Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Московский государственный технический университет имени Н.Э. Баумана» (МГТУ им. Н.Э. Баумана)

Лабораторная работа №5
по курсу «Методы машинного обучения»

Выполнил
студент группы ИУ5-22М
XXXX

Москва, 2023

1. Задание

На основе рассмотренного на лекции примера реализуйте следующие алгоритмы:

- SARSA
- Q-обучение
- Двойное Q-обучение

2. Текст программы

```python
#!/usr/bin/env python

import numpy as np
import matplotlib.pyplot as plt
import gymnasium as gym
from tqdm import tqdm

# ********************************** БАЗОВЫЙ АГЕНТ ***************************************

class BasicAgent:
    '''
    Базовый агент, от которого наследуются стратегии обучения
    '''

    # Наименование алгоритма
    ALGO_NAME = '---'

    def __init__(self, env, eps=0.1):
        # Среда
        self.env = env
        # Размерности Q-матрицы
        self.nA = env.action_space.n
        self.nS = env.observation_space.n
        #и сама матрица
        self.Q = np.zeros((self.nS, self.nA))
        # Значения коэффициентов
        # Порог выбора случайного действия
        self.eps=eps
        # Награды по эпизодам
        self.episodes_reward = []

    def print_q(self):
        print('Вывод Q-матрицы для алгоритма ', self.ALGO_NAME)
        print(self.Q)

    def get_state(self, state):
        '''
        Возвращает правильное начальное состояние
        '''
        if type(state) is tuple:
            # Если состояние вернулось с виде кортежа, то вернуть только номер состояния
            return state[0]
        else:
            return state

    def greedy(self, state):
```

```
        '''
        <<Жадное>> текущее действие
        Возвращает действие, соответствующее максимальному Q-значению
        для состояния state
        '''
        return np.argmax(self.Q[state])

    def make_action(self, state):
        '''
        Выбор действия агентом
        '''
        if np.random.uniform(0,1) < self.eps:

            # Если вероятность меньше eps
            # то выбирается случайное действие
            return self.env.action_space.sample()
        else:
            # иначе действие, соответствующее максимальному Q-значению
            return self.greedy(state)

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

    def learn():
        '''
        Реализация алгоритма обучения
        '''
        pass

# ********************************** SARSA ****************************************

class SARSA_Agent(BasicAgent):
    '''
    Реализация алгоритма SARSA
    '''
    # Наименование алгоритма
    ALGO_NAME = 'SARSA'

    def __init__(self, env, eps=0.4, lr=0.1, gamma=0.98, num_episodes=20000):
        # Вызов конструктора верхнего уровня
        super().__init__(env, eps)
        # Learning rate
        self.lr=lr
        # Коэффициент дисконтирования
        self.gamma = gamma
        # Количество эпизодов
        self.num_episodes=num_episodes
        # Постепенное уменьшение eps
        self.eps_decay=0.00005
        self.eps_threshold=0.01

    def learn(self):
        '''
```

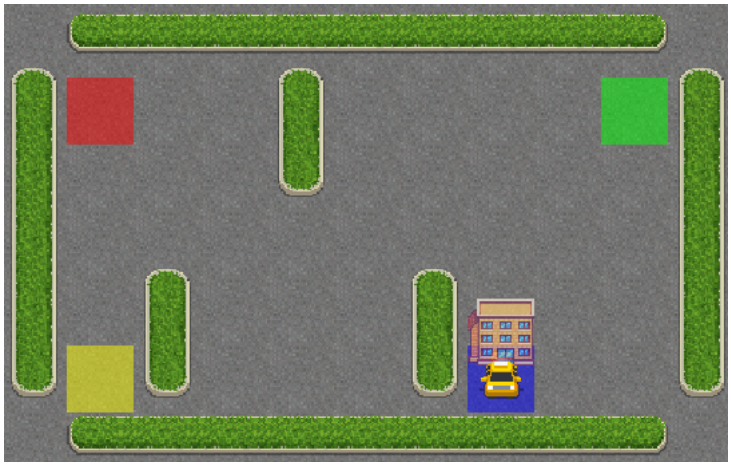
```
        Обучение на основе алгоритма SARSA
        '''
        self.episodes_reward = []
        # Цикл по эпизодам
        for ep in tqdm(list(range(self.num_episodes))):
            # Начальное состояние среды
            state = self.get_state(self.env.reset())
            # Флаг штатного завершения эпизода
            done = False
            # Флаг нештатного завершения эпизода
            truncated = False
            # Суммарная награда по эпизоду
            tot_rew = 0

            # По мере заполнения Q-матрицы уменьшаем вероятность случайного выбора действия
            if self.eps > self.eps_threshold:
                self.eps -= self.eps_decay

            # Выбор действия
            action = self.make_action(state)

            # Проигрывание одного эпизода до финального состояния
            while not (done or truncated):
                # Выполняем шаг в среде
                next_state, rew, done, truncated, _ = self.env.step(action)

                # Выполняем следующее действие
                next_action = self.make_action(next_state)

                # Правило обновления Q для SARSA
                self.Q[state][action] = self.Q[state][action] + self.lr * \
                    (rew + self.gamma * self.Q[next_state][next_action] -
                    self.Q[state][action])

                # Следующее состояние считаем текущим
                state = next_state
                action = next_action
                # Суммарная награда за эпизод
                tot_rew += rew
                if (done or truncated):
                    self.episodes_reward.append(tot_rew)

# *********************************** Q-обучение ****************************************

class QLearning_Agent(BasicAgent):
    '''
    Реализация алгоритма Q-Learning
    '''
    # Наименование алгоритма
    ALGO_NAME = 'Q-обучение'

    def __init__(self, env, eps=0.4, lr=0.1, gamma=0.98, num_episodes=20000):
        # Вызов конструктора верхнего уровня
        super().__init__(env, eps)
        # Learning rate
        self.lr=lr
        # Коэффициент дисконтирования
        self.gamma = gamma
        # Количество эпизодов
        self.num_episodes=num_episodes
        # Постепенное уменьшение eps
```

```
168         self.eps_decay=0.00005
169         self.eps_threshold=0.01
170 
171     def learn(self):
172         '''
173         Обучение на основе алгоритма Q-Learning
174         '''
175         self.episodes_reward = []
176         # Цикл по эпизодам
177         for ep in tqdm(list(range(self.num_episodes))):
178             # Начальное состояние среды
179             state = self.get_state(self.env.reset())
180             # Флаг штатного завершения эпизода
181             done = False
182             # Флаг нештатного завершения эпизода
183             truncated = False
184             # Суммарная награда по эпизоду
185             tot_rew = 0
186 
187             # По мере заполнения Q-матрицы уменьшаем вероятность случайного выбора действия
188             if self.eps > self.eps_threshold:
189                 self.eps -= self.eps_decay
190 
191             # Проигрывание одного эпизода до финального состояния
192             while not (done or truncated):
193                 # Выбор действия
194                 # В SARSA следующее действие выбиралось после шага в среде
195                 action = self.make_action(state)
196 
197                 # Выполняем шаг в среде
198                 next_state, rew, done, truncated, _ = self.env.step(action)
199 
200                 # Правило обновления Q для SARSA (для сравнения)
201                 # self.Q[state][action] = self.Q[state][action] + self.lr * \
202                 #     (rew + self.gamma * self.Q[next_state][next_action] -
                    self.Q[state][action])
203 
204                 # Правило обновления для Q-обучения
205                 self.Q[state][action] = self.Q[state][action] + self.lr * \
206                     (rew + self.gamma * np.max(self.Q[next_state]) - self.Q[state][action])
207 
208                 # Следующее состояние считаем текущим
209                 state = next_state
210                 # Суммарная награда за эпизод
211                 tot_rew += rew
212                 if (done or truncated):
213                     self.episodes_reward.append(tot_rew)
214 
215 # ********************************** Двойное Q-обучение
    ******************************************
216 
217 class DoubleQLearning_Agent(BasicAgent):
218     '''
219     Реализация алгоритма Double Q-Learning
220     '''
221     # Наименование алгоритма
222     ALGO_NAME = 'Двойное Q-обучение'
223 
224     def __init__(self, env, eps=0.4, lr=0.1, gamma=0.98, num_episodes=20000):
225         # Вызов конструктора верхнего уровня
226         super().__init__(env, eps)
```

```
227         # Вторая матрица
228         self.Q2 = np.zeros((self.nS, self.nA))
229         # Learning rate
230         self.lr=lr
231         # Коэффициент дисконтирования
232         self.gamma = gamma
233         # Количество эпизодов
234         self.num_episodes=num_episodes
235         # Постепенное уменьшение eps
236         self.eps_decay=0.00005
237         self.eps_threshold=0.01
238 
239 
240     def greedy(self, state):
241         '''
242         <<Жадное>> текущее действие
243         Возвращает действие, соответствующее максимальному Q-значению
244         для состояния state
245         '''
246         temp_q = self.Q[state] + self.Q2[state]
247         return np.argmax(temp_q)
248 
249     def print_q(self):
250         print(f"Вывод Q-матриц для алгоритма {self.ALGO_NAME}")
251         print('Q1')
252         print(self.Q)
253         print('Q2')
254         print(self.Q2)
255 
256     def learn(self):
257         '''
258         Обучение на основе алгоритма Double Q-Learning
259         '''
260         self.episodes_reward = []
261         # Цикл по эпизодам
262         for ep in tqdm(list(range(self.num_episodes))):
263             # Начальное состояние среды
264             state = self.get_state(self.env.reset())
265             # Флаг штатного завершения эпизода
266             done = False
267             # Флаг нештатного завершения эпизода
268             truncated = False
269             # Суммарная награда по эпизоду
270             tot_rew = 0
271 
272             # По мере заполнения Q-матрицы уменьшаем вероятность случайного выбора действия
273             if self.eps > self.eps_threshold:
274                 self.eps -= self.eps_decay
275 
276             # Проигрывание одного эпизода до финального состояния
277             while not (done or truncated):
278                 # Выбор действия
279                 # В SARSA следующее действие выбиралось после шага в среде
280                 action = self.make_action(state)
281 
282                 # Выполняем шаг в среде
283                 next_state, rew, done, truncated, _ = self.env.step(action)
284 
285                 if np.random.rand() < 0.5:
286                     # Обновление первой таблицы
287                     self.Q[state][action] = self.Q[state][action] + self.lr * \
```

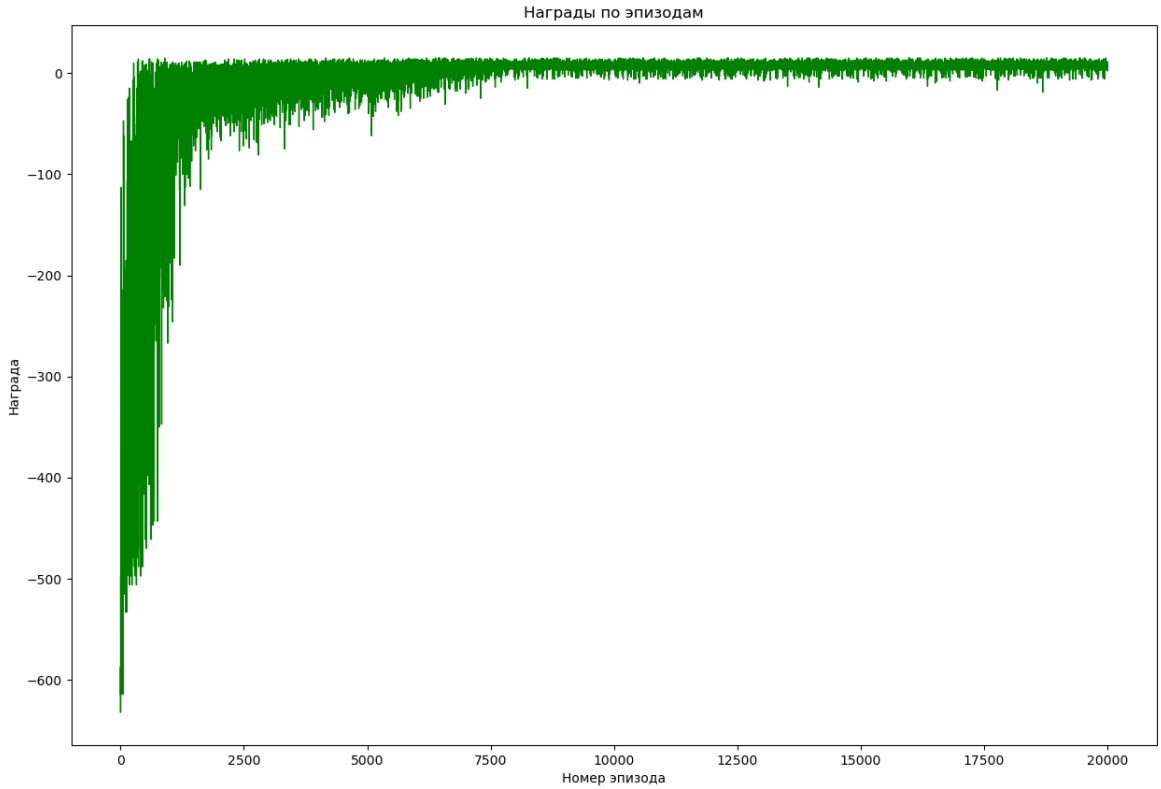
```
                        (rew + self.gamma * self.Q2[next_state][np.argmax(self.Q[next_state])]
                        - self.Q[state][action])
                else:
                    # Обновление второй таблицы
                    self.Q2[state][action] = self.Q2[state][action] + self.lr * \
                        (rew + self.gamma * self.Q[next_state][np.argmax(self.Q2[next_state])]
                        - self.Q2[state][action])

                # Следующее состояние считаем текущим
                state = next_state
                # Суммарная награда за эпизод
                tot_rew += rew
                if (done or truncated):
                    self.episodes_reward.append(tot_rew)

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

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()
```

3. Экранные формы с примерами выполнения программы



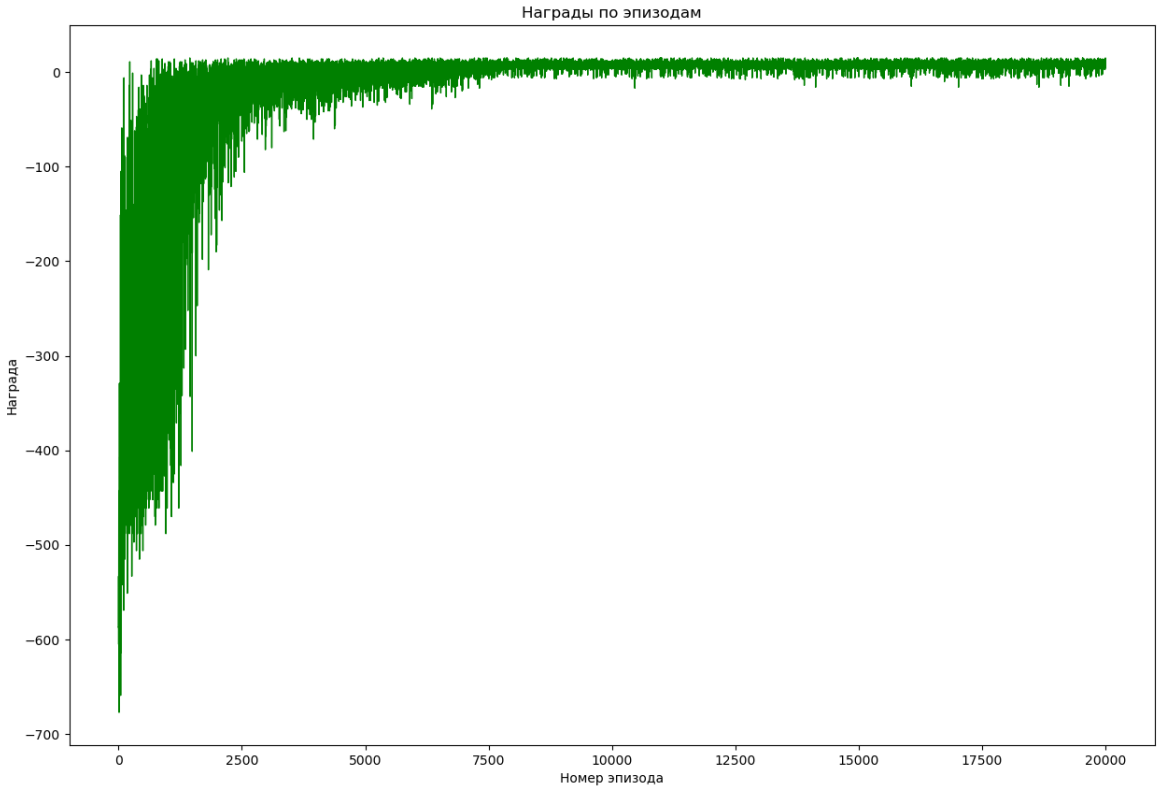
Q-обучение: eps=0.4, lr=0.1, gamma=0.98, num_episodes=20000

```
Вывод Q-матрицы для алгоритма  Q-обучение
[[ 0.          0.          0.          0.          0.          0.        ]
 [ 5.09940217  5.11613495  4.9333549   6.24799219  8.36234335 -3.62258081]
 [10.04299356 11.11574742  9.6393241  11.43318009 13.27445578  2.65833957]
 ...
 [-1.36364673 11.89866367  0.31299649  0.12111493 -2.18307592 -4.64070247]
 [-2.6539698   8.34882836 -2.02398777  0.61115667 -8.4092672  -7.78482003]
 [11.04932363  4.15528476  7.05358754 18.59990519  2.11145862  4.24396103]]
```



Двойное Q-обучение: eps=0.4, lr=0.1, gamma=0.98, num_episodes=20000

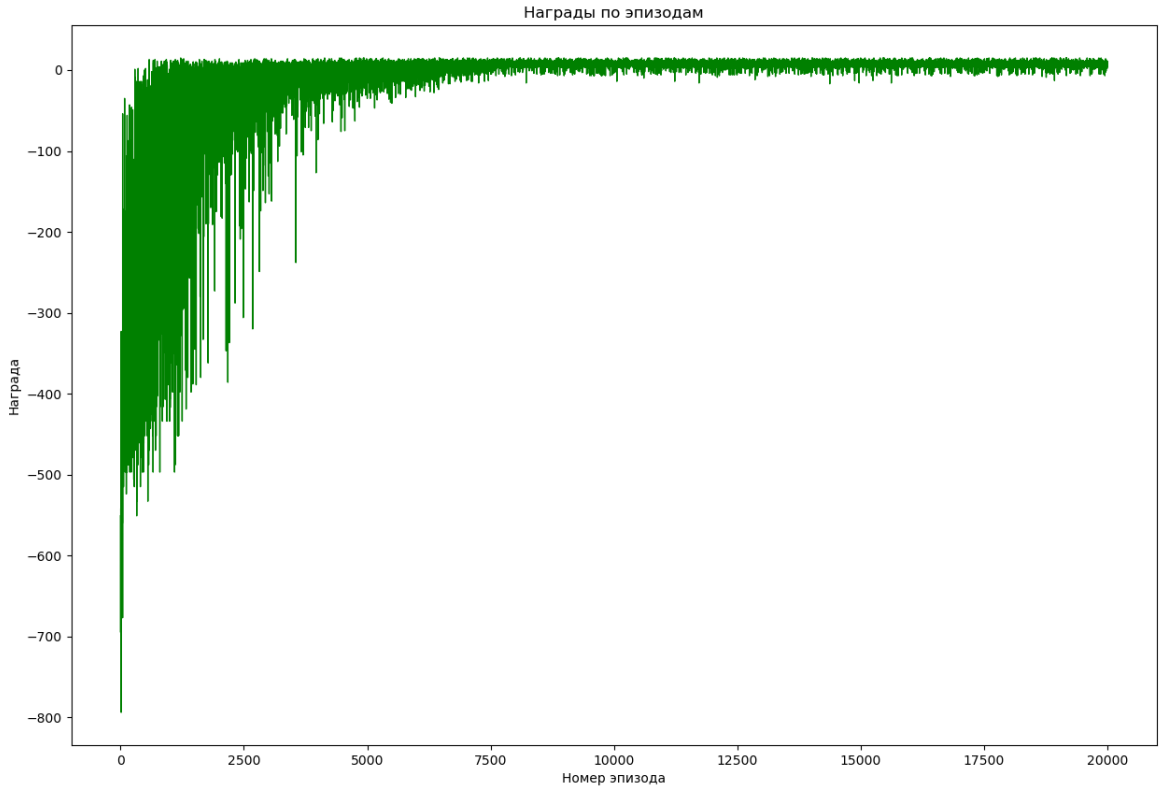
```
Вывод Q-матриц для алгоритма Двойное Q-обучение
Q1
[[ 0.          0.          0.          0.          0.          0.        ]
  [-0.72267154  1.85252442  0.19552651  2.85144428  8.36234335 -6.59195419]
  [ 7.18532871  6.59356075  5.37962679  5.5787384  13.27445578 -4.09009116]
  ...
  [-0.66467311 12.85534519 -1.95425954 -2.31569204 -5.48922472 -4.37459481]
  [-3.22093218 -3.91954695 -3.58949805  1.68910615 -7.61442506 -8.67036163]
  [ 3.79014035  3.88911184  5.13464853 18.25507162  0.68488221  1.74249151]]
Q2
[[ 0.          0.          0.          0.          0.          0.        ]
  [-0.77584626  3.15691246  0.69250749  2.73240034  8.36234335 -5.35095443]
  [ 4.20528463  7.68270196  4.62451724  3.56948691 13.27445578 -1.14602156]
  ...
  [-1.54713418 13.67037706  0.3485602  -0.64388233 -7.43964304 -4.11504528]
  [-4.99805617 -4.01629887 -3.96321961  2.922531   -4.97259003 -3.3258903 ]
  [ 2.08418592  0.94533048  1.13723617 18.48774274 -1.7645767  -0.50591363]]
```



SARSA: eps=0.4, lr=0.1, gamma=0.98, num_episodes=20000

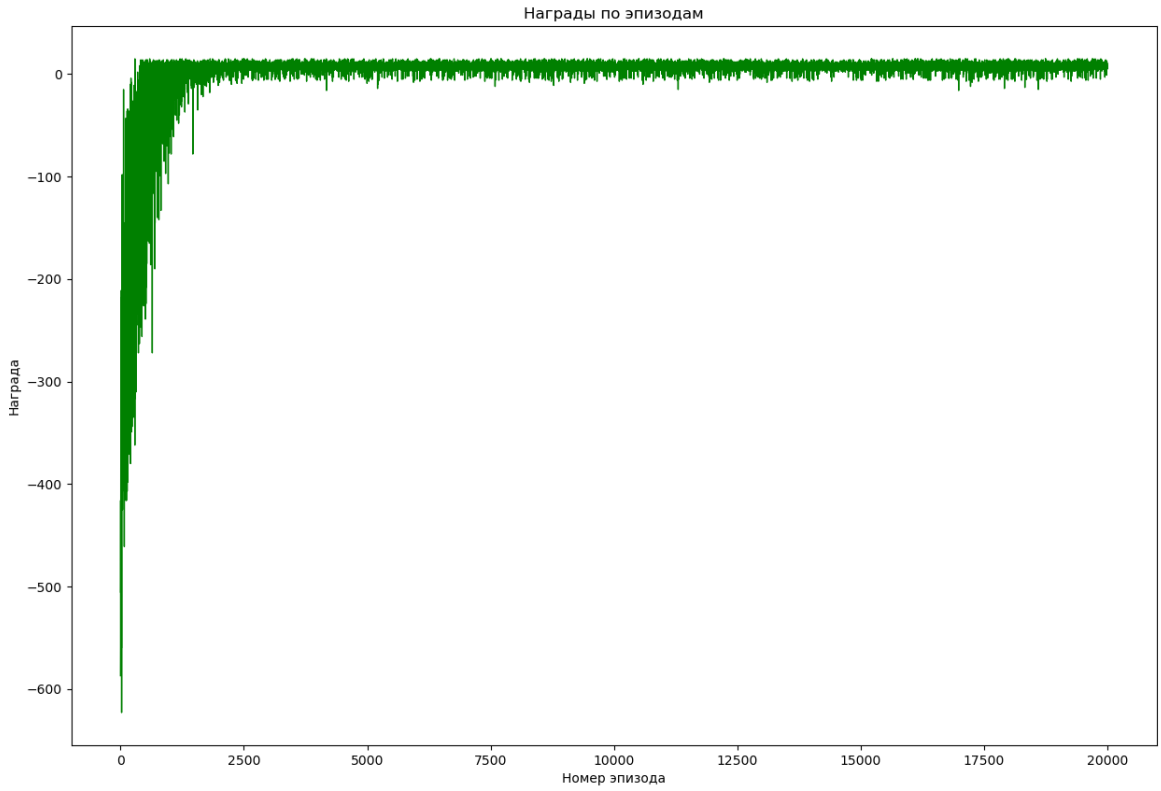
```
Вывод Q-матрицы для алгоритма  SARSA
[[  0.           0.           0.           0.           0.
    0.        ]
 [ -8.66665356  -5.19580592  -3.60445722  -2.81367279   7.94897997
   -10.02695965]
 [ -0.13055387  -0.3346179    0.34120444   5.34029633  13.19676101
    -3.77660174]
  ...
```

```
[ -0.65210655  10.87574541  -2.27322581  -2.06382686  -7.35227183
   -4.84737165]
[ -8.37712909  -4.26957472  -8.15637214  -7.89962417 -14.04817395
  -13.32447119]
[  8.81479907   7.35859204   6.83248527  18.47612868  -0.1078542
    0.98660318]]
```



4. Вывод

На параметрах по умолчанию быстрее всего сходится Q-обучение. Gamma подобрана удачно, крупные изменения значения делают результат хуже. Быстрая сходимость получается на eps = 0.1:



SARSA также быстрее сходится при уменьшении eps. Learning rate и gamma подобраны удачно.