Андреев А.В ИУ5-25М

Рубежный контроль №2

Задание: Для одного из алгоритмов временных различий, реализованных Вами в соответствующей лабораторная работе: SARSA осуществите подбор гиперпараметров. Критерием оптимизации должна являться суммарная награда.

In [3]:

```
! pip install gymnasium
import numpy as np
import matplotlib.pyplot as plt
import gymnasium as gym
from tqdm import tqdm
import matplotlib
# matplotlib.use('TkAgg')
```

```
Looking in indexes: https://pypi.org/simple, (https://pypi.org/simple,) ht
tps://us-python.pkg.dev/colab-wheels/public/simple/ (https://us-python.pk
g.dev/colab-wheels/public/simple/)
Requirement already satisfied: gymnasium in /usr/local/lib/python3.10/dist
-packages (0.28.1)
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.10/
dist-packages (from gymnasium) (1.22.4)
Requirement already satisfied: jax-jumpy>=1.0.0 in /usr/local/lib/python3.
10/dist-packages (from gymnasium) (1.0.0)
Requirement already satisfied: cloudpickle>=1.2.0 in /usr/local/lib/python
3.10/dist-packages (from gymnasium) (2.2.1)
Requirement already satisfied: typing-extensions>=4.3.0 in /usr/local/lib/
python3.10/dist-packages (from gymnasium) (4.5.0)
Requirement already satisfied: farama-notifications>=0.0.1 in /usr/local/l
ib/python3.10/dist-packages (from gymnasium) (0.0.4)
```

In [4]:

```
!pip install pygame

import os
os.environ['SDL_VIDEODRIVER']='dummy'
import pygame
pygame.display.set_mode((640,480))
```

```
Looking in indexes: https://pypi.org/simple, (https://pypi.org/simple,) ht
tps://us-python.pkg.dev/colab-wheels/public/simple/ (https://us-python.pk
g.dev/colab-wheels/public/simple/)
Requirement already satisfied: pygame in /usr/local/lib/python3.10/dist-pa
ckages (2.3.0)
```

Out[4]:

```
<Surface(640x480x32 SW)>
```

In [5]:

```
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
```

In [6]:

```
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 действ
            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][a

                # 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)
```

In [7]:

```
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
        self.eps_decay=0.00005
        self.eps_threshold=0.01

    def learn(self):
        '''
        Обучение на основе алгоритма Q-Learning
        '''
        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 действ
            if self.eps > self.eps_threshold:
                self.eps -= self.eps_decay

            # Проигрывание одного эпизода до финального состояния
            while not (done or truncated):
                # Выбор действия
                # В SARSA следующее действие выбиралось после шага в среде
                action = self.make_action(state)

                # Выполняем шаг в среде
                next_state, rew, done, truncated, _ = self.env.step(action)

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

                # Правило обновления для Q-обучения
                self.Q[state][action] = self.Q[state][action] + self.lr * \
                    (rew + self.gamma * np.max(self.Q[next_state]) - self.Q[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)
```

In [8]:

```
class DoubleQLearning_Agent(BasicAgent):
    '''
    Реализация алгоритма Double Q-Learning
    '''
    # Наименование алгоритма
    ALGO_NAME = 'Двойное 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)
        # Вторая матрица
        self.Q2 = np.zeros((self.nS, self.nA))
        # 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 greedy(self, state):
        '''
        <<Жадное>> текущее действие
        Возвращает действие, соответствующее максимальному Q-значению
        для состояния state
        '''
        temp_q = self.Q[state] + self.Q2[state]
        return np.argmax(temp_q)

    def print_q(self):
        print(f"Вывод Q-матриц для алгоритма {self.ALGO_NAME}")
        print('Q1')
        print(self.Q)
        print('Q2')
        print(self.Q2)

    def learn(self):
        '''
        Обучение на основе алгоритма Double Q-Learning
        '''
        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 действ
            if self.eps > self.eps_threshold:
                self.eps -= self.eps_decay
```

```
            # Проигрывание одного эпизода до финального состояния
            while not (done or truncated):
                # Выбор действия
                # В SARSA следующее действие выбиралось после шага в среде
                action = self.make_action(state)

                # Выполняем шаг в среде
                next_state, rew, done, truncated, _ = self.env.step(action)

                if np.random.rand() < 0.5:
                    # Обновление первой таблицы
                    self.Q[state][action] = self.Q[state][action] + self.lr * \
                        (rew + self.gamma * self.Q2[next_state][np.argmax(self.Q[next_sta
                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_sta

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

In [9]:

```
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
```

In [10]:

```
def plot_rewards(x, y):
    # Построение графика наград по эпизодам
    fig, ax = plt.subplots(figsize = (15,10))
    plt.plot(x, y, '-', linewidth=1, color='green')
    plt.title('Награды')
    plt.xlabel('Параметр')
    plt.ylabel('Награда')
    plt.show()
```

In [11]:

```
def bruteforce_sarsa():
    env = gym.make('Taxi-v3')
    rewards_eps = []
    rewards_lr = []
    rewards_gamma = []
    x = np.arange(0.1, 1, 0.1)
    for i in x:
        agent = SARSA_Agent(env,eps=i)
        agent.learn()
        agent.print_q()
        rewards_eps.append(np.asarray(agent.episodes_reward).sum())
    plot_rewards(x, rewards_eps)
    best_eps = x[rewards_eps.index(max(rewards_eps))]
    print(f"Best eps: {best_eps}")
    x = np.arange(0, 1, 0.03)
    for i in x:
        agent = SARSA_Agent(env, eps = best_eps, lr = i)
        agent.learn()
        agent.print_q()
        rewards_lr.append(np.asarray(agent.episodes_reward).sum())
    best_lr = x[rewards_lr.index(max(rewards_lr))]
    print(f"Best lr: {best_lr}")
    plot_rewards(x, rewards_lr)
    x = np.arange(0, 1, 0.03)
    for i in x:
        agent = SARSA_Agent(env, eps = best_eps, lr = best_lr, gamma = i)
        agent.learn()
        agent.print_q()
        rewards_gamma.append(np.asarray(agent.episodes_reward).sum())
    best_gamma = x[rewards_gamma.index(max(rewards_gamma))]
    print(f"Best gamma: {best_gamma}")
    plot_rewards(x, rewards_gamma)
    print(rewards_eps)
    print(rewards_lr)
    print(rewards_gamma)
    print(f"Best params: eps={best_eps}, lr={best_lr}, gamma={best_gamma}")

def run_sarsa():
    env = gym.make('Taxi-v3')
    agent = SARSA_Agent(env, eps=0.1, lr=0.33, gamma=0.99)
    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)
```

In [12]:

```
bruteforce_sarsa()
```

```
100%|██████████| 20000/20000 [00:16<00:00, 1232.55it/s]

Вывод Q-матрицы для алгоритма  SARSA
[[ 0.          0.          0.          0.          0.          0.
]
 [-2.0653178  -3.17570924 -4.31915177 -2.12840343  7.16928295 -3.9596160
8]
 [-0.05364352  0.31592053  1.08554057 -0.89922633 12.98960843 -2.2762735
3]
 ...
 [-1.31089377  0.07919229 -1.32782099 -1.28302485 -2.75512488 -2.8102807
2]
 [-3.08908918 -2.37960685 -3.12597332 -3.15944558 -4.85511671 -4.7639060
7]
 [-0.19       -0.1998     -0.19       11.7131259  -1.9        -1.9098
]]

100%|██████████| 20000/20000 [00:10<00:00, 1905.34it/s]

Вывод Q-матрицы для алгоритма  SARSA
```

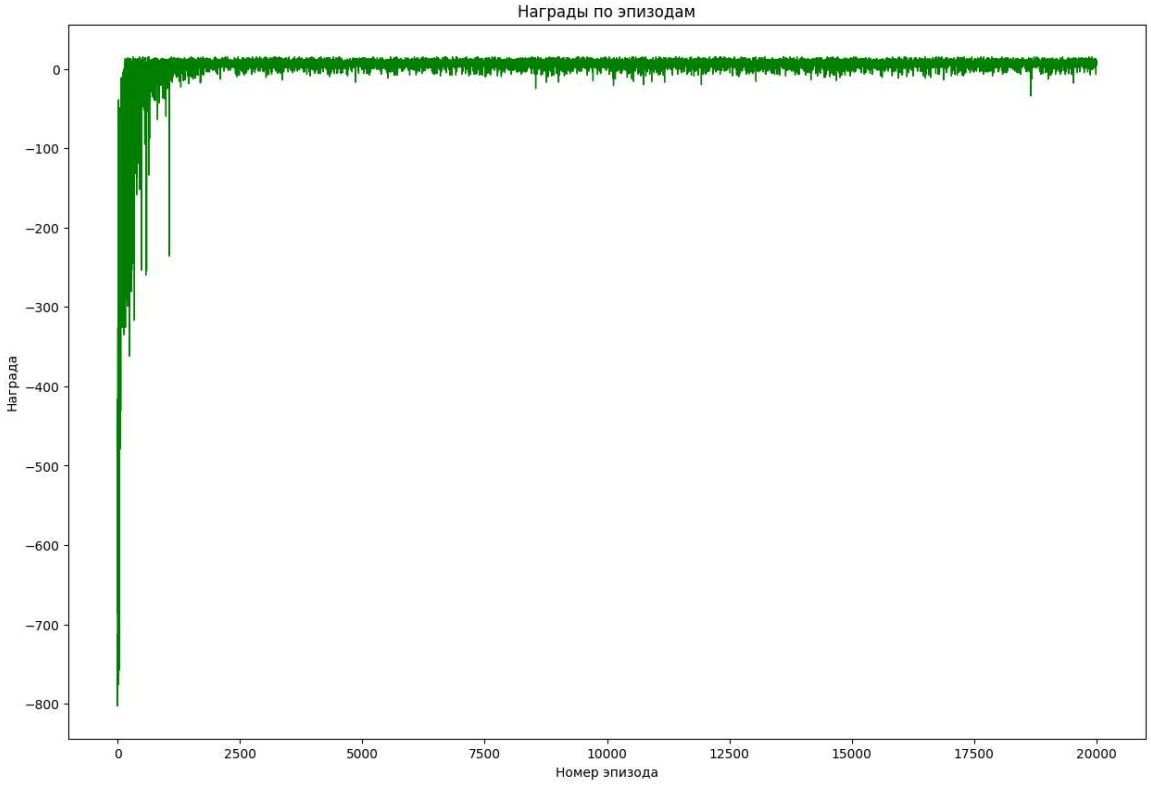
SARSA: eps=0.1, lr=0.33, gamma=0.99, num_episodes=20000

In [13]:

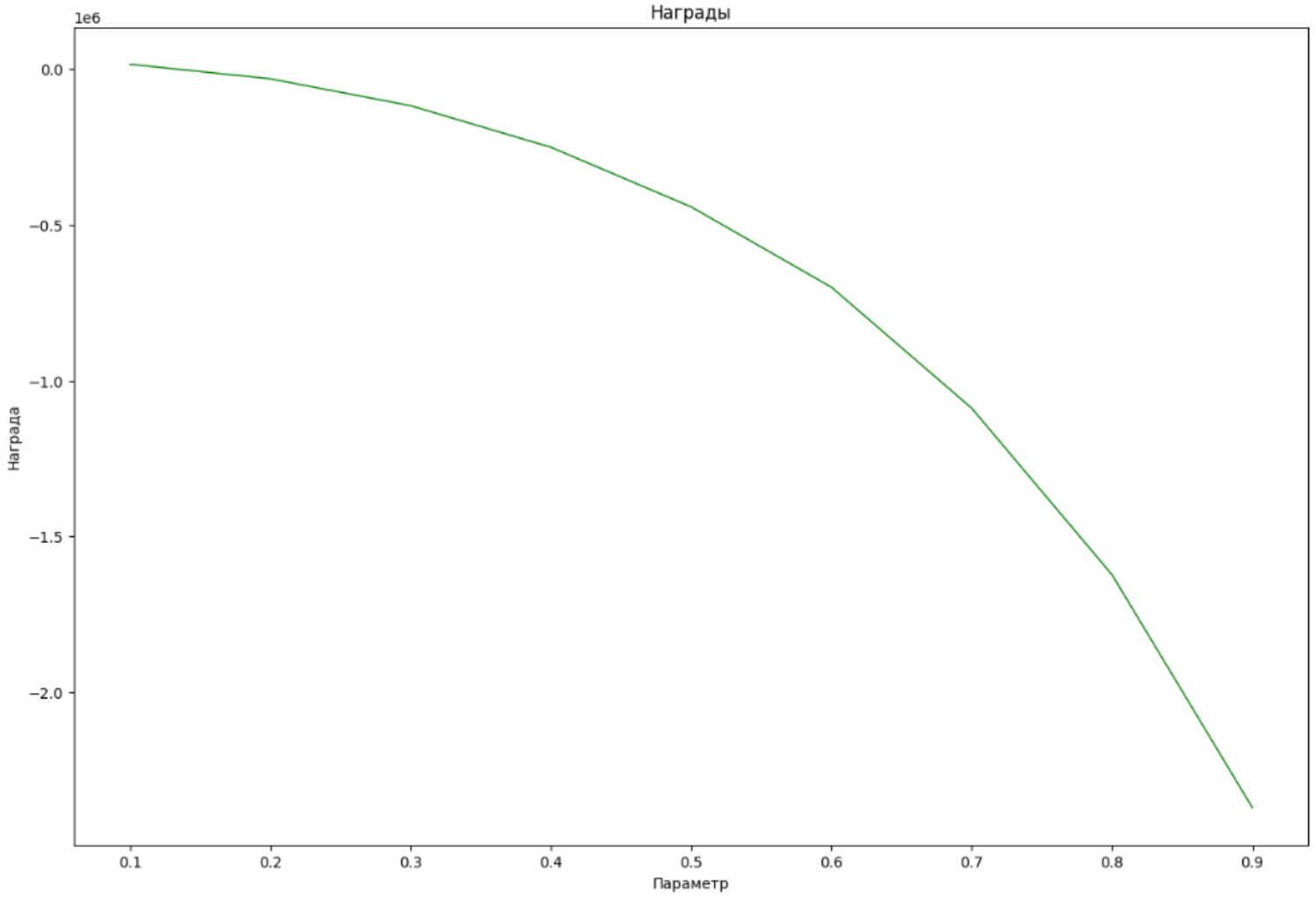
```
run_sarsa()
```

```
100%|██████████| 20000/20000 [00:10<00:00, 1979.79it/s]

Вывод Q-матрицы для алгоритма  SARSA
[[ 0.          0.          0.          0.          0.          0.        ]
 [-2.61052784 -2.64154913 -6.10348968 -0.99542035  3.10580089 -7.86833146]
 [-0.80195019  3.57042242  0.74173644  2.09522323  9.56645827 -4.881662  ]
 ...
 [-2.42725745 -1.97717651 -2.48997284  3.63590283 -3.9        -3.9       ]
 [-2.14656885 -1.99469214 -2.15957614  1.38633056 -6.279      -9.3535425 ]
 [-0.6279     -0.769353   -0.6279     17.53299361 -6.279      -6.420453  
]]
```

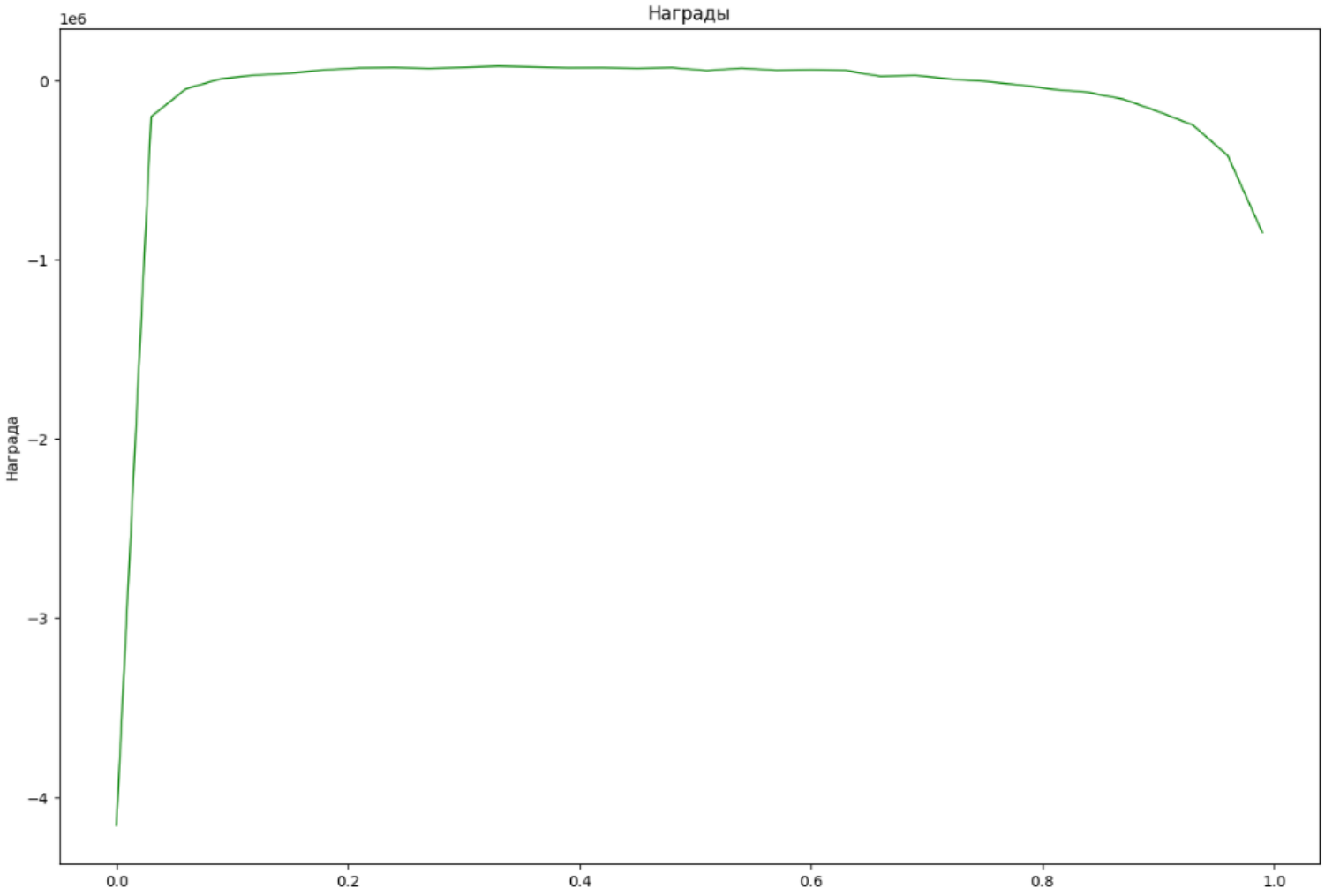


```
Вывод Q-матрицы для алгоритма  SARSA
[[  0.           0.           0.           0.           0.
    0.        ]
 [-23.92892001  -9.89686051 -23.79496921 -10.52410245   8.05745249
  -26.46336845]
 [ -7.51340473   2.12180723  -4.36523308  -0.23230366  12.93920949
   -8.98409269]
 ...
 [-23.13651308   4.54369007 -27.85351997 -28.08158218 -35.5469432
  -35.3669142 ]
 [-52.21653368 -46.91200222 -49.69677312  -7.78060155 -62.63726124
  -56.37048046]
 [ 14.91220905  10.18028574  14.96527255  18.59997406   5.18705741
    4.42043214]]
```




```
Best eps: 0.1
```

```
Вывод Q-матрицы для алгоритма  SARSA
[[  0.           0.           0.           0.           0.
    0.        ]
 [-41.37534603 -41.19063871 -41.56628025 -41.3198229    6.82856599
  -42.96721454]
 [-16.65777324 -43.25356167 -43.23813929 -43.59970841  13.27445578
  -39.21842023]
 ...
 [-22.53564393  14.5657712  -22.83335569 -22.76942824 -27.42677246
  -23.56707743]
 [-54.04857307 -57.04799626 -59.92053797 -54.80034905 -58.08048767
  -59.04303274]
 [  7.34882523   7.02889906   7.35074334  18.6          8.00848906
    8.22608125]]
Best lr: 0.32999999999999996
```



```
Вывод Q-матрицы для алгоритма  SARSA
[[ 0.          0.          0.          0.          0.          0.        ]
 [ 1.70875204  2.1700757   3.92732672  3.6910926   9.55790237 -2.78098883]
 [ 3.75636928 10.42497007  5.23799687  8.13067528 14.11229857 -0.38439832]
 ...
 [-2.72144706 12.52370272 -3.89858789 -2.90085085 -3.3        -8.76053155]
 [-4.73004595 -4.79235023 -4.82739351  7.63460506 -9.00725521 -9.93207083]
 [-0.77147037 -0.78996487 -0.87928137 18.53054291 -5.511      -5.618811  ]]
Best gamma: 0.99
```