Homework 4: Swamp

劉安得

106061151

1. Implementation

- choose_action()

  It is better to always choose the best option, but to keep the exploration going, we sometimes choose random action with a probability of ϵ.

```
prob = rd.random()

if prob < epsilon:
    action = rd.randint(len(ACTIONS))
else:
    action = rd.choice(np.flatnonzero(q_table[state[0], state[1]] ==
np.amax(q_table[state[0], state[1]])))

return action
```

- step()

  Build the environment, R = -100 if moves into the swamp; all other transitions yield R = -1

```
next_state = (state + action).tolist()
x, y = next_state
if x < 0 or x >= 4 or y < 0 or y >= 3:
    next_state = state

x, y = next_state
if y == 2:
    reward = -100
else:
    reward = -1

return next_state, reward
```

- episode()

  Implement On-policy first-visit MC control (for ε-soft policies), pseudo code as shown below.

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$

Initialize:

$\pi \leftarrow$ an arbitrary ε-soft policy

$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow G + R_{t+1}$

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:

Append G to $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$A^* \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken arbitrarily)

For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

To be noticed, I use Exponential recency-weighted average method rather than equal weighted average, and alpha is 0.1

$$Q_{n+1} \doteq Q_n + \alpha\Big[R_n - Q_n\Big],$$

where the step-size parameter $\alpha \in (0, 1]$ is constant.

```
returns = []
q_table = rd.rand(4, 3, len(ACTIONS))
q_table[ST[0], ST[1]] = 0
for i in range(500):
    pair = []
    R = []

    state = S0
    while np.any(state != ST):
        action = choose_action(state, q_table, EPSILON)
        next_state, reward = step(state, ACTIONS[action])
        pair.append([state[0], state[1], action])
        R.append(reward)
        state = next_state

    G = 0
    visited = []
    for j in range(len(pair)-1, -1, -1):
        G += R[j]
```

```
            if pair[j] not in visited:
                visited.append(pair[j])
                x = pair[j][0]
                y = pair[j][1]
                At = pair[j][2]
                q_table[x, y, At] += ALPHA * (G-q_table[x, y, At])

        returns.append(G)

    return returns, q_table
```

- drawTable()

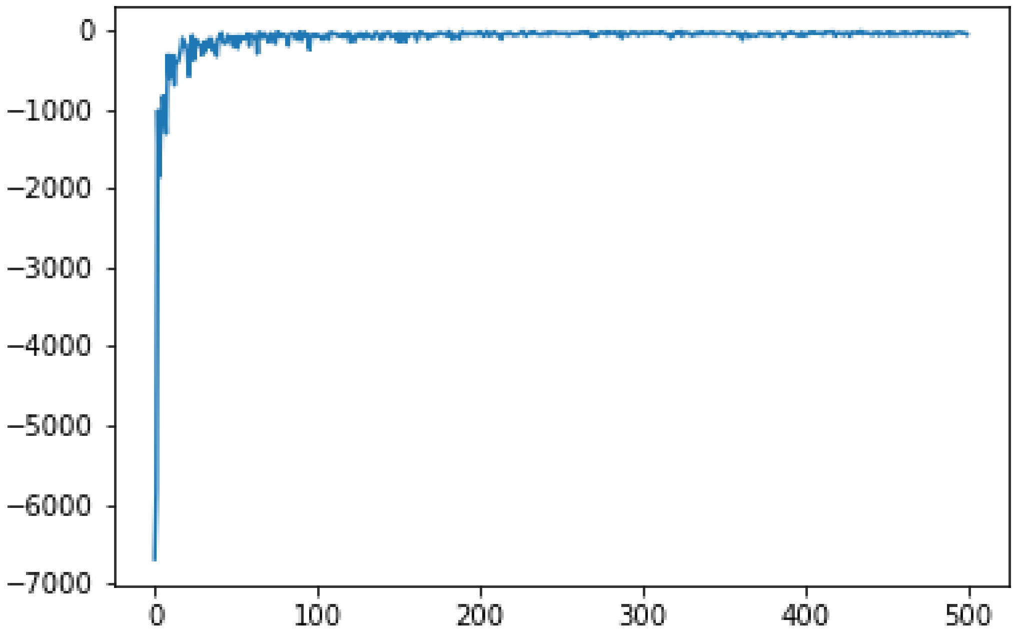
  plot the q-table, minor revise from HW3

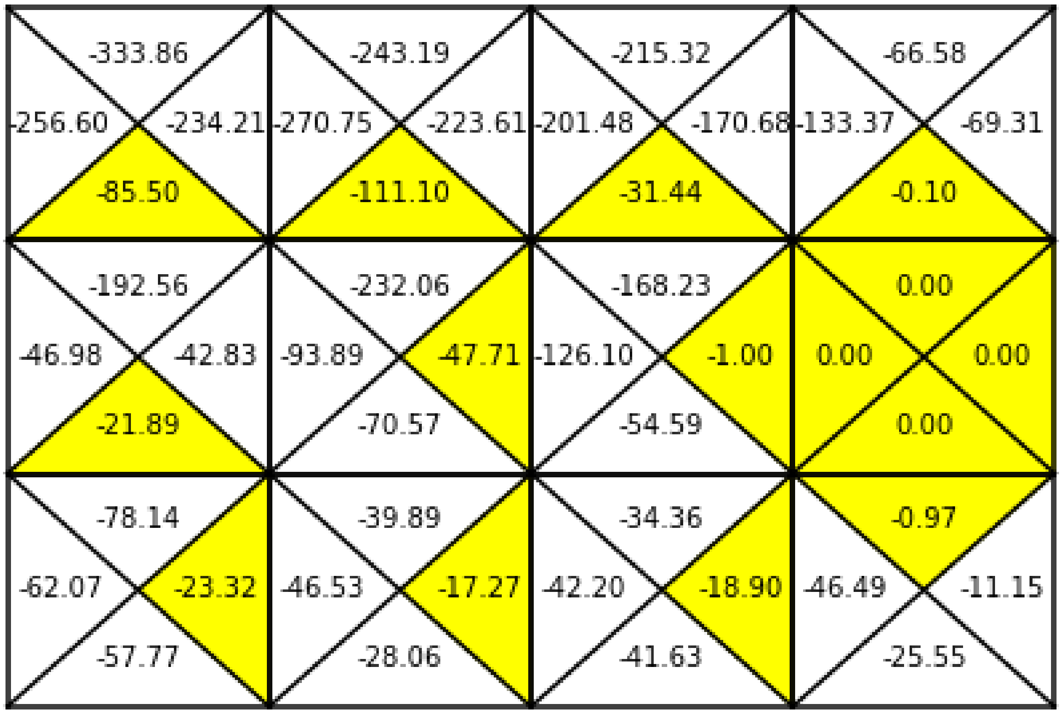
```
def drawTable(data):
  fig = plt.figure()
  ax = plt.Axes(fig, [0., 0., 1., 1.])
  ax.set_axis_off()
  fig.add_axes(ax)
  plt.gca().invert_yaxis()
  for i in range(4):
    for j in range(3):
      plt.plot((i,i),(j,j+1),'-k')
      plt.plot((i,i+1),(j,j+1),'-k')
      plt.plot((i,i+1),(j,j),'-k')
      plt.plot((i+1,i),(j,j+1),'-k')
      plt.plot((i+1,i+1),(j,j+1),'-k')
      plt.plot((i,i+1),(j+1,j+1),'-k')
      temp = max(data[i][2-j])
      if data[i][2-j][0]==temp:
        plt.gca().add_patch(plt.Polygon([[i+0.5,j+0.5], [i,j], [i,j+1]],
color='yellow'))
      if data[i][2-j][1]==temp:
        plt.gca().add_patch(plt.Polygon([[i+0.5,j+0.5], [i,j], [i+1,j]],
color='yellow'))
      if data[i][2-j][2]==temp:
        plt.gca().add_patch(plt.Polygon([[i+0.5,j+0.5], [i+1,j],
[i+1,j+1]], color='yellow'))
      if data[i][2-j][3]==temp:
        plt.gca().add_patch(plt.Polygon([[i+0.5,j+0.5], [i,j+1],
[i+1,j+1]], color='yellow'))

      plt.text(i+0.2,j+0.5,'%.2f' %data[i][2-j][0],
```

```
verticalalignment='center', horizontalalignment='center')
     plt.text(i+0.5,j+0.2,'%.2f' %data[i][2-j][1],
verticalalignment='center', horizontalalignment='center')
     plt.text(i+0.8,j+0.5,'%.2f' %data[i][2-j][2],
verticalalignment='center', horizontalalignment='center')
     plt.text(i+0.5,j+0.8,'%.2f' %data[i][2-j][3],
verticalalignment='center', horizontalalignment='center')
  # from google.colab import files #google.colab only
  plt.savefig("4.png")
  # files.download("3.png") #google.colab only
```

2. Experiments and Analysis





q_values are reasonable because the actions of optimal policy move forward to S_T, and try to avoid swamp.