

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ  
им. Н.Э. Баумана

Факультет «Информатика и системы управления»  
Кафедра «Систем обработки информации и управления»

## ОТЧЕТ

**Лабораторная работа №5**  
по дисциплине «Методы машинного обучения»

Тема: «Обучение на основе временных различий»

ИСПОЛНИТЕЛЬ: Подопригорова Н.С.  
группа ИУ5-24М \_\_\_\_\_  
"\_\_\_" \_\_\_\_\_ 2023 г.

ПРЕПОДАВАТЕЛЬ: \_\_\_\_\_  
\_\_\_\_\_  
"\_\_\_" \_\_\_\_\_ 2023 г.

Москва - 2023

## 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).

## Текст программы.

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

ENV_NAME = 'Taxi-v3'

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

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)

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 действия
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][action])

    # 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])
#           (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)

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('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 действия
        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_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(ENV_NAME, render_mode='human')  

state = env2.reset()[0]  

done = False  

tot_rew = 0  

while not done:  

    action = agent.greedy(state)  

    next_state, reward, terminated, truncated, _ = env2.step(action)  

    env2.render()  

    state = next_state  

    tot_rew += reward  

    if terminated or truncated:  

        done = True  

        print(tot_rew)

def run_sarsa():
    env = gym.make(ENV_NAME)
    agent = SARSA_Agent(env)
    agent.learn()
    agent.print_q()
    agent.draw_episodes_reward()
    play_agent(agent)

def run_q_learning():
    env = gym.make(ENV_NAME)
    agent = QLearning_Agent(env)
    agent.learn()
    agent.print_q()
    agent.draw_episodes_reward()
    play_agent(agent)

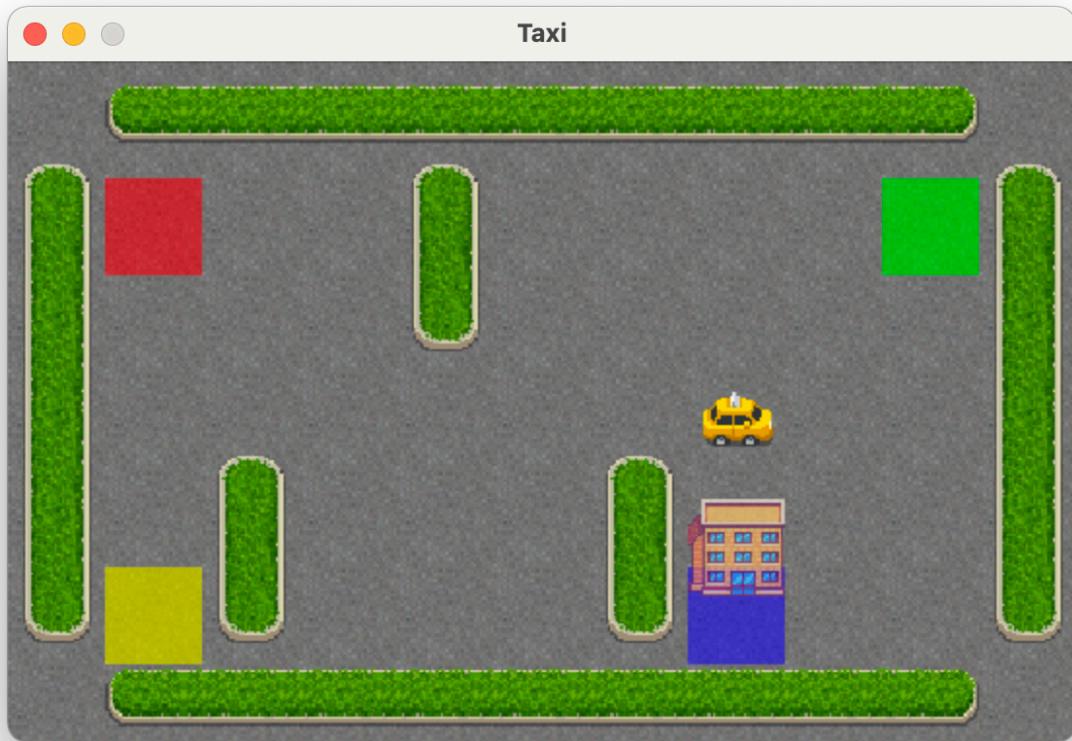
def run_double_q_learning():
    env = gym.make(ENV_NAME)
    agent = DoubleQLearning_Agent(env)
    agent.learn()
    agent.print_q()
    agent.draw_episodes_reward()
    play_agent(agent)

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

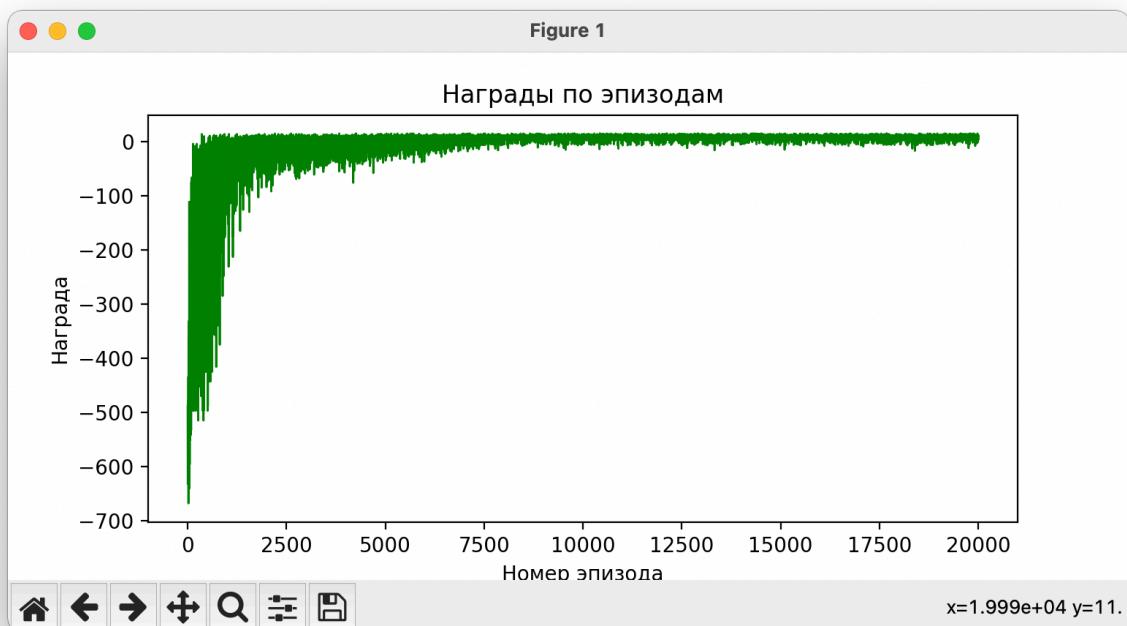
if __name__ == '__main__':
    main()

```

## Экранные формы с примерами выполнения программы



## Q-обучение

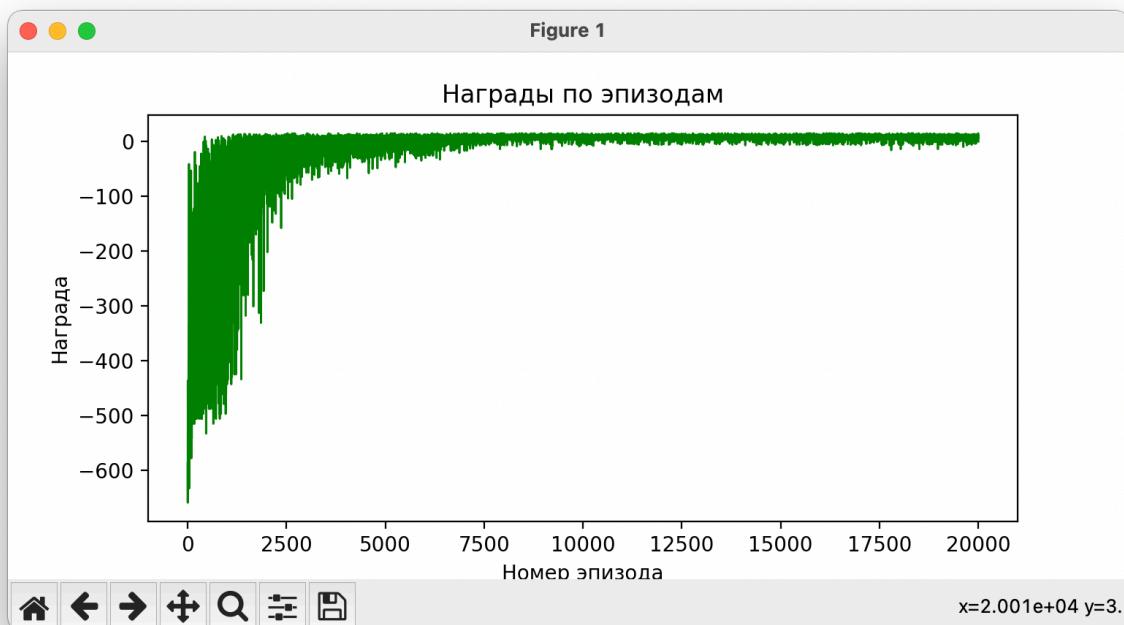


Вывод Q-матрицы для алгоритма Q-обучение

```
[ 0.          0.          0.          0.          0.          0.        ]  
[ 4.5262931   6.08654698  3.24566334  5.92962062  8.36234335 -2.54799492 ]
```

```
[10.4427577 11.47462066 9.68716683 11.60760916 13.27445578 2.86913967]
[ 0.30150319 12.62182526 -1.23707522  0.02791121 -2.79550468 -6.48357256]
[-2.55258955 -2.84928305 -2.88443289  8.57983872 -9.22201791 -8.1838152 ]
[ 3.07972811  6.90743572  7.76874971 18.59864666  0.73072799  1.4461676 ]
10
```

## Двойное Q-обучение



Вывод Q-матриц для алгоритма Двойное Q-обучение

```
Q1
[[ 0.          0.          0.          0.          0.          0.        ]
 [ 3.04059601  3.16769564  2.11447449  2.75659399  8.36234335 -6.35527011]
 [ 4.61849147  6.26000034  5.03272104  7.13165288 13.27445578  1.15212944]

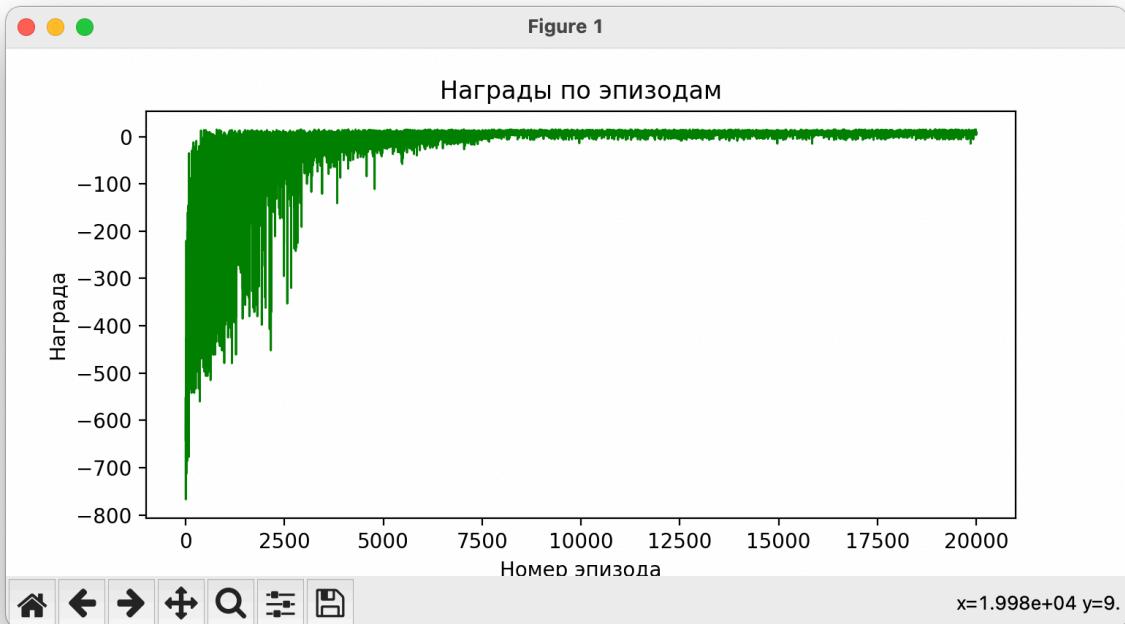
[ 1.41163487 14.26735473 -0.1015012 -1.369295   -3.74498636 -3.02315768]
[-4.53868487 -5.33904719 -4.35619169  5.66299791 -9.65096293 -6.38794166]
[-0.1         5.45863425 -0.1107604  18.48555063 -0.93114132  0.25804376]

Q2
[[ 0.          0.          0.          0.          0.          0.        ]
 [ 0.42398752  3.30829188 -3.5538858  2.59422112  8.36234335 -6.62800945]
 [ 7.55882247  8.5434893   3.58727302  6.52004098 13.27445578 -0.23234787]

[-0.47216536 13.21722684  0.61900007 -0.06417369 -3.73303445 -3.99032572]
[-4.67644096 -3.87884637 -4.06549384  5.65815358 -9.75122463 -4.99960509]
[ 1.59181272  1.39929213  2.74749855 18.29840424 -1.42364394 -1.0098 ]]
```

10

## SARSA



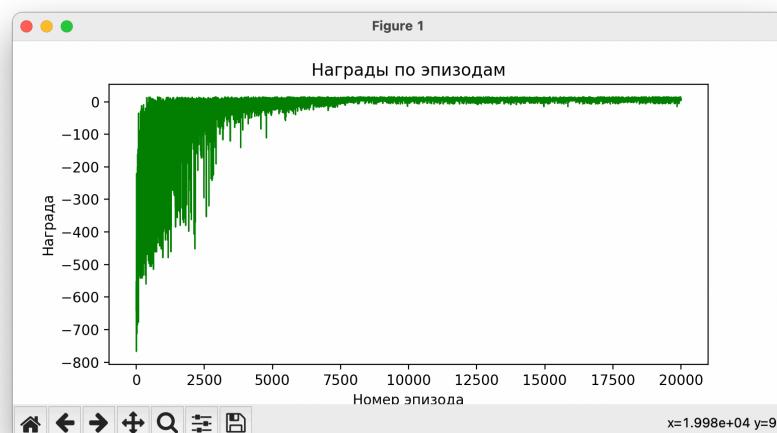
Вывод Q-матрицы для алгоритма SARSA

```
[[ 0.          0.          0.          0.          0.  
  0.          ]  
 [ -3.89432412 -0.44417085 -4.88360641 -3.32735086  6.44669903  
 -13.94421576]  
 [  3.17807795   1.55285704  -1.95855559   5.21144439 13.18275759  
 -6.443111 ]  
 ...  
 [ -3.51452316  10.63736841  -2.95432848  -3.1749972  -7.30948553  
 -8.1570948 ]  
 [ -7.32851489  -6.79460728  -7.29290953  -0.0176256  -10.32113435  
 -13.73815408]  
 [  3.64562315   3.18920852   0.52095186 17.53499977  -1.11483595  
 -4.53496358]]
```

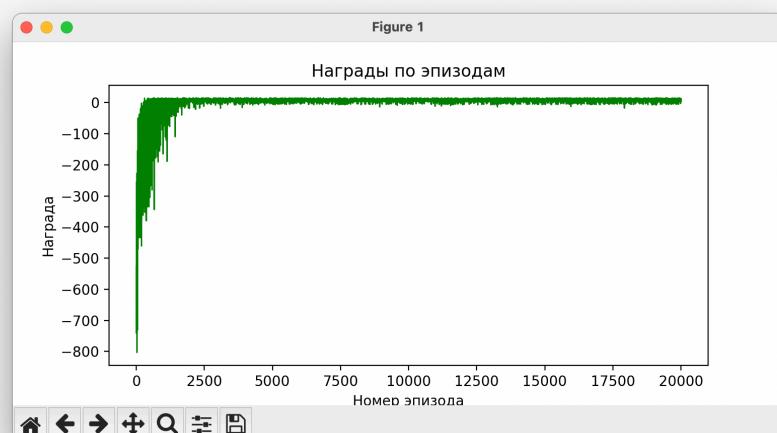
12

## Эксперименты с гиперпараметрами

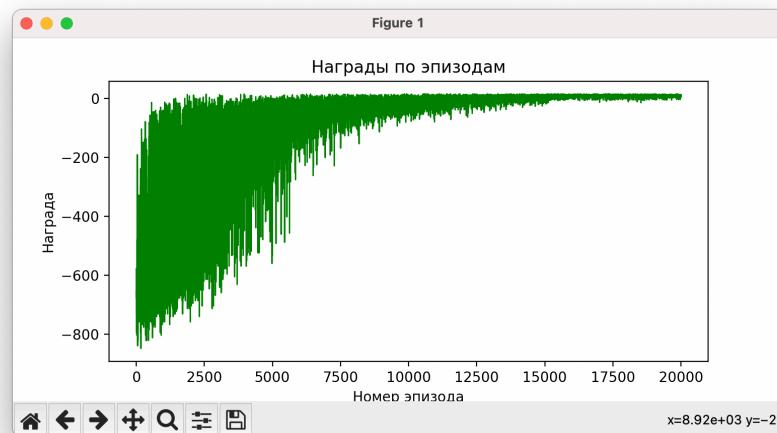
eps=0.4,  
lr=0.1,  
gamma=0.98,  
num\_episodes=20000



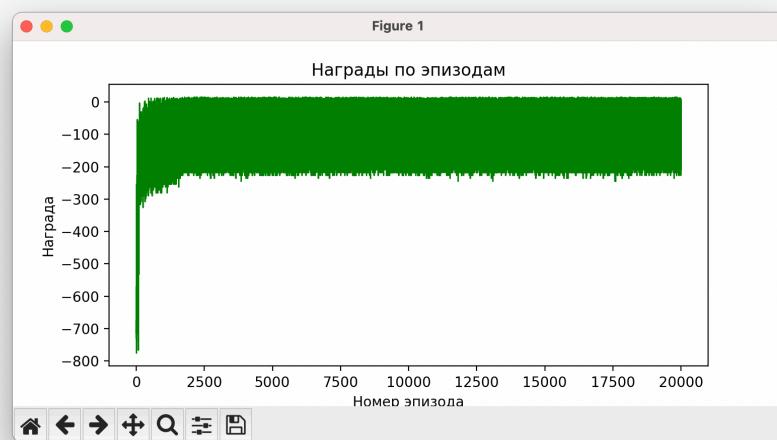
eps=0.1,  
lr=0.1,  
gamma=0.98,  
num\_episodes=20000



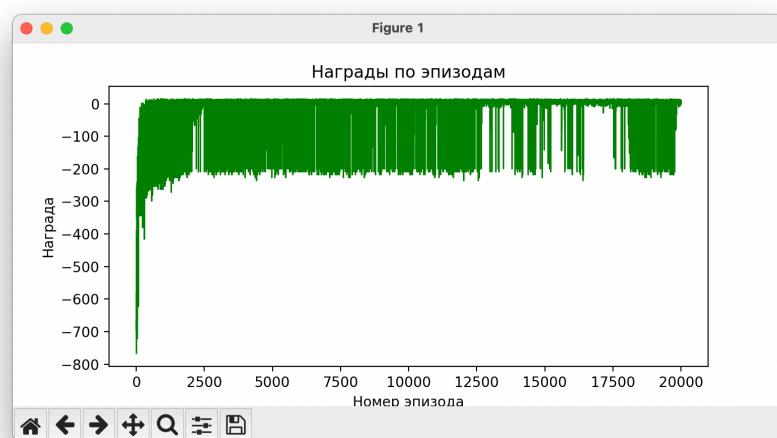
eps=0.8,  
lr=0.1,  
gamma=0.98,  
num\_episodes=20000



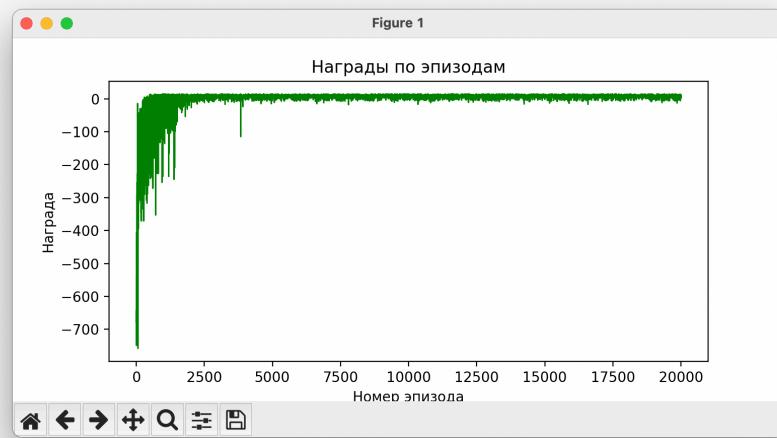
eps=0.1,  
lr=0.1,  
gamma=0.2,  
num\_episodes=20000



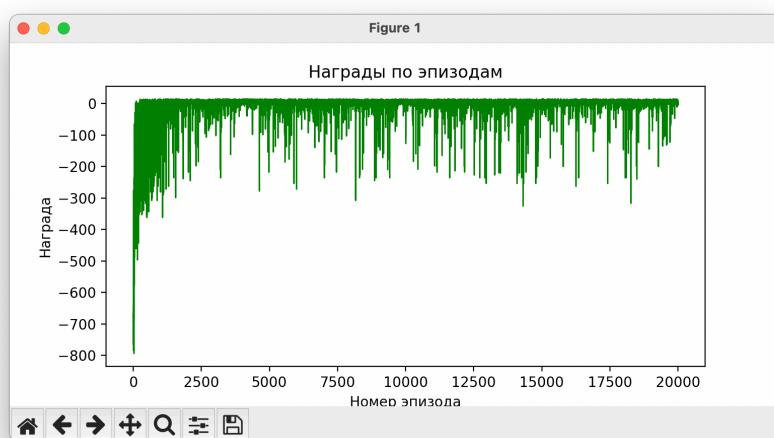
eps=0.1,  
lr=0.1,  
gamma=0.7,  
num\_episodes=20000



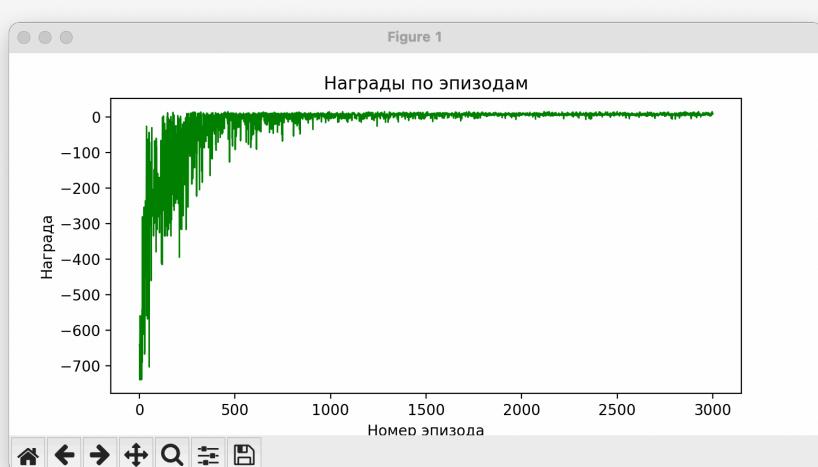
eps=0.1,  
lr=0.1,  
gamma=0.8,  
num\_episodes=20000



eps=0.1,  
lr=0.8,  
gamma=0.98,  
num\_episodes=20000



eps=0.1,  
lr=0.1,  
gamma=0.98,  
num\_episodes=3000



## Выводы

Q-обучение с параметрами по умолчанию сходится быстрее остальных алгоритмов.

Для SARSA-алгоритма уменьшение eps позволяет ускорить сходимость.

Слишком большой lr (0,8) увеличивает разброс результатов (некоторые проходы успешны, другие сильно нет). Уменьшение gamma не позволяет достичь наилучшей ценности (агент застревает надолго в одних ячейках).