Exercise 2

Gengcong Yan - 1009903
ELEC-E8125 - Reinforcement Learning

October 10, 2021

1 Question 1

- Agent: The sailor
- Environment: Water area, Harbour, Narrow passage, Rocks, Possible wind.

2 Task 1

```
1  def values_iterations(env,gamma=1.0,itNum=100):
2
3      value_est, policy = np.zeros((env.w, env.h)), np.zeros((env
           .w, env.h))
4
5      for it in range(itNum):
6          # iterate every state in 2D
7          env.clear_text()
8          for w in range(env.w):
9              for h in range(env.h):
10                 Q_value=[]
11                 for tran in env.transitions[w,h]:
12                     # print("tran:")
13                     # print(tran)
14                     A_value=0
15                     for next_state,reward,done,prob in tran:
16                         # consider all situation under
                               different prob
17                         A_value+=prob*(reward+(gamma*value_est[
                               next_state] if not done else 0))
18                     Q_value.append(A_value)
19                 # print(it,Q_value)
20                 value_est[w,h]=np.max(Q_value)
21                 policy[w,h]=np.argmax(Q_value)
22     return value_est,policy
```

The render environment of Task 1 is shown in Fig.1.

3 Question 2

The reward value of harbour and rock states are 10 and -2, respectively. But the state values of them are both 0. Because in our default setup of environment, the episode terminates as soon as the sailor arrives harbour and rocks. There are no next state for them to transfer,their values can't get updated then always be zero.

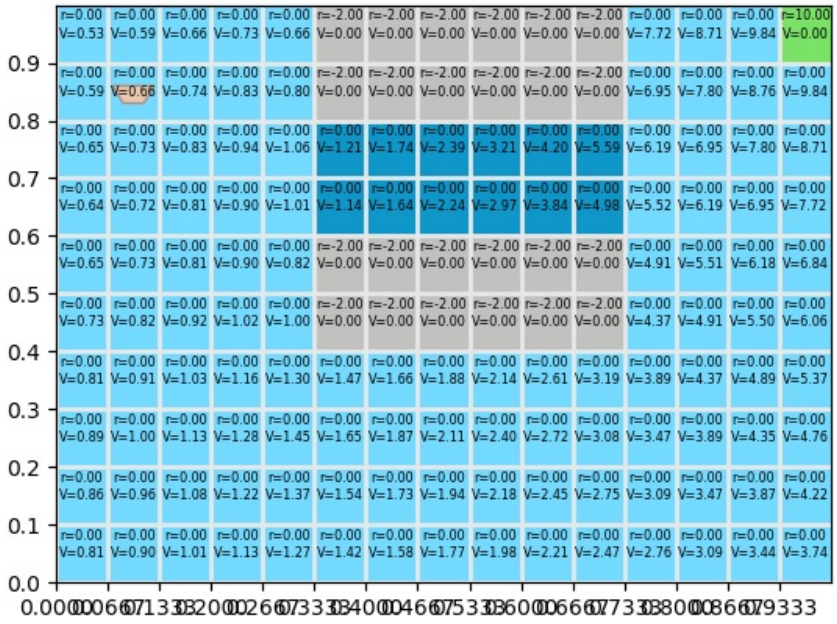


Figure 1: Values in environment after 100 iterations

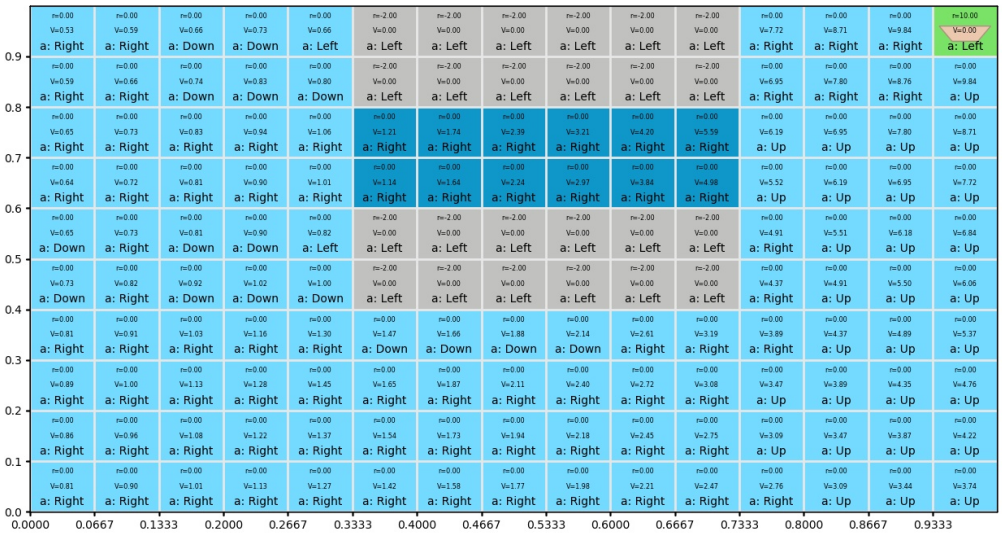


Figure 2: Actions in environment after 100 iterations

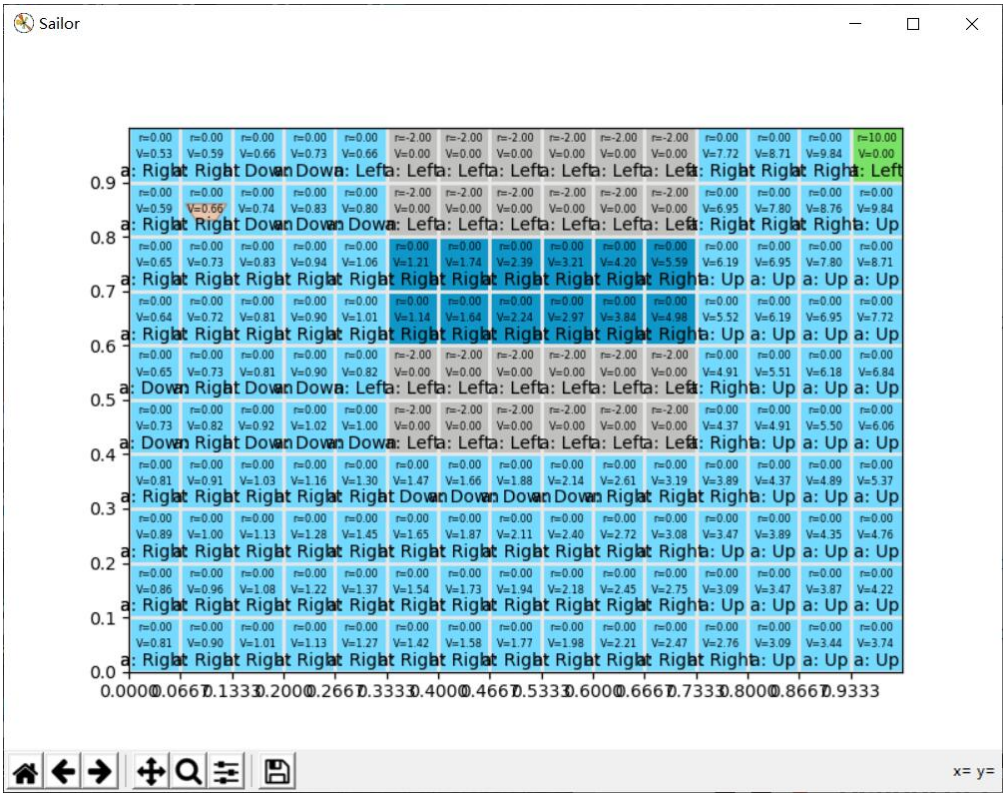
4 Task 2

The actions in environment is shown in Fig.2.

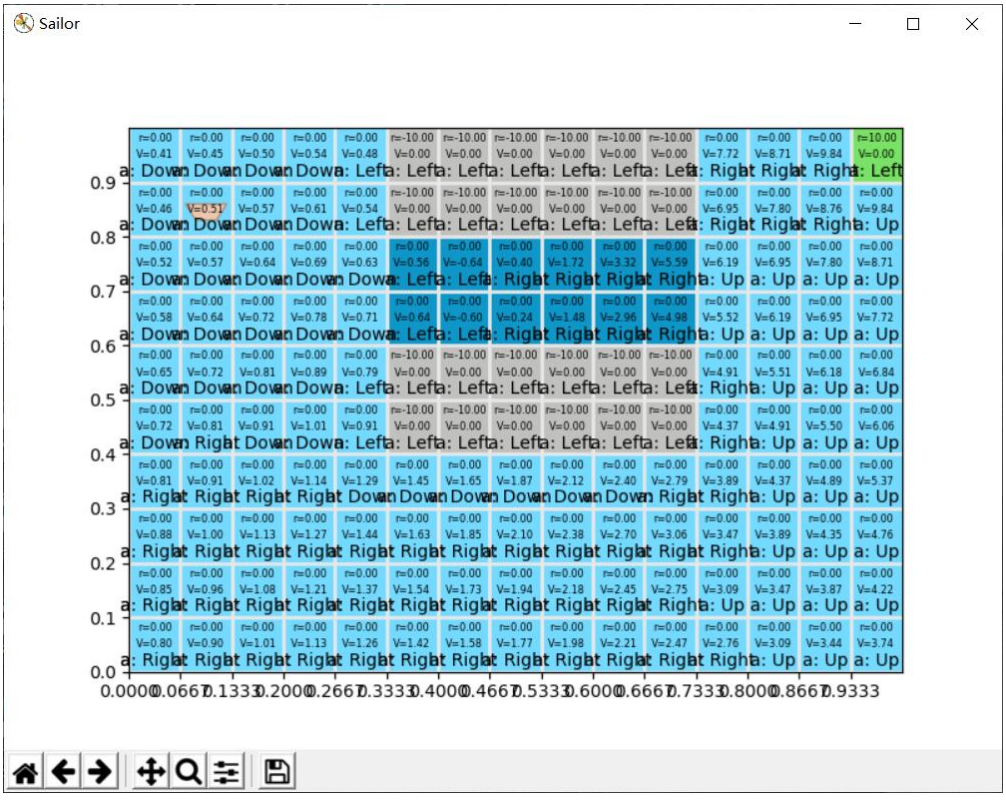
5 Question 3

- Rock reward=-2. The sailor tend to risk choosing the dangerous path between the rocks.
- Rock reward=-10. The sailor will choose the safe path below the rocks due to the smaller negative reward. The sailor value their own life more.

The comparison is in Fig.3.



(a) Rock reward:-2



(b) Rock reward:-10

Figure 3: Choices in different rock rewards

6 Question 4

```
def values_iterations(env,gamma=1.0,itNum=100):
    value_est, policy = np.zeros((env.w, env.h)), np.zeros((env
       .w, env.h))

    old_values=value_est.copy()
    old_policy=policy.copy()

    for it in range(itNum):
        # iterate every state in 2D
        env.clear_text()
        for w in range(env.w):
            for h in range(env.h):
                Q_value=[]
                for tran in env.transitions[w,h]:
                    A_value=0
                    for next_state,reward,done,prob in tran:
                        # consider all situation under
                           different prob
                        A_value+=prob*(reward+(gamma*value_est[
                           next_state] if not done else 0))
                    Q_value.append(A_value)
                # print(it,Q_value)
                value_est[w,h]=np.max(Q_value)
                policy[w,h]=np.argmax(Q_value)

        # value difference checks converge
        if ((value_est-old_values)<epsilon).all():
            print("converge at "+ str(it))
            break
        else:
            # print("Not converge at "+ str(it))
            old_values=np.copy(value_est)           # should
               use np.copy, directly equal causes fault

        # policy difference checks converge
        if ((policy-old_policy)<epsilon).all():
            print("Policy converge at "+ str(it))
            break
        else:
            print("Policy didn't converge at "+ str(it))
            old_policy=np.copy(policy)
    return value_est,policy
```

I implemented the convergence check in original function at Task 1, using the formula in introduction file. There are two types of data to detect convergence, the value function and

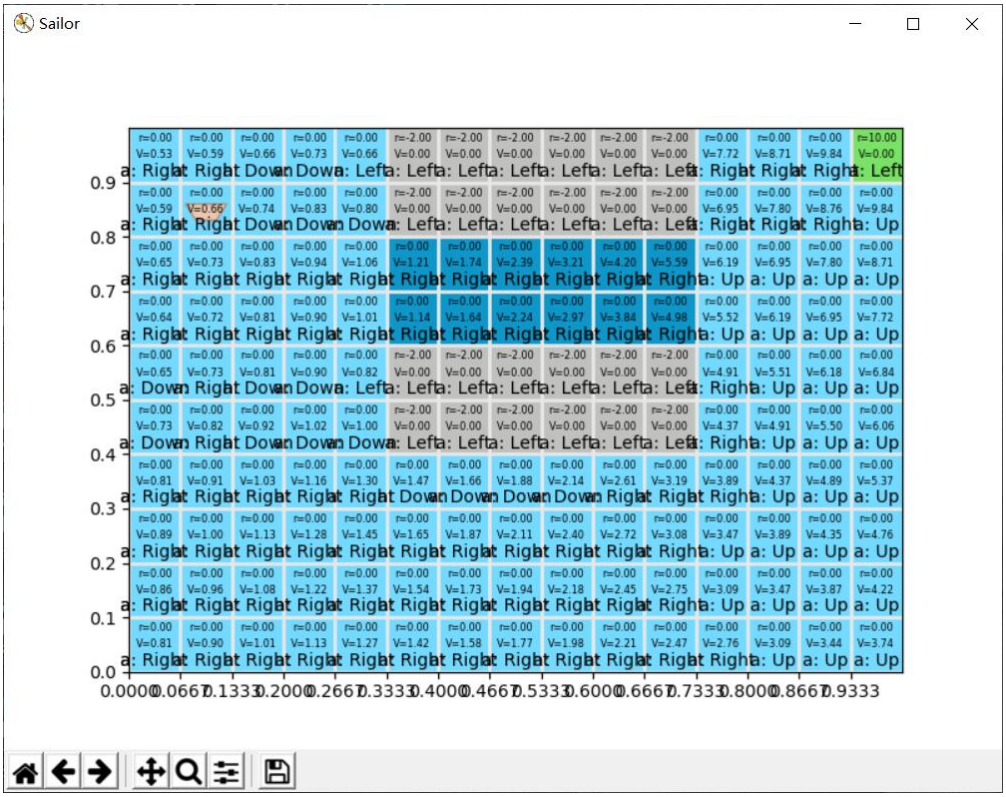


Figure 4: Convergence at rock reward=-2

the policy. After test in both of them, it shows they converged at different iterations. from my experiment, the value function converged at iteration 29, whereas the policy at 23. The policy converged sooner than the value function does.The reason is that the value funtion needs the whole state space to take into account. A small change in state values does not necessarily lead to a change in policy.

7 Task 3

The algorithm converged at iteration 39, when the rock reward equals -2. The grid is shown in Fig.4.

8 Task 4

After 1000 episodes:

- Average: 0.683
- Standard deviation: 1.361

```
    results=[]
    for ep in range(1000):
        discount_return=0
        i=0
        done = False
        while not done:
            # Select a random action
            # TODO: Use the policy to take the optimal action (
               Task 2)
            action = policy[state]

            # Step the environment
            state, reward, done, _ = env.step(action)
            discount_return+=reward*(gamma**i)
            i+=1
            # Render and sleep
            # env.render()
            # sleep(0.2)

        print("Episode"+ str(ep)+": "+ str(round(
           discount_return,5)))
        results.append(discount_return)

        state = env.reset()

    # print(results)
    print("--------Average& STD") # 0.683 & 1.361
    print(np.average(results))
    print(np.std(results))
```

9 Question 5

From the equation below we know, the value function of a state s under a policy π , denoted $v(s)$, is the expected discounted return when starting in s and following thereafter. [Chapter 3 Sutto & Barto]

$$v_\pi(s) = E_\pi[G_t|S_t = s] \tag{1}$$

10 Question 6

No. It cannot directly apply into the problem involving a robot exploring an **unknown** environment. Mainly because in our exercise, we have finite state space of environment with certain limited number of actions. What's more, in a more real world setting computing the values in different states like we did in this exercies will be computionally expensive. You can

see there are 3 or 4 loops in code, requiring a lot of time and computing power when states become numerous. Lastly, the various states and actions in exercise have been abstractly expressed so clearly while it is difficult to abstract the complexity of the situation in a real application problem.

11 Feedback

1. 5h