強化学習による迷路探索

1. Epsilon-Greedy TD(0)

1.1 パラメータ：$\epsilon = 0.05$、0,2、0.5、0.8, $\gamma = 0.95$, $\alpha = 0.05$

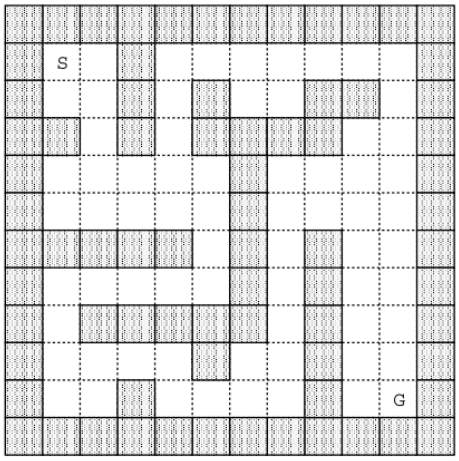


Figure 1: 迷路

1.2 初期化

価値関数を 0 に初期化する。すなわち、迷路の各座標の価値関数 V(x,y)=0 とする。

1.3 学習を行う

500 回のエピソードを繰り返し、その中で価値観数の更新を行う。エピソードごとに次の手順を繰り返す。

次の時刻でのエージェントの位置を以下の epsilon-greedy 方策で定める。ここで、(x', y')は、現在の位置から移動できるマスである。

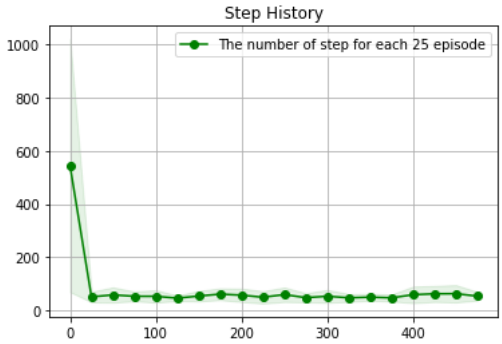
エージェントの新しい位置(x', y')がゴール(G)ならば、r=1、そうでなければ r=0 とする。

価値関数の更新：$V(x,y) := V(x,y) + \alpha(r + \gamma V(x', y') - V(x,y))$

1.4 結果

$\epsilon = 0.2$の場合：

探索を行った 50 エピソードごとにゴールまでの平均ステップ数をプロットした。最小ステップである 60 付近に漸近していることが分かる。



At Episode 50 average steps is 48.32 (+/-18.406)

At Episode 100 average steps is 54.32 (+/-19.231)

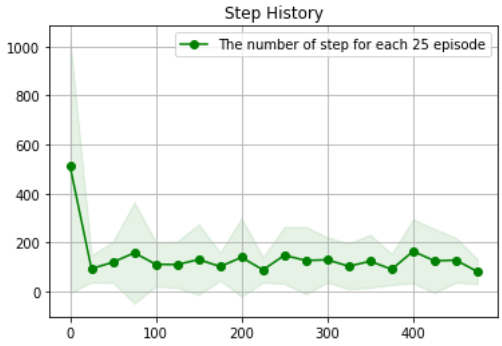
At Episode 150 average steps is 44.72 (+/-9.861)
At Episode 200 average steps is 62.64 (+/-22.319)
At Episode 250 average steps is 51.28 (+/-23.805)
At Episode 300 average steps is 48.48 (+/-16.784)
At Episode 350 average steps is 48.24 (+/-15.953)
At Episode 400 average steps is 46.8 (+/-13.303)
At Episode 450 average steps is 61.92 (+/-30.287)

また、学習終了時の各マスの価値関数から、経路を確認した。正しい結果が示された。

```
[[0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.   ]
 [0.    0.026 0.02  0.    0.122 0.135 0.143 0.167 0.195 0.212 0.229 0.   ]
 [0.    0.034 0.04  0.    0.113 0.    0.075 0.094 0.    0.    0.25  0.   ]
 [0.    0.    0.048 0.    0.1   0.    0.    0.    0.    0.243 0.267 0.   ]
 [0.    0.012 0.055 0.068 0.086 0.013 0.    0.007 0.165 0.293 0.282 0.   ]
 [0.    0.002 0.023 0.021 0.03  0.001 0.    0.006 0.22  0.318 0.276 0.   ]
 [0.    0.    0.    0.    0.    0.    0.    0.    0.    0.34  0.293 0.   ]
 [0.    0.    0.    0.    0.    0.    0.    0.    0.    0.369 0.369 0.   ]
 [0.    0.    0.    0.    0.    0.    0.    0.    0.    0.406 0.425 0.   ]
 [0.    0.    0.    0.    0.    0.    0.    0.    0.    0.435 0.5   0.   ]
 [0.    0.    0.    0.    0.    0.    0.    0.    0.    0.421 0.    0.   ]
 [0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.   ]]
```

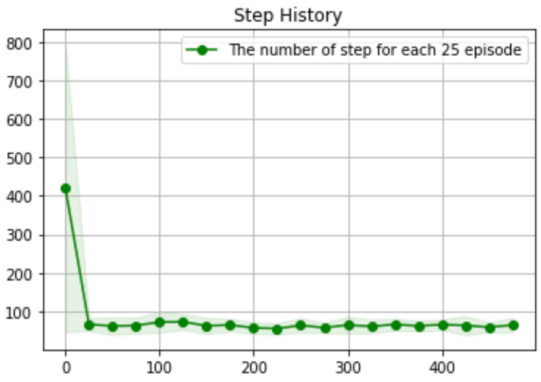
次に、ϵ の値を変化させて実験をいくつか行った。

$\epsilon = 0.05$の場合：



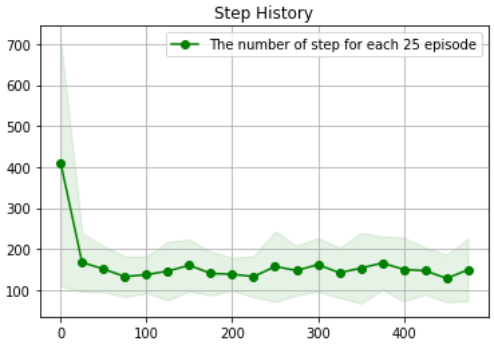
At Episode 50 average steps is 92.64 (+/-55.222)
At Episode 100 average steps is 158.88 (+/-205.864)
At Episode 150 average steps is 112.96 (+/-94.82)
At Episode 200 average steps is 103.6 (+/-57.408)
At Episode 250 average steps is 84.64 (+/-37.714)
At Episode 300 average steps is 133.92 (+/-138.35)
At Episode 350 average steps is 102.08 (+/-94.314)
At Episode 400 average steps is 105.36 (+/-96.578)
At Episode 450 average steps is 131.12 (+/-129.072)

$\epsilon = 0.5$の場合：



At Episode 50 average steps is 66.24 (+/-16.631)
At Episode 100 average steps is 63.2 (+/-20.451)
At Episode 150 average steps is 72.4 (+/-21.174)
At Episode 200 average steps is 64.88 (+/-16.1)
At Episode 250 average steps is 55.28 (+/-15.251)
At Episode 300 average steps is 58.88 (+/-13.66)
At Episode 350 average steps is 61.52 (+/-17.657)
At Episode 400 average steps is 61.92 (+/-14.327)
At Episode 450 average steps is 64.0 (+/-24.384)

$\epsilon = 0.8$の場合：



At Episode 50 average steps is 164.0 (+/-72.084)
At Episode 100 average steps is 130.32 (+/-47.972)
At Episode 150 average steps is 147.04 (+/-71.232)
At Episode 200 average steps is 137.6 (+/-54.17)
At Episode 250 average steps is 138.0 (+/-55.327)
At Episode 300 average steps is 143.2 (+/-58.812)
At Episode 350 average steps is 141.6 (+/-61.259)
At Episode 400 average steps is 174.32 (+/-79.633)
At Episode 450 average steps is 144.08 (+/-55.617)

ϵを大きくすることで、探索がより進み、平均ステップ数の収束が早くなっていることが分かる。しかしながら、ϵを大きくすることで、収束した平均ステップ数の値が大きくなってしまうこと、分散が大きくなってしまっていることが見られる。これは ε は探索と利用の

トレードオフのパラメータであり、 ε を大きくすることで、探索優先度が上がり、様々な状態について学習が進むが、その利用が少ないために、学習による最効率の結果がでないという結果となった。
逆に、ϵの値を小さくすることで、探索による学習の進行は遅くなり、平均ステップ数の収束が遅くなっていることが分かる。収束した平均ステップ数の値が大きくなり、分散が大きくなってしまっていることが見られる。
実験の結果から、$\epsilon = 0.2$は最適のϵである。

2. Q 学習法

2.1 パラメータ：$\epsilon = 0.05$、0.2、0.5、0.8, $\gamma = 0.95$, $\alpha = 0.05$

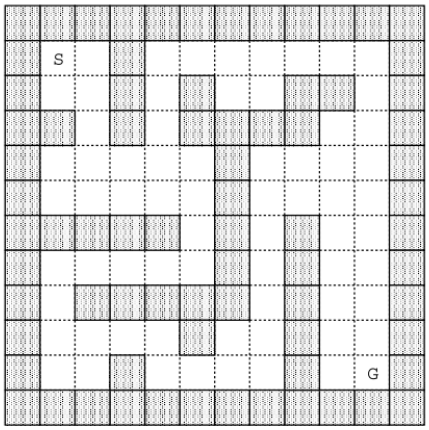


Figure 2: 迷路

2.2 初期化

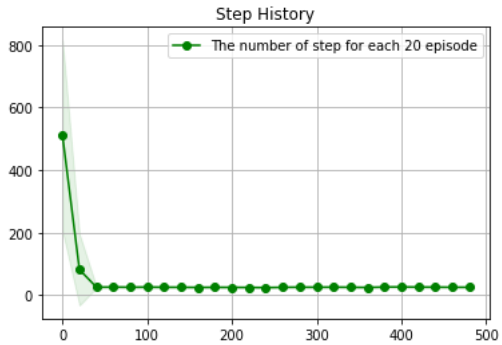
価値関数を 0 に初期化する。

2.3 学習を行う

500 回のエピソードを繰り返し、その中で価値観数の更新を行う。
次の時刻でのエージェントの位置を以下の epsilon-greedy 方策で定める。ここで、(x', y')は、現在の位置から移動できるマスである。
エージェントの新しい位置(x', y')がゴール(G)ならば、r=1、そうでなければ r=0 とする。
価値関数の更新：$Q(s,a) := Q(s,a) + \alpha(r + \gamma Q(s', a') - Q(s,a))$

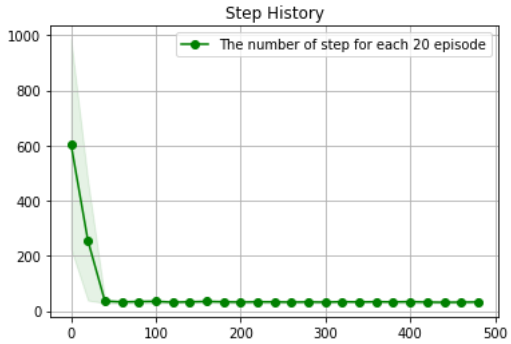
2.4 結果

$\epsilon = 0.05$の場合：



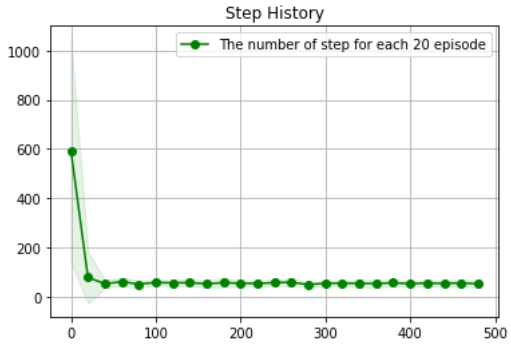
At Episode 50 average steps is 25.5 (+/-1.884)
At Episode 100 average steps is 26.0 (+/-3.033)
At Episode 150 average steps is 25.9 (+/-2.322)
At Episode 200 average steps is 25.2 (+/-2.04)
At Episode 250 average steps is 24.4 (+/-1.02)
At Episode 300 average steps is 25.6 (+/-2.059)
At Episode 350 average steps is 25.3 (+/-1.706)
At Episode 400 average steps is 26.5 (+/-1.658)
At Episode 450 average steps is 25.8 (+/-2.441)

$\epsilon = 0.2$の場合：



At Episode 50 average steps is 64.0 (+/-49.014)
At Episode 100 average steps is 34.5 (+/-5.617)
At Episode 150 average steps is 32.7 (+/-3.48)
At Episode 200 average steps is 33.0 (+/-5.04)
At Episode 250 average steps is 32.8 (+/-3.544)
At Episode 300 average steps is 32.8 (+/-4.75)
At Episode 350 average steps is 33.5 (+/-3.154)
At Episode 400 average steps is 32.6 (+/-4.294)
At Episode 450 average steps is 32.4 (+/-4.176)

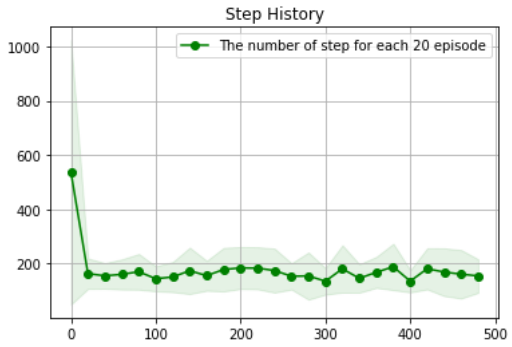
$\epsilon = 0.5$の場合：



At Episode 50 average steps is 46.8 (+/-15.302)
At Episode 100 average steps is 49.6 (+/-14.961)

At Episode 150 average steps is 56.0 (+/-13.871)
At Episode 200 average steps is 58.3 (+/-18.998)
At Episode 250 average steps is 53.4 (+/-11.386)
At Episode 300 average steps is 50.0 (+/-9.757)
At Episode 350 average steps is 52.2 (+/-13.681)
At Episode 400 average steps is 55.8 (+/-14.791)
At Episode 450 average steps is 56.0 (+/-13.191)

$\epsilon = 0.8$の場合：



At Episode 50 average steps is 160.9 (+/-52.655)
At Episode 100 average steps is 160.7 (+/-60.898)
At Episode 150 average steps is 168.5 (+/-70.933)
At Episode 200 average steps is 181.4 (+/-77.934)
At Episode 250 average steps is 170.9 (+/-75.722)
At Episode 300 average steps is 150.4 (+/-88.004)
At Episode 350 average steps is 154.5 (+/-56.635)
At Episode 400 average steps is 186.6 (+/-87.249)
At Episode 450 average steps is 186.2 (+/-89.492)

ϵを大きくすることで、探索がより進み、平均ステップ数の収束が早くなっていることが分かる。しかしながら、ϵを大きくすることで、収束した平均ステップ数の値が大きくなってしまうこと、分散が大きくなってしまっていることが見られる。これはϵは探索と利用のトレードオフのパラメータであり、ϵを大きくすることで、探索優先度が上がり、様々な状態について学習が進むが、その利用が少ないために、学習による最効率の結果がでないという結果となった。
逆に、ϵの値を小さくすることで、探索による学習の進行は遅くなり、平均ステップ数の収束が遅くなっていることが分かる。q-learning 法は TD 法とは異なることは、ϵの値を小さくすることで、探索による学習の進行は遅くなってしまうが、利用の優先度が高く、最終的には最適な行動を取り続けていると考えられる。
実験の結果から、$\epsilon = 0.05$は最適のϵである。

3.　もっと大きく難しい迷路を作った。

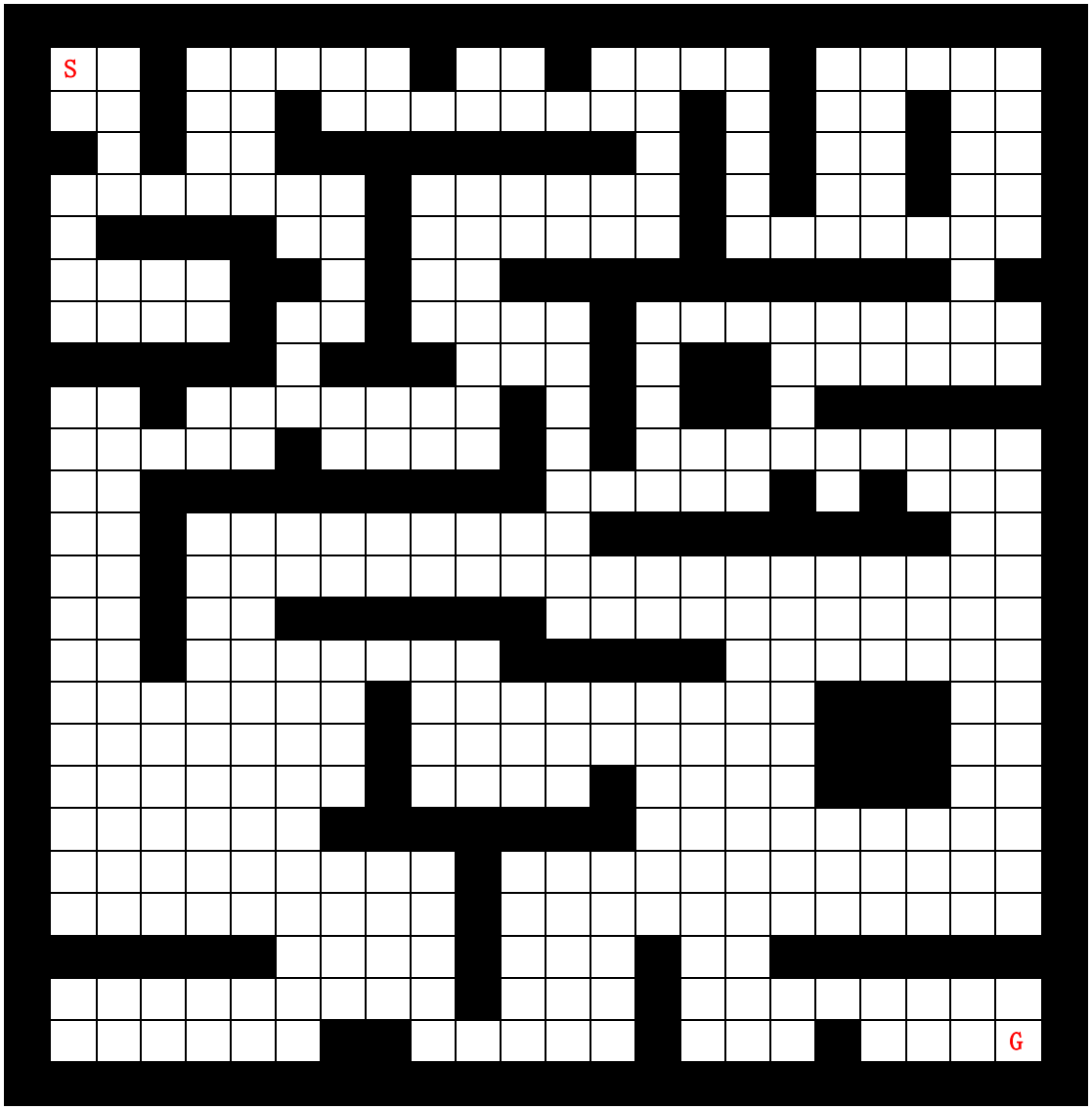
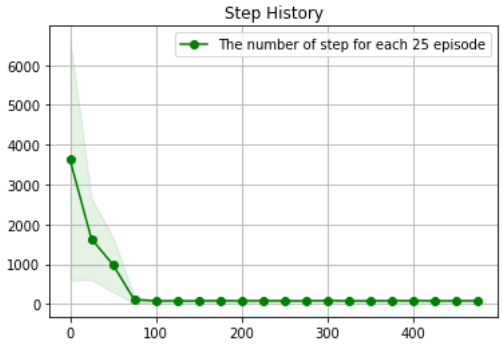


Figure 3: もっと難しい迷路

3.1 TD 法

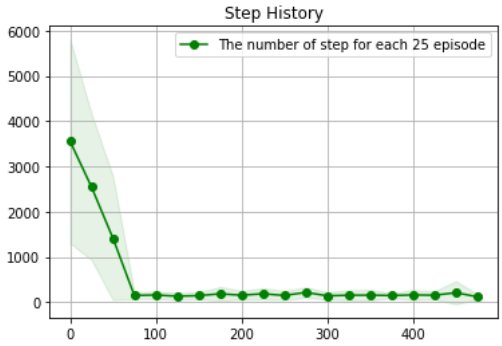
パラメータ：$\gamma = 0.95,\ \alpha = 0.05$

$\epsilon = 0.2$の場合：



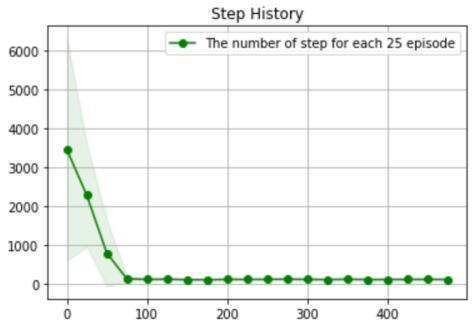
At Episode 50 average steps is 1594.64 (+/-1047.026)
At Episode 100 average steps is 110.16 (+/-91.58)
At Episode 150 average steps is 81.12 (+/-19.042)
At Episode 200 average steps is 85.84 (+/-24.683)
At Episode 250 average steps is 87.52 (+/-29.569)
At Episode 300 average steps is 82.16 (+/-21.473)
At Episode 350 average steps is 79.76 (+/-13.474)
At Episode 400 average steps is 85.84 (+/-17.654)
At Episode 450 average steps is 83.2 (+/-23.799)

$\epsilon = 0.05$の場合：



At Episode 50 average steps is 2584.48 (+/-1587.607)
At Episode 100 average steps is 140.32 (+/-52.917)
At Episode 150 average steps is 133.44 (+/-76.916)
At Episode 200 average steps is 183.6 (+/-154.802)
At Episode 250 average steps is 190.48 (+/-125.164)
At Episode 300 average steps is 221.6 (+/-114.834)
At Episode 350 average steps is 150.48 (+/-112.1)
At Episode 400 average steps is 146.08 (+/-55.611)
At Episode 450 average steps is 151.04 (+/-87.555)

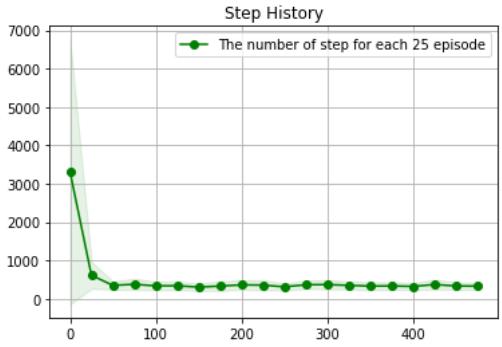
$\epsilon = 0.5$の場合：



At Episode 50 average steps is 2257.2 (+/-1346.022)
At Episode 100 average steps is 137.6 (+/-59.668)
At Episode 150 average steps is 129.92 (+/-30.207)
At Episode 200 average steps is 116.4 (+/-21.518)

At Episode 250 average steps is 122.72 (+/-21.732)
At Episode 300 average steps is 128.96 (+/-26.297)
At Episode 350 average steps is 118.08 (+/-21.44)
At Episode 400 average steps is 120.0 (+/-27.788)
At Episode 450 average steps is 123.04 (+/-20.318)

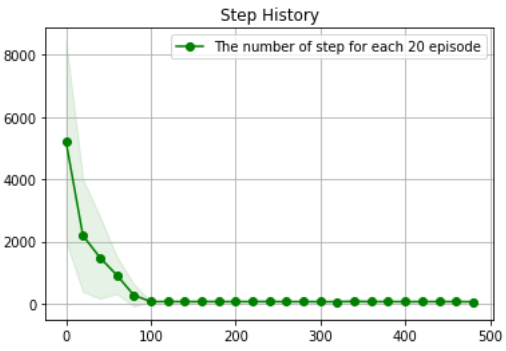
$\epsilon = 0.8$の場合：



At Episode 50 average steps is 598.16 (+/-357.615)
At Episode 100 average steps is 378.96 (+/-144.97)
At Episode 150 average steps is 338.96 (+/-111.41)
At Episode 200 average steps is 321.12 (+/-124.646)
At Episode 250 average steps is 352.72 (+/-135.426)
At Episode 300 average steps is 358.08 (+/-103.922)
At Episode 350 average steps is 337.84 (+/-98.635)
At Episode 400 average steps is 334.88 (+/-100.956)
At Episode 450 average steps is 370.88 (+/-130.471)

3.2 Q学習法

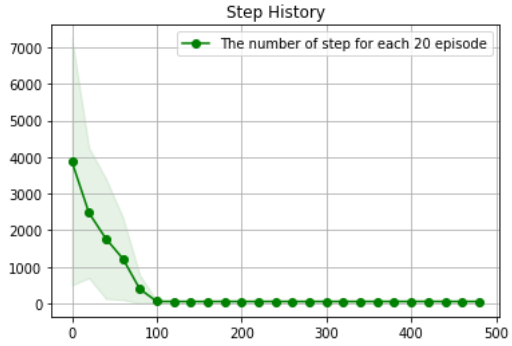
$\epsilon = 0.2$の場合：



At Episode 50 average steps is 1792.5 (+/-1250.485)
At Episode 100 average steps is 238.0 (+/-335.086)
At Episode 150 average steps is 68.7 (+/-8.301)
At Episode 200 average steps is 69.9 (+/-7.279)
At Episode 250 average steps is 69.7 (+/-8.539)

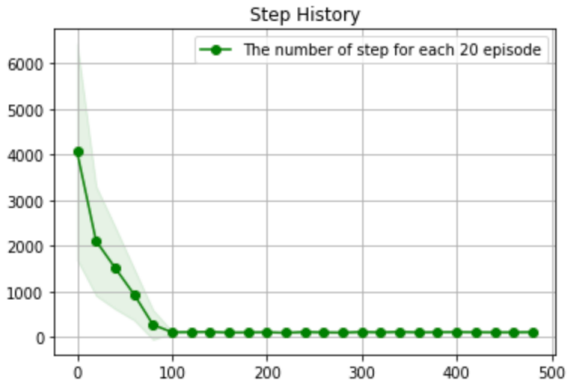
At Episode 300 average steps is 67.8 (+/-7.04)
At Episode 350 average steps is 79.1 (+/-46.067)
At Episode 400 average steps is 70.3 (+/-7.191)
At Episode 450 average steps is 70.2 (+/-7.666)

$\epsilon = 0.05$の場合：



At Episode 50 average steps is 2028.5 (+/-1697.384)
At Episode 100 average steps is 376.4 (+/-395.676)
At Episode 150 average steps is 54.8 (+/-2.713)
At Episode 200 average steps is 55.6 (+/-2.728)
At Episode 250 average steps is 55.2 (+/-2.857)
At Episode 300 average steps is 54.4 (+/-2.653)
At Episode 350 average steps is 55.4 (+/-2.289)
At Episode 400 average steps is 56.1 (+/-2.567)
At Episode 450 average steps is 55.5 (+/-2.82)

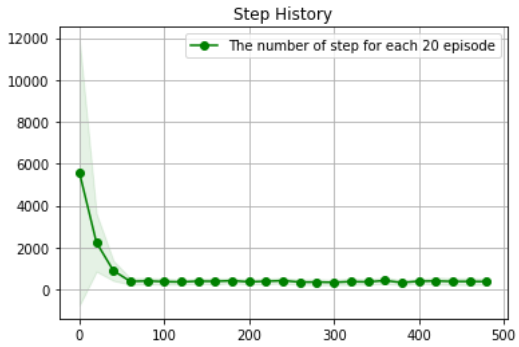
$\epsilon = 0.5$の場合：



At Episode 50 average steps is 1858.3 (+/-905.587)
At Episode 100 average steps is 219.5 (+/-231.872)
At Episode 150 average steps is 117.2 (+/-21.141)
At Episode 200 average steps is 108.6 (+/-16.806)
At Episode 250 average steps is 103.4 (+/-12.978)
At Episode 300 average steps is 102.8 (+/-21.056)
At Episode 350 average steps is 111.0 (+/-16.426)
At Episode 400 average steps is 113.2 (+/-14.905)

At Episode 450 average steps is 106.1 (+/-19.468)

$\epsilon = 0.8$の場合：



At Episode 50 average steps is 1456.9 (+/-896.46)
At Episode 100 average steps is 405.2 (+/-149.032)
At Episode 150 average steps is 376.7 (+/-114.146)
At Episode 200 average steps is 428.0 (+/-107.15)
At Episode 250 average steps is 409.2 (+/-103.184)
At Episode 300 average steps is 358.7 (+/-136.143)
At Episode 350 average steps is 355.6 (+/-128.079)
At Episode 400 average steps is 337.8 (+/-107.18)
At Episode 450 average steps is 432.9 (+/-164.354)

大きい図の場合には小さい図と同じ、ϵを大きくすることで、探索がより進み、平均ステップ数の収束が早くなっている。しかしながら、ϵを大きくすることで、収束した平均ステップ数の値が大きくなってしまうこと、分散が大きくなってしまっていることが見られる。これはϵは探索と利用のトレードオフのパラメータであり、ϵを大きくすることで、探索優先度が上がり、様々な状態について学習が進むが、その利用が少ないために、学習による最効率の結果がでないという結果となった。
逆に、ϵの値を小さくすることで、探索による学習の進行は遅くなり、平均ステップ数の収束が遅くなっていることが分かる。
TD(0)方法では、ϵの値を小さくすることで、収束した平均ステップ数の値が大きくなり、分散が大きくなってしまっていることが見られる。
Q学習法では、ϵの値を小さくすることで、探索による学習の進行は遅くなってしまうが、利用の優先度が高く、最終的には最適な行動を取り続けていると考えられる。
実験の結果から、TD法では$\epsilon = 0.2$は最適のϵである。Q学習法では$\epsilon = 0.05$は最適のϵである。

Code：

(1)　Epsilon-Greedy TD(0)アルゴリズム

```
import numpy as np
import matplotlib.pyplot as plt
from enum import Enum
from collections import defaultdict
import decimal

class State():

    def __init__(self, row=-1, column=-1):
        self.row = row
        self.column = column

    def __repr__(self):
        return "<State: [{}, {}]>".format(self.row, self.column)

    def clone(self):
        return State(self.row, self.column)

    def __hash__(self):
        return hash((self.row, self.column))

    def __eq__(self, other):
        return self.row == other.row and self.column == other.column


class Action(Enum):
    UP = 0
    DOWN = 1
    LEFT = 2
    RIGHT = 3

class Environment():

    def __init__(self, grid, move_prob=1.0):
        self.grid = grid
        self.agent_state = State()
        self.default_reward = 0
        self.move_prob = move_prob
        self.reset()

    @property
    def row_length(self):
        return len(self.grid)

    @property
    def column_length(self):
        return len(self.grid[0])

    @property
    def actions(self):
        return [Action.UP, Action.DOWN,
                Action.LEFT, Action.RIGHT]

    @property
    def states(self):
        states = []
        for row in range(self.row_length):
            for column in range(self.column_length):
                # Block cells are not included to the state.
```

```
                if self.grid[row][column] != 9:
                    states.append(State(row, column))
        return states


    def can_action_at(self, state):
        if self.grid[state.row][state.column] == 0 or -1:
            return True
        else:
            return False

    def can_action(self, state, actions):
        can_actions = []
        # Check whether the agent bumped a block cell.
        if self.grid[state.row -1][state.column] != 9:
            can_actions.append(0)
        if self.grid[state.row +1][state.column] != 9:
            can_actions.append(1)
        if self.grid[state.row][state.column -1] != 9:
            can_actions.append(2)
        if self.grid[state.row][state.column +1] != 9:
            can_actions.append(3)
        return can_actions


    def _move(self, state, action):
        if not self.can_action_at(state):
            print(state.row, state.column)
            raise Exception("Can't move from here!")

        next_state = state.clone()

        # Execute an action (move).
        if action == 0:
            next_state.row -= 1
        elif action == 1:
            next_state.row += 1
        elif action == 2:
            next_state.column -= 1
        elif action == 3:
            next_state.column += 1

        # Check whether a state is out of the grid.
        if not (0 <= next_state.row < self.row_length):
            next_state = state
        if not (0 <= next_state.column < self.column_length):
            next_state = state

        # Check whether the agent bumped a block cell.
        if self.grid[next_state.row][next_state.column] == 9:
            next_state = state

        return next_state

    def reward_func(self, state):
        reward = self.default_reward
        done = False

        # Check an attribute of next state.
        attribute = self.grid[state.row][state.column]
        if attribute == -1:
```

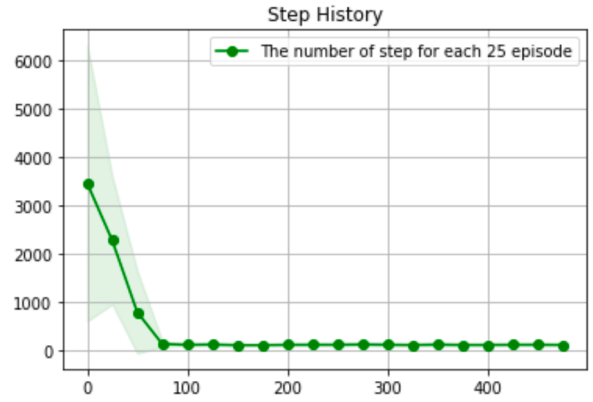
```
            # Get reward! and the game ends.
            reward = 1
            done = True

        return reward, done

    def reset(self):
        # Locate the agent at lower left corner.
        self.agent_state = State(1, 1)
        return self.agent_state

    def step(self, action):
        next_state, reward, done = self.transit(self.agent_state, action)
        if next_state is not None:
            self.agent_state = next_state

        return next_state, reward, done

    def transit(self, state, action):
        next_state = self._move(state, action)
        reward, done = self.reward_func(next_state)
        return next_state, reward, done


class ELAgent():

    def __init__(self, epsilon):
        self.V = {}
        self.epsilon = epsilon
        self.reward_log = []

    def policy(self, env, s, actions):
        if np.random.random() < self.epsilon:
            return actions[np.random.randint(len(actions))]
        else:
            if len(actions) == 1:
                next_state1 = env._move(s,actions[0])
                return actions[0]
            elif len(actions) == 2:
                next_state1 = env._move(s,actions[0])
                next_state2 = env._move(s,actions[1])
                reward1, done = env.reward_func(next_state1)
                reward2, done = env.reward_func(next_state2)
                if self.V[next_state1]+self.V[next_state2]!= 0:
                    return actions[np.argmax([self.V[next_state1],
                                              self.V[next_state2]])]
                else:
                    return actions[np.random.randint(len(actions))]
            elif len(actions) == 3:
                next_state1 = env._move(s,actions[0])
                next_state2 = env._move(s,actions[1])
                next_state3 = env._move(s,actions[2])
                if self.V[next_state1]+self.V[next_state2]+self.V[next_state3] != 0:
                    return actions[np.argmax([self.V[next_state1],
                                              self.V[next_state2],
                                              self.V[next_state3]])]
                else:
                    return actions[np.random.randint(len(actions))]
            elif len(actions) == 4:
                next_state1 = env._move(s,actions[0])
                next_state2 = env._move(s,actions[1])
```

```
                next_state3 = env._move(s,actions[2])
                next_state4 = env._move(s,actions[3])
                if self.V[next_state1]+self.V[next_state2]
                                      +self.V[next_state3]
                                      +self.V[next_state4] != 0:
                    return actions[np.argmax([self.V[next_state1],
                                              self.V[next_state2],
                                              self.V[next_state3],
                                              self.V[next_state4]])]
                else:
                    return actions[np.random.randint(len(actions))]

            else:
                return actions[np.random.randint(len(actions))]
    def init_log(self):
        self.reward_log = []

    def log(self, reward):
        self.reward_log.append(reward)

    def show_reward_log(self,env, interval=25, episode=-1):
        if episode > 0:
            rewards = self.reward_log[-interval:]
            mean = np.round(np.mean(rewards), 3)
            std = np.round(np.std(rewards), 3)
            print('At Episode {} average steps is {} (+/-{})'
                  .format(episode, mean, std))

        else:
            indices = list(range(0, len(self.reward_log), interval))
            means = []
            stds = []
            for i in indices:
                rewards = self.reward_log[i:(i + interval)]
                means.append(np.mean(rewards))
                stds.append(np.std(rewards))
            means = np.array(means)
            stds = np.array(stds)
            plt.figure()
            plt.title('Step History')
            plt.grid()
            plt.fill_between(indices, means - stds, means + stds,
                             alpha=0.1, color='g')
            plt.plot(indices, means, 'o-', color='g',
                     label='The number of step for each {} episode'
                     .format(interval))
            plt.legend(loc='best')
            plt.savefig('Step History_{}.png'.format(self.epsilon))
            plt.show()
            K = np.zeros((env.row_length,env.column_length))
            for i in range(0,env.row_length):
                for j in range(0,env.column_length):
                    K[i,j] = decimal.Decimal(self.V[State(i,j)])
                    .quantize(decimal.Decimal("0.001"))
            print(K)
class TD(ELAgent):

    def __init__(self, epsilon=0.5):
        super().__init__(epsilon)

    def learn(self, env, episode_count, gamma=0.95, learning_rate=0.05,
```

```
              report_interval=50):
        self.init_log()
        self.V = defaultdict(lambda: 0)
        actions = list(range(4))
        for e in range(episode_count):
            s = env.reset()
            done = False
            count = 0
            while not done:
                can_actions = env.can_action(s, actions)
                a = self.policy(env, s, can_actions)
                n_state, reward, done = env.step(a)
                gain = reward + gamma * (self.V[n_state])
                estimated = self.V[s]
                self.V[s] += learning_rate * (gain - estimated)
                s = n_state
                count += 1

            else:
                self.log(count)

            if e != 0 and e % report_interval == 0:
                self.show_reward_log(env,episode=e)


def train():
#     grid = [
#     [9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9],
#     [9, 0, 0, 9, 0, 0, 0, 0, 0, 0, 0, 9],
#     [9, 0, 0, 9, 0, 9, 0, 0, 9, 9, 0, 9],
#     [9, 9, 0, 9, 0, 9, 9, 9, 9, 0, 0, 9],
#     [9, 0, 0, 0, 0, 0, 9, 0, 0, 0, 0, 9],
#     [9, 0, 0, 0, 0, 0, 9, 0, 0, 0, 0, 9],
#     [9, 9, 9, 9, 9, 0, 9, 0, 9, 0, 0, 9],
#     [9, 0, 0, 0, 0, 0, 9, 0, 9, 0, 0, 9],
#     [9, 0, 9, 9, 9, 9, 9, 0, 9, 0, 0, 9],
#     [9, 0, 0, 0, 0, 9, 0, 0, 9, 0, 0, 9],
#     [9, 0, 0, 9, 0, 0, 0, 0, 9, 0, -1, 9],
#     [9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9]]
    grid = [
    [9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9],
    [9, 0, 0, 9, 0, 0, 0, 0, 0, 9, 0, 0, 9, 0, 0, 0, 0, 9, 0, 0, 0, 0, 9],
    [9, 0, 0, 9, 0, 0, 9, 0, 0, 0, 0, 0, 0, 0, 0, 9, 0, 9, 0, 0, 9, 0, 9],
    [9, 9, 0, 9, 0, 0, 9, 9, 9, 9, 9, 9, 9, 9, 0, 9, 0, 9, 0, 0, 9, 0, 9],
    [9, 0, 0, 0, 0, 0, 0, 0, 9, 0, 0, 0, 0, 0, 0, 9, 0, 9, 0, 0, 9, 0, 9],
    [9, 0, 9, 9, 9, 9, 0, 0, 9, 0, 0, 0, 0, 0, 0, 9, 0, 0, 0, 0, 0, 0, 9],
    [9, 0, 0, 0, 0, 9, 9, 0, 9, 0, 0, 9, 9, 9, 9, 9, 9, 9, 9, 9, 0, 9, 9],
    [9, 0, 0, 0, 0, 9, 0, 0, 9, 0, 0, 0, 0, 9, 0, 0, 0, 0, 0, 0, 0, 0, 9],
    [9, 9, 9, 9, 9, 9, 0, 9, 9, 9, 0, 0, 0, 9, 0, 9, 9, 0, 0, 0, 0, 0, 9],
    [9, 0, 0, 9, 0, 0, 0, 0, 0, 0, 0, 9, 0, 9, 0, 9 ,9 ,0, 9, 9, 9, 9, 9 ,9],
    [9, 0, 0, 0, 0, 0, 9, 0, 0, 0, 0, 9, 0, 9, 0, 0, 0, 0, 0, 0, 0, 0, 9],
    [9, 0, 0, 9, 9, 9, 9, 9, 9, 9, 9, 9, 0, 0, 0, 0, 0, 9, 0, 9, 0, 0, 0, 9],
    [9, 0, 0, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 9, 9, 9, 9, 9, 9, 9, 9, 0, 0, 9],
    [9, 0, 0, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 9],
    [9, 0, 0, 9, 0, 0, 9, 9, 9, 9, 9, 9, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 9],
    [9, 0, 0, 9, 0, 0, 0, 0, 0, 0, 0, 0, 9, 9, 9, 9, 9, 0, 0, 0, 0, 0, 0, 9],
    [9, 0, 0, 0, 0, 0, 0, 0, 0, 9, 0, 0, 9, 0, 0, 0, 0, 0, 0, 0, 9, 9, 9, 9],
    [9, 0, 0, 0, 0, 0, 0, 0, 0, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 9, 9, 9, 9],
    [9, 0, 0, 0, 0, 0, 0, 0, 0, 9, 0, 0, 0, 0, 0, 9, 0, 0, 0, 0, 9, 9, 9, 9],
    [9, 0, 0, 0, 0, 0, 0, 9, 9, 9, 9, 9, 9, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 9],
    [9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 9],
```

```
    [9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 9, 0, 0, 0, 0, 0 ,0 ,0, 0, 0, 0, 0, 0, 9],
    [9, 9, 9, 9, 9, 9, 0, 0, 0, 0, 9, 0, 0, 0, 9, 0, 0, 9, 9, 9, 9, 9, 9, 9],
    [9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 9, 0, 0, 0, 9, 0, 0, 0, 0, 0, 0, 0, 0, 9],
    [9, 0, 0, 0, 0, 0, 0, 9, 9, 0, 0, 0, 0, 0, 9, 0 ,0 ,0, 9, 0, 0, 0, -1, 9],
    [9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9]
    ]


    env = Environment(grid)
    row_length = env.row_length
    column_length = env.column_length
    agent = TD()
    agent.learn(env, episode_count=500)
    agent.show_reward_log(env)

if __name__ == '__main__':
    train()
```

```
At Episode 50 average steps is 2257.2 (+/-1346.022)
At Episode 100 average steps is 137.6 (+/-59.668)
At Episode 150 average steps is 129.92 (+/-30.207)
At Episode 200 average steps is 116.4 (+/-21.518)
At Episode 250 average steps is 122.72 (+/-21.732)
At Episode 300 average steps is 128.96 (+/-26.297)
At Episode 350 average steps is 118.08 (+/-21.44)
At Episode 400 average steps is 120.0 (+/-27.788)
At Episode 450 average steps is 123.04 (+/-20.318)
```



(2)　Q 学習法アルゴリズム

```
import numpy as np
import matplotlib.pyplot as plt
from enum import Enum
from collections import defaultdict
import decimal

class State():

    def __init__(self, row=-1, column=-1):
        self.row = row
        self.column = column

    def __repr__(self):
        return "<State: [{}, {}]>".format(self.row, self.column)

    def clone(self):
        return State(self.row, self.column)

    def __hash__(self):
        return hash((self.row, self.column))

    def __eq__(self, other):
        return self.row == other.row and self.column == other.column


class Action(Enum):
    UP = 0
    DOWN = 1
    LEFT = 2
    RIGHT = 3

class Environment():

    def __init__(self, grid, move_prob=1.0):
        # grid is 2d-array. Its values are treated as an attribute.
        # Kinds of attribute is following.
        #  0: ordinary cell
        #  -1: damage cell (game end)
        #  1: reward cell (game end)
        #  9: block cell (can't locate agent)
        self.grid = grid
        self.agent_state = State()

        # Default reward is minus. Just like a poison swamp.
        # It means the agent has to reach the goal fast!
        self.default_reward = 0

        # Agent can move to a selected direction in move_prob.
        # It means the agent will move different direction
        # in (1 - move_prob).
        self.move_prob = move_prob
        self.reset()

    @property
    def row_length(self):
        return len(self.grid)

    @property
    def column_length(self):
```

```
    return len(self.grid[0])

@property
def actions(self):
    return [Action.UP, Action.DOWN,
            Action.LEFT, Action.RIGHT]

@property
def states(self):
    states = []
    for row in range(self.row_length):
        for column in range(self.column_length):
            # Block cells are not included to the state.
            if self.grid[row][column] != 9:
                states.append(State(row, column))
    return states


def can_action_at(self, state):
    if self.grid[state.row][state.column] == 0 or -1:
        return True
    else:
        return False

def can_action(self, state, actions):
    can_actions = []
    # Check whether the agent bumped a block cell.
    if self.grid[state.row -1][state.column] != 9:
        can_actions.append(0)
    if self.grid[state.row +1][state.column] != 9:
        can_actions.append(1)
    if self.grid[state.row][state.column -1] != 9:
        can_actions.append(2)
    if self.grid[state.row][state.column +1] != 9:
        can_actions.append(3)
    return can_actions


def _move(self, state, action):
    if not self.can_action_at(state):
        print(state.row, state.column)
        raise Exception("Can't move from here!")

    next_state = state.clone()

    # Execute an action (move).
    if action == 0:
        next_state.row -= 1
    elif action == 1:
        next_state.row += 1
    elif action == 2:
        next_state.column -= 1
    elif action == 3:
        next_state.column += 1

    # Check whether a state is out of the grid.
    if not (0 <= next_state.row < self.row_length):
        next_state = state
    if not (0 <= next_state.column < self.column_length):
        next_state = state
```

```
        # Check whether the agent bumped a block cell.
        if self.grid[next_state.row][next_state.column] == 9:
            next_state = state

        return next_state

    def reward_func(self, state):
        reward = self.default_reward
        done = False

        # Check an attribute of next state.
        attribute = self.grid[state.row][state.column]
        if attribute == -1:
            # Get reward! and the game ends.
            reward = 1
            done = True

        return reward, done

    def reset(self):
        # Locate the agent at lower left corner.
        self.agent_state = State(1, 1)
        return self.agent_state

    def step(self, action):
        next_state, reward, done = self.transit(self.agent_state, action)
        if next_state is not None:
            self.agent_state = next_state

        return next_state, reward, done

    def transit(self, state, action):
        next_state = self._move(state, action)
        reward, done = self.reward_func(next_state)
        return next_state, reward, done


class ELAgent():

    def __init__(self, epsilon):
        self.Q = {}
        self.epsilon = epsilon
        self.reward_log = []

    def policy(self, s, actions):
        if np.random.random() < self.epsilon:
            return actions[np.random.randint(len(actions))]
        else:
            if s in self.Q and sum(self.Q[s]) != 0:
                return np.argmax(self.Q[s])
            else:
                return actions[np.random.randint(len(actions))]

    def init_log(self):
        self.reward_log = []

    def log(self, reward):
        self.reward_log.append(reward)

    def show_reward_log(self,env,interval=20, episode=-1):
        if episode > 0:
```

```
            rewards = self.reward_log[-interval:]
            mean = np.round(np.mean(rewards), 3)
            std = np.round(np.std(rewards), 3)
            print('At Episode {} average steps is {} (+/-{})'
                  .format(episode, mean, std))

        else:

            indices = list(range(0, len(self.reward_log), interval))
            means = []
            stds = []
            for i in indices:
                rewards = self.reward_log[i:(i + interval)]
                means.append(np.mean(rewards))
                stds.append(np.std(rewards))
            means = np.array(means)
            stds = np.array(stds)
            plt.figure()
            plt.title('Step History')
            plt.grid()
            plt.fill_between(indices, means - stds, means + stds,
                             alpha=0.1, color='g')
            plt.plot(indices, means, 'o-', color='g',
                     label='The number of step for each {} episode'
                     .format(interval))
            plt.legend(loc='best')
            plt.savefig('Step History_{}.png'.format(self.epsilon))
            plt.show()
            K = np.zeros((env.row_length,env.column_length))
            for i in range(0,env.row_length):
                for j in range(0,env.column_length):
                    K[i,j] = decimal.Decimal(max(self.Q[State(i,j)]))
                    .quantize(decimal.Decimal("0.001"))
            print(K)

class QLearningAgent(ELAgent):

    def __init__(self, epsilon=0.5):
        super().__init__(epsilon)

    def learn(self, env, episode_count=500, gamma=0.95,
              learning_rate=0.05, report_interval=50):

        self.init_log()
        self.Q = defaultdict(lambda: [0] * len(actions))
        actions = list(range(4))
        for e in range(episode_count):
            s = env.reset()
            done = False
            count = 0
            while not done:
                can_actions = env.can_action(s, actions)
                a = self.policy(s, can_actions)
                n_state, reward, done = env.step(a)
                gain = reward + gamma * (max(self.Q[n_state]))
                estimated = self.Q[s][a]
                self.Q[s][a] += learning_rate * (gain - estimated)
                s = n_state
                count += 1
```

```
            else:
                self.log(count)

            if e != 0 and e % report_interval == 0:
                self.show_reward_log(env,episode=e)


def train():
#     grid = [
#     [9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9],
#     [9, 0, 0, 9, 0, 0, 0, 0, 0, 0, 0, 9],
#     [9, 0, 0, 9, 0, 9, 0, 0, 9, 9, 0, 9],
#     [9, 9, 0, 9, 0, 9, 9, 9, 9, 0, 0, 9],
#     [9, 0, 0, 0, 0, 0, 9, 0, 0, 0, 0, 9],
#     [9, 0, 0, 0, 0, 0, 9, 0, 0, 0, 0, 9],
#     [9, 9, 9, 9, 9, 0, 9, 0, 9, 0, 0, 9],
#     [9, 0, 0, 0, 0, 0, 9, 0, 9, 0, 0, 9],
#     [9, 0, 9, 9, 9, 9, 9, 0, 9, 0, 0, 9],
#     [9, 0, 0, 0, 0, 9, 0, 0, 9, 0, 0, 9],
#     [9, 0, 0, 9, 0, 0, 0, 0, 9, 0, -1, 9],
#     [9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9]]
    grid = [
    [9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9],
    [9, 0, 0, 9, 0, 0, 0, 0, 9, 0, 0, 9, 0, 0, 0, 0, 9, 0, 0, 0, 0, 9],
    [9, 0, 0, 9, 0, 0, 9, 0, 0, 0, 0, 0, 0, 0, 9, 0, 9, 0, 0, 9, 0, 9],
    [9, 9, 0, 9, 0, 0, 9, 9, 9, 9, 9, 9, 9, 9, 0, 9, 0, 9, 0, 0, 9, 9],
    [9, 0, 0, 0, 0, 0, 0, 0, 9, 0, 0, 0, 0, 0, 0, 9, 0, 9, 0, 0, 9, 9],
    [9, 0, 9, 9, 9, 9, 0, 0, 9, 0, 0, 0, 0, 0, 0, 9, 0, 0, 0, 0, 0, 9],
    [9, 0, 0, 0, 0, 9, 9, 0, 9, 0, 0, 9, 9, 9, 9, 9, 9, 9, 9, 0, 9, 9],
    [9, 0, 0, 0, 0, 9, 0, 0, 9, 0, 0, 0, 0, 9, 0, 0, 0, 0, 0, 0, 0, 9],
    [9, 9, 9, 9, 9, 9, 0, 9, 9, 9, 0, 0, 0, 9, 0, 9, 9, 0, 0, 0, 0, 9],
    [9, 0, 0, 9, 0, 0, 0, 0, 0, 0, 0, 9, 0, 9, 0, 9 ,9 ,0, 9, 9, 9, 9, 9 ,9],
    [9, 0, 0, 0, 0, 0, 9, 0, 0, 0, 0, 9, 0, 9, 0, 0, 0, 0, 0, 0, 0, 0, 9],
    [9, 0, 0, 9, 9, 9, 9, 9, 9, 9, 9, 9, 0, 0, 0, 0, 9, 0, 9, 0, 0, 0, 9],
    [9, 0, 0, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 9, 9, 9, 9, 9, 9, 9, 0, 0, 9],
    [9, 0, 0, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 9],
    [9, 0, 0, 9, 0, 0, 9, 9, 9, 9, 9, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 9],
    [9, 0, 0, 9, 0, 0, 0, 0, 0, 0, 0, 9, 9, 9, 9, 9, 0, 0, 0, 0, 0, 0, 9],
    [9, 0, 0, 0, 0, 0, 0, 0, 9, 0, 0, 9, 0, 0, 0, 0, 0, 0, 9, 9, 9, 0, 0, 9],
    [9, 0, 0, 0, 0, 0, 0, 0, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 9, 9, 9, 0, 0, 9],
    [9, 0, 0, 0, 0, 0, 0, 0, 9, 0, 0, 0, 0, 9, 0, 0, 0, 0, 9, 9, 9, 0, 0, 9],
    [9, 0, 0, 0, 0, 0, 0, 9, 9, 9, 9, 9, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 9],
    [9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 9],
    [9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 9, 0, 0, 0, 0, 0 ,0 ,0, 0, 0, 0, 0, 9],
    [9, 9, 9, 9, 9, 9, 0, 0, 0, 0, 9, 0, 0, 0, 9, 0, 0, 9, 9, 9, 9, 9, 9],
    [9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 9, 0, 0, 0, 9, 0, 0, 0, 0, 0, 0, 0, 9],
    [9, 0, 0, 0, 0, 0, 0, 9, 9, 0, 0, 0, 0, 0, 9, 0 ,0 ,0, 9, 0, 0, 0, -1, 9],
    [9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9]]


    env = Environment(grid)
    agent = QLearningAgent()
    agent.learn(env, episode_count=500)
    agent.show_reward_log(env)

if __name__ == '__main__':
    train()
```

At Episode 50 average steps is 1858.3 (+/-905.587)

```
At Episode 100 average steps is 219.5 (+/-231.872)
At Episode 150 average steps is 117.2 (+/-21.141)
At Episode 200 average steps is 108.6 (+/-16.806)
At Episode 250 average steps is 103.4 (+/-12.978)
At Episode 300 average steps is 102.8 (+/-21.056)
At Episode 350 average steps is 111.0 (+/-16.426)
At Episode 400 average steps is 113.2 (+/-14.905)
At Episode 450 average steps is 106.1 (+/-19.468)
```