Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления и искусственный интеллект

КАФЕДРА Системы обработки информации и управления

Лабораторная работа №5 по курсу «Методы машинного обучения в автоматизированных системах обработки информации и управления»

Подготовили:

У Жун

ИУ5И-25М

30.05.2024

Проверил:

Гапанюк Ю. Е.

2024 г.

Цель лабораторной работы:

ознакомление с базовыми методами обучения с подкреплением на основе временных различий.

Задание:

На основе рассмотренного на лекции примера реализуйте следующие алгоритмы:

- SARSA
- Q-обучение
- Двойное Q-обучение

для любой среды обучения с подкреплением (кроме рассмотренной на лекции среды Toy Text / Frozen Lake) из библиотеки Gym (или аналогичной библиотеки).

Для реализации алгоритмов SARSA, Q-learning и двойного Q-learning используется среда Taxi-v3 из библиотеки Gym. В этой задаче агенту необходимо управлять такси, чтобы забрать пассажира и доставить его в нужное место.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import gym
from tqdm import tqdm

class BasicAgent:
    ALGO_NAME = '---'

    def __init__(self, env, eps=0.1):
        self.env = env
        self.nA = env.action_space.n
        self.nS = env.observation_space.n
        self.Q = np.zeros((self.nS, self.nA))
        self.eps = eps
        self.episodes_reward = []

    def print_q(self):
        print(f'Q-матрица для алгоритма {self.ALGO_NAME}')
        print(self.Q)

    def get_state(self, state):
        return state

    def greedy(self, state):
        return np.argmax(self.Q[state])
```

```
    def greedy(self, state):
        return np.argmax(self.Q[state])

    def make_action(self, state):
        if np.random.uniform(0, 1) < self.eps:
            return self.env.action_space.sample()
        else:
            return self.greedy(state)

    def draw_episodes_reward(self):
        fig, ax = plt.subplots(figsize=(15, 10))
        plt.plot(range(1, len(self.episodes_reward) + 1), self.episodes_reward, '-', linewidth=1, color='green')
        plt.title('Награды по эпизодам')
        plt.xlabel('Номер эпизода')
        plt.ylabel('Награда')
        plt.show()

    def learn(self):
        pass

class SARSA_Agent(BasicAgent):
    ALGO_NAME = 'SARSA'

    def __init__(self, env, eps=0.4, lr=0.1, gamma=0.98, num_episodes=20000):
        super().__init__(env, eps)
        self.lr = lr
        self.gamma = gamma
        self.num_episodes = num_episodes
        self.eps_decay = 0.00005
        self.eps_threshold = 0.01
```

```
        self.eps_threshold = 0.01

    def learn(self):
        self.episodes_reward = []
        for ep in tqdm(range(self.num_episodes)):
            state = self.get_state(self.env.reset()[0])
            done = False
            tot_rew = 0
            if self.eps > self.eps_threshold:
                self.eps -= self.eps_decay
            action = self.make_action(state)
            while not done:
                next_state, rew, done, truncated, _ = self.env.step(action)
                next_action = self.make_action(next_state)
                self.Q[state][action] += self.lr * (rew + self.gamma * self.Q[next_state][next_action] - self.Q[state][action])
                state, action = next_state, next_action
                tot_rew += rew
                if done or truncated:
                    self.episodes_reward.append(tot_rew)

class QLearning_Agent(BasicAgent):
    ALGO_NAME = 'Q-обучение'

    def __init__(self, env, eps=0.4, lr=0.1, gamma=0.98, num_episodes=20000):
        super().__init__(env, eps)
        self.lr = lr
        self.gamma = gamma
        self.num_episodes = num_episodes
        self.eps_decay = 0.00005
        self.eps_threshold = 0.01

    def learn(self):
        self.episodes_reward = []
```

```
    def learn(self):
        self.episodes_reward = []
        for ep in tqdm(range(self.num_episodes)):
            state = self.get_state(self.env.reset()[0])
            done = False
            tot_rew = 0
            if self.eps > self.eps_threshold:
                self.eps -= self.eps_decay
            while not done:
                action = self.make_action(state)
                next_state, rew, done, truncated, _ = self.env.step(action)
                self.Q[state][action] += self.lr * (rew + self.gamma * np.max(self.Q[next_state]) - self.Q[state][action])
                state = next_state
                tot_rew += rew
                if done or truncated:
                    self.episodes_reward.append(tot_rew)

class DoubleQLearning_Agent(BasicAgent):
    ALGO_NAME = 'Двойное Q-обучение'

    def __init__(self, env, eps=0.4, lr=0.1, gamma=0.98, num_episodes=20000):
        super().__init__(env, eps)
        self.Q2 = np.zeros((self.nS, self.nA))
        self.lr = lr
        self.gamma = gamma
        self.num_episodes = num_episodes
        self.eps_decay = 0.00005
        self.eps_threshold = 0.01

    def greedy(self, state):
        temp_q = self.Q[state] + self.Q2[state]
        return np.argmax(temp_q)

    def print_q(self):
        print(f'Q-матрицы для алгоритма {self.ALGO_NAME}')
        print('Q1')
```

```
        print(f'Q-матрицы для алгоритма {self.ALGO_NAME}')
        print('Q1')
        print(self.Q)
        print('Q2')
        print(self.Q2)

    def learn(self):
        self.episodes_reward = []
        for ep in tqdm(range(self.num_episodes)):
            state = self.get_state(self.env.reset()[0])
            done = False
            tot_rew = 0
            if self.eps > self.eps_threshold:
                self.eps -= self.eps_decay
            while not done:
                action = self.make_action(state)
                next_state, rew, done, truncated, _ = self.env.step(action)
                if np.random.rand() < 0.5:
                    self.Q[state][action] += self.lr * (rew + self.gamma * self.Q2[next_state][np.argmax(self.Q[next_state])] - self.Q[state
                else:
                    self.Q2[state][action] += self.lr * (rew + self.gamma * self.Q[next_state][np.argmax(self.Q2[next_state])] - self.Q2[sta
                state = next_state
                tot_rew += rew
                if done or truncated:
                    self.episodes_reward.append(tot_rew)

def play_agent(agent):
    env2 = gym.make('Taxi-v3', render_mode='human')
    state = env2.reset()[0]
    done = False
    while not done:
        action = agent.greedy(state)
        next_state, reward, done, truncated, _ = env2.step(action)
        env2.render()
        state = next_state
        if done or truncated:
            break
```

```
        if done or truncated:
            break

def run_sarsa():
    env = gym.make('Taxi-v3')
    agent = SARSA_Agent(env)
    agent.learn()
    agent.print_q()
    agent.draw_episodes_reward()
    play_agent(agent)

def run_q_learning():
    env = gym.make('Taxi-v3')
    agent = QLearning_Agent(env)
    agent.learn()
    agent.print_q()
    agent.draw_episodes_reward()
    play_agent(agent)

def run_double_q_learning():
    env = gym.make('Taxi-v3')
    agent = DoubleQLearning_Agent(env)
    agent.learn()
    agent.print_q()
    agent.draw_episodes_reward()
    play_agent(agent)

def main():
    #run_sarsa()
    #run_q_learning()
    run_double_q_learning()

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

Рис.1-Все коды работают

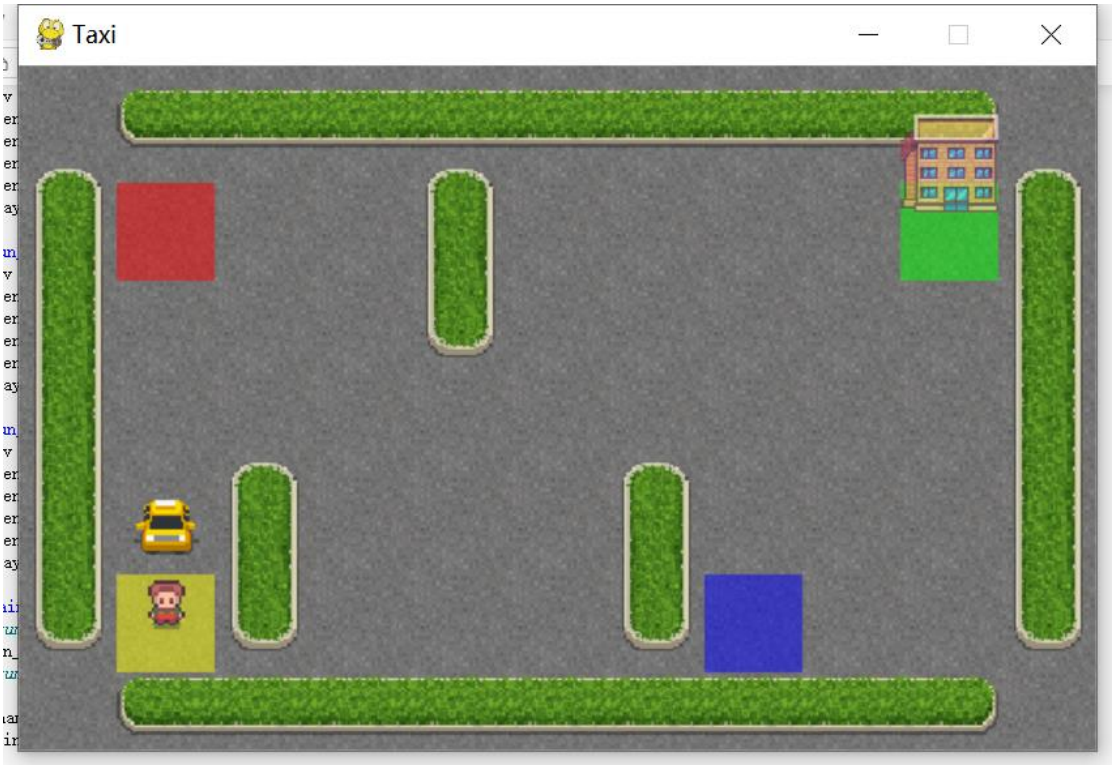

Рис.2-Окно визуализации среды Taxi-v3

- SARSA

```
100%|██████████████████████████████████████████████████████████████████████████████| 20000/20000 [00:12<00:00, 1606.61it/s]
Q-матрица для алгоритма SARSA
[[  0.           0.           0.           0.           0.
    0.        ]
 [ -8.30426087  -4.50474368  -9.69214165  -6.50334611   7.40548641
  -16.62622975]
 [  2.92782117   5.42360305   1.73069274   5.1982052   13.05638901
   -4.20996541]
 ...
 [ -1.38588597   6.47257128  -0.45817532  -3.02479818  -8.75915961
   -7.8242664 ]
 [ -7.75851792  -4.60606676  -7.81489247  -8.0382267   -9.61003258
  -13.11445746]
 [  4.4519614    2.18070971   8.60392222  18.47138532  -2.13989782
   -0.47628944]]
```

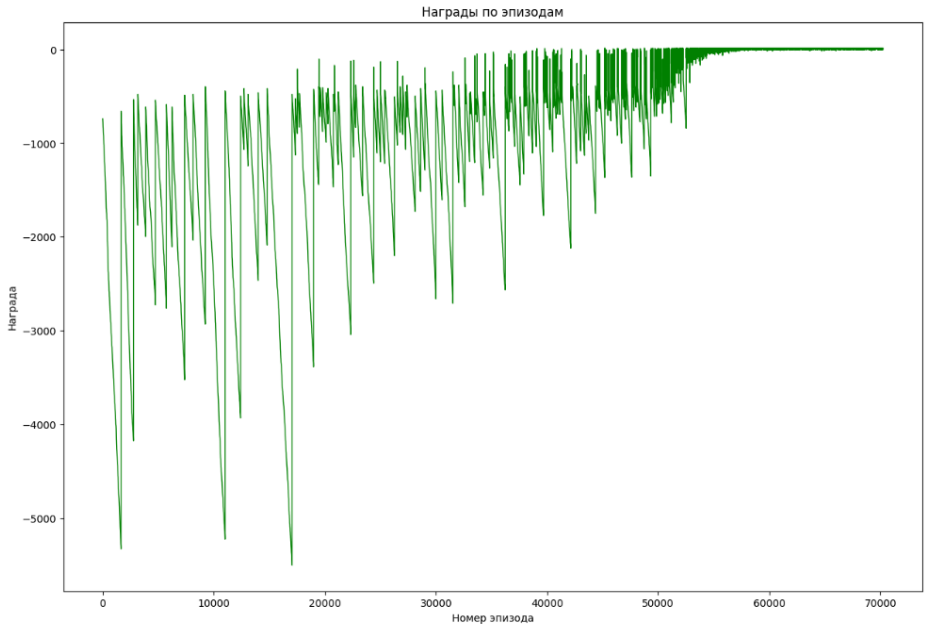

Рис.3-Результаты реализации алгоритма SARSA

- Q-обучение

```
100%|██████████████████████████████████████████████████████████████████████████████
████████| 20000/20000 [00:14<00:00, 1367.66it/s]

Q-матрица для алгоритма Q-обучение
[[ 0.          0.          0.          0.          0.          0.        ]
 [ 4.96362939  6.38760655  4.33626834  6.09444905  8.36234335 -3.01251887]
 [ 9.44057403 11.62228883  8.85346661 10.37003714 13.27445578  2.06417491]
 ...
 [-0.87160849 12.37459414 -1.30786827  0.14575665 -4.90665032 -2.77341095]
 [-2.14639888 -2.86818126 -2.89303068  8.54390491 -7.54060991 -9.35512287]
 [ 6.29379305  7.38478501  4.49111078 18.5994451   1.192372    1.67581754]]
```

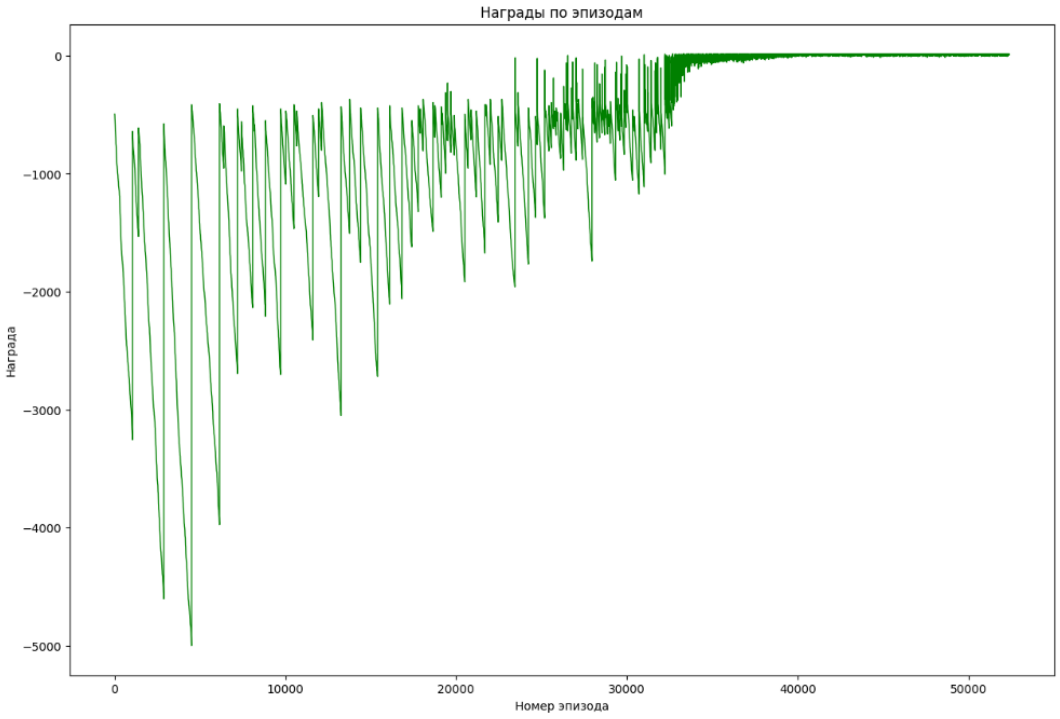

Рис.4-Результаты выполнения алгоритма Q-обучение

- Двойное Q-обучение

```
100%|██████████| 20000/20000 [00:14<00:00, 1422.62it/s]
Q-матрицы для алгоритма Двойное Q-обучение
Q1
[[ 0.          0.          0.          0.          0.          0.        ]
 [ 1.43194168  3.0376162  -1.8327853   3.59625275  8.36234335 -3.47517862]
 [ 4.99805284  7.02015831  5.23634041  8.80822548 13.27445578 -0.84147801]
 ...
 [ 9.53744541 14.5657712   7.51538604  8.01080848  1.91888404  2.02600944]
 [-3.79272421 -3.99108123 -2.80505413  6.86788473 -8.25407414 -9.21326756]
 [ 3.75821106  0.48188641  1.17797326 18.40083951 -0.41139316 -0.69750491]]
Q2
[[ 0.          0.          0.          0.          0.          0.        ]
 [ 1.68479303  1.62243662 -2.75703009  0.34418146  8.36234335 -5.83469708]
 [ 6.48566922  8.34782885  6.47356826  4.8550552  13.27445578 -1.19239052]
 ...
 [ 9.00936586 14.5657712  11.07960658  9.53503717  2.92695743  1.99359854]
 [-3.90574726 -3.46407079 -3.93980722  6.33760437 -5.94674487 -8.98061035]
 [ 3.7866738   3.97425705  1.58748565 18.23767934  0.75369346  1.155212  ]]
```

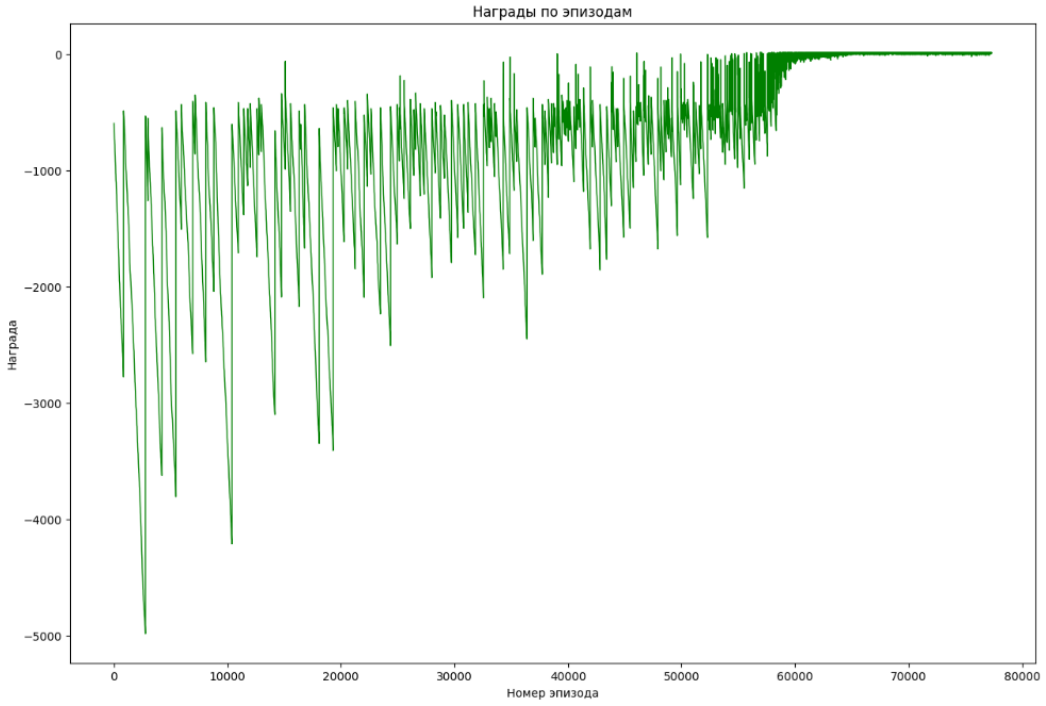

Рис.5-Результаты выполнения алгоритма Двойное Q-обучение