Exercise 2

Christian Montecchiani - 100655834
ELEC-E8125 - Reinforcement Learning

September 26, 2022

1 Task 1

In the first task of this second assignment we were required to implement the value iteration algorithm. Below is reported my implementation.

```
        ########## Your code starts here ##########
        temp_table = np.zeros((env.w, env.h))
        for x in range(env.w):
            for y in range(env.h):
                value_list = np.zeros(env.n_actions)
                for action in range(env.n_actions):
                    transitions = env.transitions[x, y, action]
                    action_value = 0
                    for transition in transitions:
                        next_state, reward, done, prob = transition.state, transition.reward, transition.done, transition.prob
                        if done:
                            action_value += prob * reward
                            continue
                        action_value += prob * (reward + gamma * v_est[next_state[0], next_state[1]])
                    value_list[action] = action_value

                next_value = np.max(value_list)
                temp_table[x, y] = next_value
                policy[x, y] = np.argmax(value_list)

        v_est = temp_table
        env.draw_values_policy(v_est, policy)
        ########## Your code ends here ##########
```

1.1 Question 1.1

In this exercise the **agent** is represented by the *boat* and the **environment** is represented by the *grid world*.

1.2 Question 1.2

The state values of the harbor and rock states are zero. We obtain this result because they are *terminal states* and during the running of the algorithm they are never updated.

1.3 Question 1.3

If the penalty for hitting the rocks is set to -2 the sailor chooses the dangerous path between the rocks. If the reward for hitting the rocks is set to -10 then the sailor prefers to choose the longer but safer path.

2 Task 2

The state value function converges with a threshold set equal to 10^{-4} during the 42th iteration. This means that if I run the algorithm for 30 iterations the state value function still did not converge. Instead, if we check the policy, it converges faster. I check the convergence of the policy in two ways:

1. Check the changes between two consequent iterations and the policy converges at the 30th iteration.
2. Check the last time that the policy changes through all the iterations. It converges at iteration 32.

Generally, the policy needs less iteration to converge, which means that to find an optimal policy π^* is not required to compute the exact full value function.

```
        ########## Your code starts here ##########
        temp_value = np.zeros((env.w, env.h))
        temp_policy = np.zeros((env.w, env.h))

        for x in range(env.w):
            for y in range(env.h):
                value_list = np.zeros(env.n_actions)

                for action in range(env.n_actions):
                    transitions = env.transitions[x, y, action]
                    action_value = 0
                    for transition in transitions:
                        next_state, reward, done, prob = transition.state, transition.reward, transition.done, transition.prob
                        if done:
                            action_value += prob * reward
                            continue
                        action_value += prob * (reward + gamma * v_est[next_state[0], next_state[1]])
                    value_list[action] = action_value

                next_value = np.max(value_list)
                temp_value[x, y] = next_value
                temp_policy[x, y] = np.argmax(value_list)

        # Value function convergence
        delta = np.abs(v_est - temp_value).max()
        if delta < eps and i > 0 and first:
            first = False
            print(f"Value function has converged during iteration: {i}")

        v_est = temp_value

        # Policy convergence
        if np.array_equal(temp_policy, policy) == False:
            last_change = i
        policy = temp_policy

        env.draw_values_policy(v_est, policy)

        if i +1 == iterations:
            print(f"The last change in the policy is: {last_change}")
        ########## Your code ends here ##########
```

3 Task 3

The number of iterations required for the value function to converge, when reward for crashing into the rocks is -2, is 38. The code used is the same as the previous task.

4 Task 4

In this task we had to evaluate the learned policy and to do that we compute the *discounted return*:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

(as reported in [1]) of the initial state. The code is shown below.

```
# Eval policy
N = 1000 # TODO: change for task 4
returns = list()
for ep in range(N):
    if (ep + 1) % 200 == 0:
        print(f"Episodes {ep+1}/{N}")
    state = env.reset()
    done = False
    rewards = list()
    while not done:
        ########## You code starts here ##########

        action = policy[state[0], state[1]]
        # Take a step in the environment
        state, reward, done, _ = env.step(action)
        # Calculate discounted return for the initial state
        rewards.append(reward)
        if done:
            G = np.sum(np.array([(gamma**i) * rew for i,rew in enumerate(rewards)]))
            returns.append(G)
        ########## Your code ends here ##########
```

The mean return and the standard deviation over 1000 episodes are: 0.66 and 1.36, respectively.

4.1 Question 4.1

As stated in [1] in Eq.3.12 the value function is:

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s \right]$$

Basically, the value function in a given state, s, is the expected discounted return started in that state and following policy π.

4.2 Question 4.2

In a reinforcement learning problem involving a robot exploring an *unknown* environment the value iteration approach cannot be applied. That is because the value iteration approach requires the to be in a fully observable Markov Decision Process (MDP). And this is the *unrealistic* the assumption that does not permits to use of the value iteration approach in each type of RL problems, since the majority of the environments, are not totally observable.

References

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