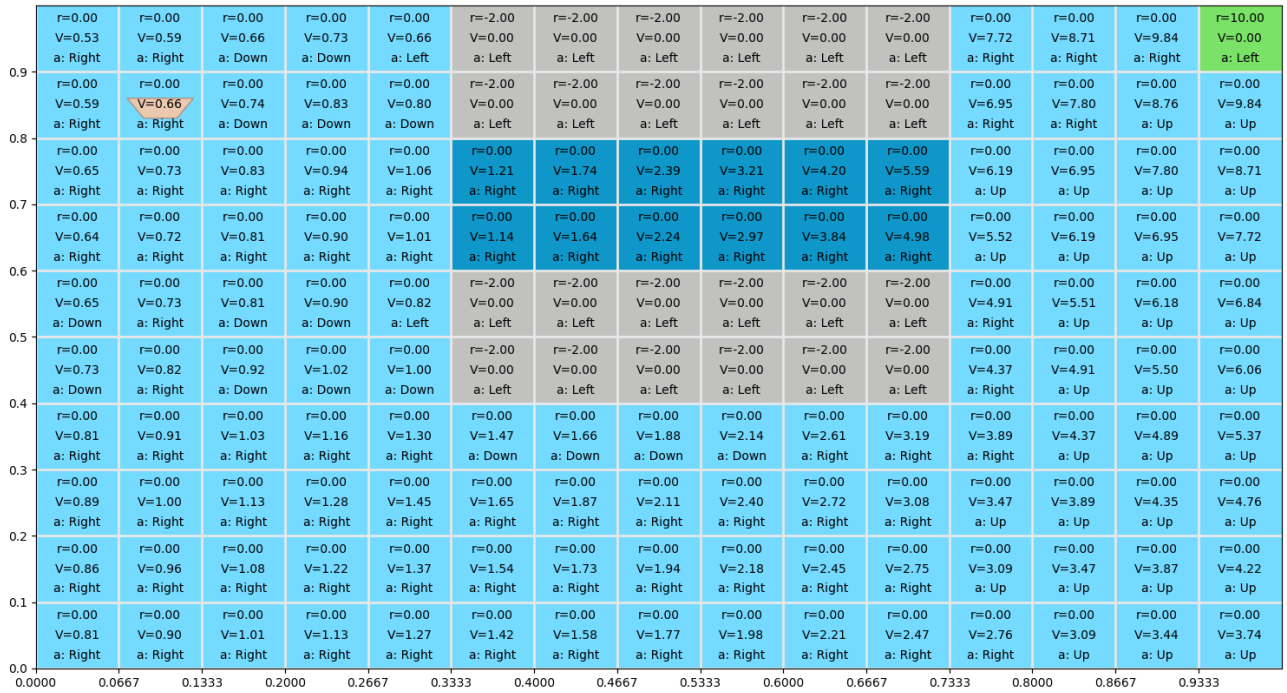

Figure 1: The optimal state value and policy

3 Question 2

The state values of harbour and rock ar Zeros, because they are terminating states. The terminating states mark the end of the episode, there is no transition from it to any other state, thus the return equals to zero.

4 Task2

The optimal policy is computed from the same value function and state values shown in Task1. The optimal policy can be seen from Figure. 1.// The greedy policy for any state function has been computed using the following code. The task has been repeated several times and the sailor manage to reach the goal most of the times.

```
def q_greedify_policy(env, V, w, h, gamma):
    min = float('-inf')
    argmax = 0
    actions = [env.UP, env.DOWN, env.RIGHT, env.LEFT]

#it is very similar to bellman equation, the only difference it
    return argmax, instead of the maximum state value.
    for a in actions:
        transition = env.transitions[w, h, a]
        Vaction = 0
        if(transition[0].state != None):
            for t in range(len(transition)):
                    Vnext = V[transition[t].state[0],
                       transition[t].state[1]]
                    Vaction = Vaction + transition[t].prob * (
                       transition[t].reward + gamma * Vnext)
            if Vaction > min:
                min = Vaction
                argmax = a
        else:
            argamx = 0
    return argmax

------------------------------
#The agent was observed using the optimal policy by using
action = int(policy[state[0], state[1]])
```

5 Question 3

The sailor as we can seen from the observation and Figure. 1, choose the dangerous path between the rocks. If we choose reward of -10, the sailer change his path taking the longer path, trying to get far awy from the rock by 1 grid, as shown in Figure. 2.

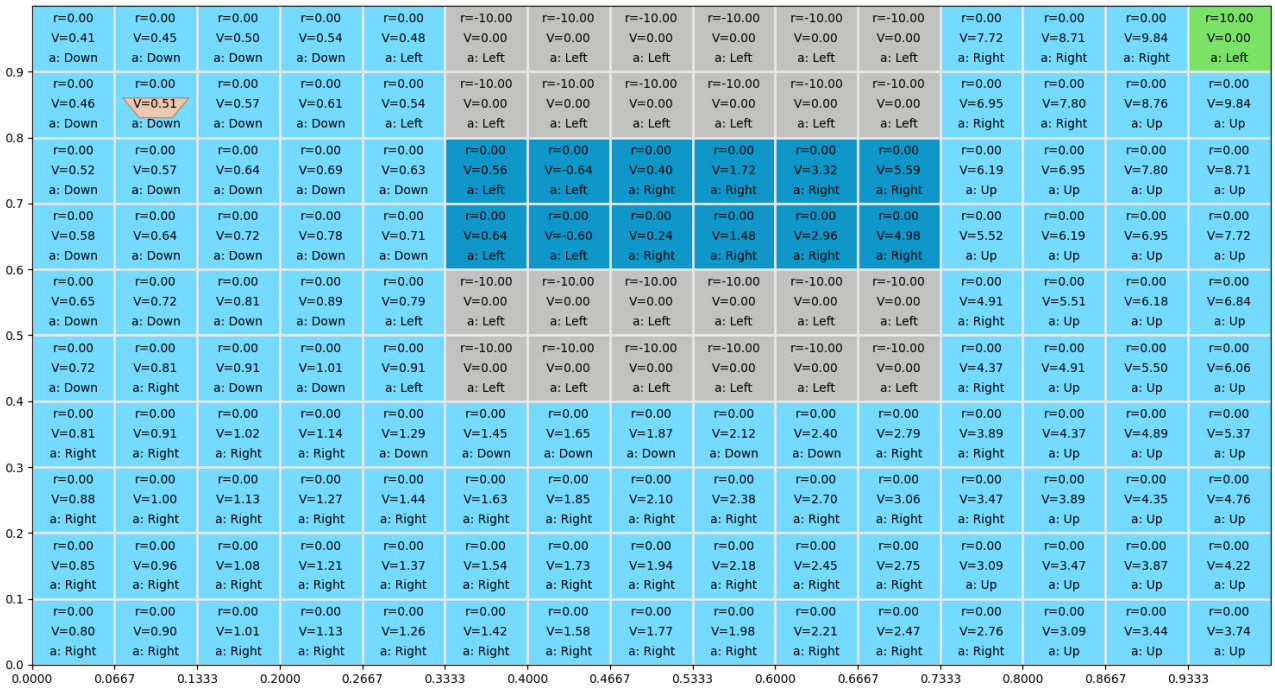

Figure 2: Behavior under -10 rewards

6 Question 4

Running the algorithm for smaller amount like 10 will results in a value function and policy very far from the optimal. Thus, this means that both will not converge. The policy will converge faster, in a sense that we assume that the value function only converge when the maximum change is lower than a certain threshold as it approach the true optimal value asymptotically, however for a policy it converge when all action for all state are equal to the optimal policy which happens before reaching the threshold. Thus, a policy can achieve the optimal policy before a value iteration converge. This matter can be observe at iterations equals to 25, Figure .3, where we can see that the policy at state has already converge, while the value policy is not, it can be observed from states (0,0) - at the bottom left, by comparing with Figure. 1.

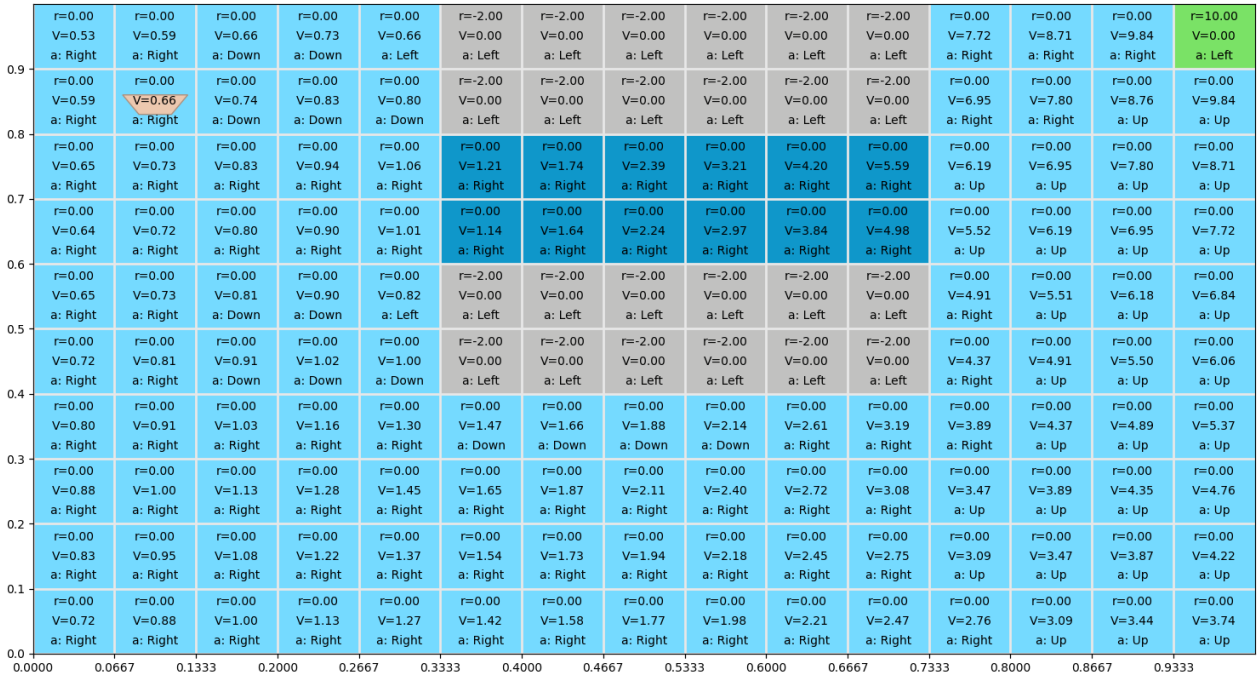

Figure 3: Value state and policy at iteration (25)

7 Task 3

To run until convergance, we modified the value function as shown, the comments for other comment has been remove here to show only what is essential.

```
def value_interation(env, gamma, theta):
    V = np.zeros((env.w, env.h))

#theta is the epsilon given my the main code

    while True:
        delta = 0
        for w in range(env.w):
            for h in range(env.h):
                v = V[w, h]
                V[w, h] = bellman_optimality_update(env, V, w,
                   h, gamma)

#break when delta achieved less than theta for all states
                delta = max(delta, abs(v - V[w, h]))
        if delta < theta:
            break

        #env.clear_text()
        #env.draw_values(V)
        #env.render()
        #sleep(1)

    pi = np.zeros((env.w, env.h))
    for w in range(env.w):
        for h in range(env.h):
            pi[w, h] = q_greedify_policy(env, V, w, h, gamma)
    return V, pi
```

8 Task 4

Program modifications

```
1
2
3  #store all return for the intial states
4   cum_returns = np.zeros(1000, dtype=float)
5
6      # TODO: Run multiple episodes and compute the discounted
          returns (Task 4)
7  #run for 1000 times each times
8      for i in range(1000):
9          state = env.reset()
10         cum_return = 0
11         power = 0
12         done = False
13         while not done:
14             # Select a random action
15             # TODO: Use the policy to take the optimal action (
                  Task 2)
16             #action = int(np.random.random()*4)
17
18             action = int(policy[state[0], state[1]])
19             # Step the environment
20             state, reward, done, _ = env.step(action)
21             cum_return = cum_return + pow(gamma, power)*reward
22             power = power + 1
23             # Render and sleep
24            # env.render()
25            #sleep(0.5)
26         #print(cum_return)
27         cum_returns[i] = cum_return
28
29 #compute mean and std by using numpy
30     print("The mean is equal", np.mean(cum_returns))
31     print("STD", np.std(cum_returns))
```

The mean and std for starting state (1,8):
The mean is equal 0.6417659452721378
STD 1.350261367200206

9 Question 5

The discound return which is discribed for single episode, summation of discounted reward under optimal policy, Equation. (3.8) [1].While, value iteration Equation (4.10) [1] can be described as the discounted return multiplied by probability of state transition. Which means the value function is the estimate of the return for many episodes, in other word, an estimate of the avearage return, and we can see the mean calculate form Task4 is nearly equals to value function for state (1,8).

10 Question 6

No, since the robot is exploring an unknown environment, this means we dont have the state transitions probabilities, in that context the equations of value iteration cannot be used since it requires that probability. Monte carlo and TD algorithms resolve issues of unknow environment. In value iteration, we assume the environment is somehow known to the agent, however it is unrealistic to assume state transition for an environment the agent doesnot know, and for a robot discovering the environment it is impossible to have such assumption due to the complexity of the task and large number of states.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.