



**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»
Кафедра ИУ5 «Системы обработки информации и управления»**

**Отчет по лабораторной работе №5
по дисциплине «Методы машинного обучения»
по теме «Обучение на основе временных
различий»**

**Выполнил:
студент группы № ИУ5-21М
Камалов М.Р.
подпись, дата**

**Проверил:
Балашов А.М.
подпись, дата**

2023 г.

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

BasicAgent.py

```
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(self):
        '''
```

```
Реализация алгоритма обучения
'''
pass
```

DoubleQLearning_Agent.py

```
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 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)

```

QLearning Agent.py

```

class QLearningAgent(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])

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

```

SARSA Agent.py

```

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)

```

main.py

```

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_q_learning()
    run_sarsa()

```

```
run_double_q_learning()
```

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

Экранные формы

C:\Users\Pes_Tick\PycharmProjects\Laba_5\Scripts\python.exe C:/Users/Pes_Tick/Documents/GitHub/MMO/Laba_5/main.py

100%|██████████| 20000/20000 [00:07<00:00, 2653.45it/s]

Вывод Q-матрицы для алгоритма Q-обучение

```
[[ 0.          0.          0.          0.          0.          0.         ]  
 [ 5.52143126  6.86775597  5.74511127  6.0405223   8.36234335 -2.78936583]  
 [ 9.72193748 10.70101343  7.71265927 11.6275632  13.27445578  2.18438377]  
 ...  
 [-1.17120277 13.79807389 -0.55308959  1.29691931 -4.80450608 -4.97349337]  
 [-2.95120294  7.88467449 -2.98882175 -2.81009845 -6.16536252 -5.79958292]  
 [ 2.92342313  7.63251837  1.59493208 18.59777911  0.66544083  2.1632928  ]]
```

100%|██████████| 20000/20000 [00:06<00:00, 2942.22it/s]

Вывод Q-матрицы для алгоритма SARSA

```
[[ 0.          0.          0.          0.          0.         ]  
 [ 0.          ]  
 [ -4.89048373 -4.06559968 -6.48184181 -7.29149042  7.03167409  
 -15.08756845]  
 [ 0.64473492  1.24104148  0.9805527   4.72911114 12.8611346  
 -2.73524457]  
 ...  
 [-3.21175813  7.53827108 -2.79026233 -3.42775864 -4.7948677  
 -7.22347721]  
 [-6.69691937 -6.88658671 -6.5703318  -0.66563988 -12.13549877  
 -10.0378743  ]  
 [ 5.52685166  1.18190491  8.83947059 18.51530708 -1.36262535  
 0.96043514]]
```

100%|██████████| 20000/20000 [00:08<00:00, 2424.23it/s]

Вывод Q-матриц для алгоритма Двойное Q-обучение

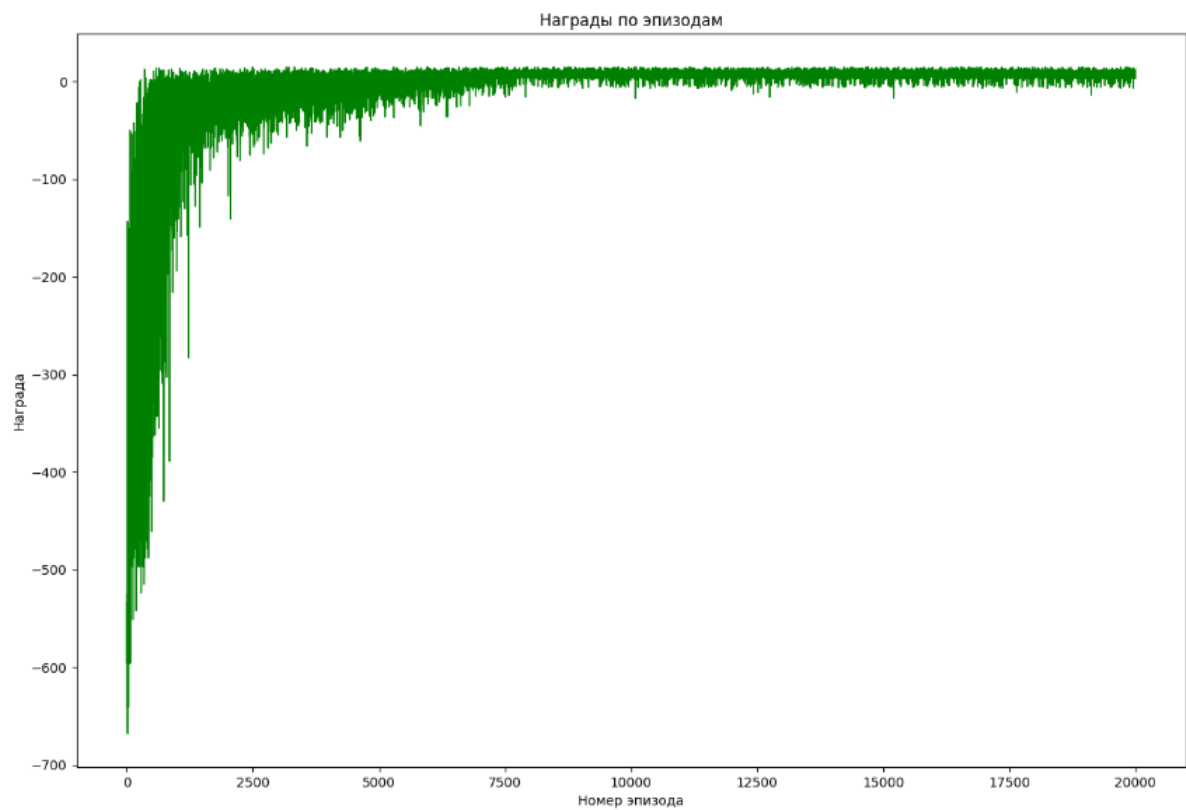
Q1

```
[[ 0.          0.          0.          0.          0.         ]  
 [ 0.          ]  
 [ 1.63853371 -1.0381934  -3.99289027  2.46775567  8.36234335  
 -6.16494027]  
 [ 6.46091158  7.09691525  1.69797111  5.13650562 13.27445578  
 0.508859  ]  
 ...  
 [ 5.7242938  14.5657712  5.55199323  7.72082775 -1.29105504  
 -1.1767286  ]  
 [-4.41801169 -4.00285875 -4.74259869  0.75479753 -9.9385444  
 -10.46810947]  
 [ 4.16798562  2.01402328  1.18758792 18.30237288 -0.71683691  
 -1.44856652]]
```

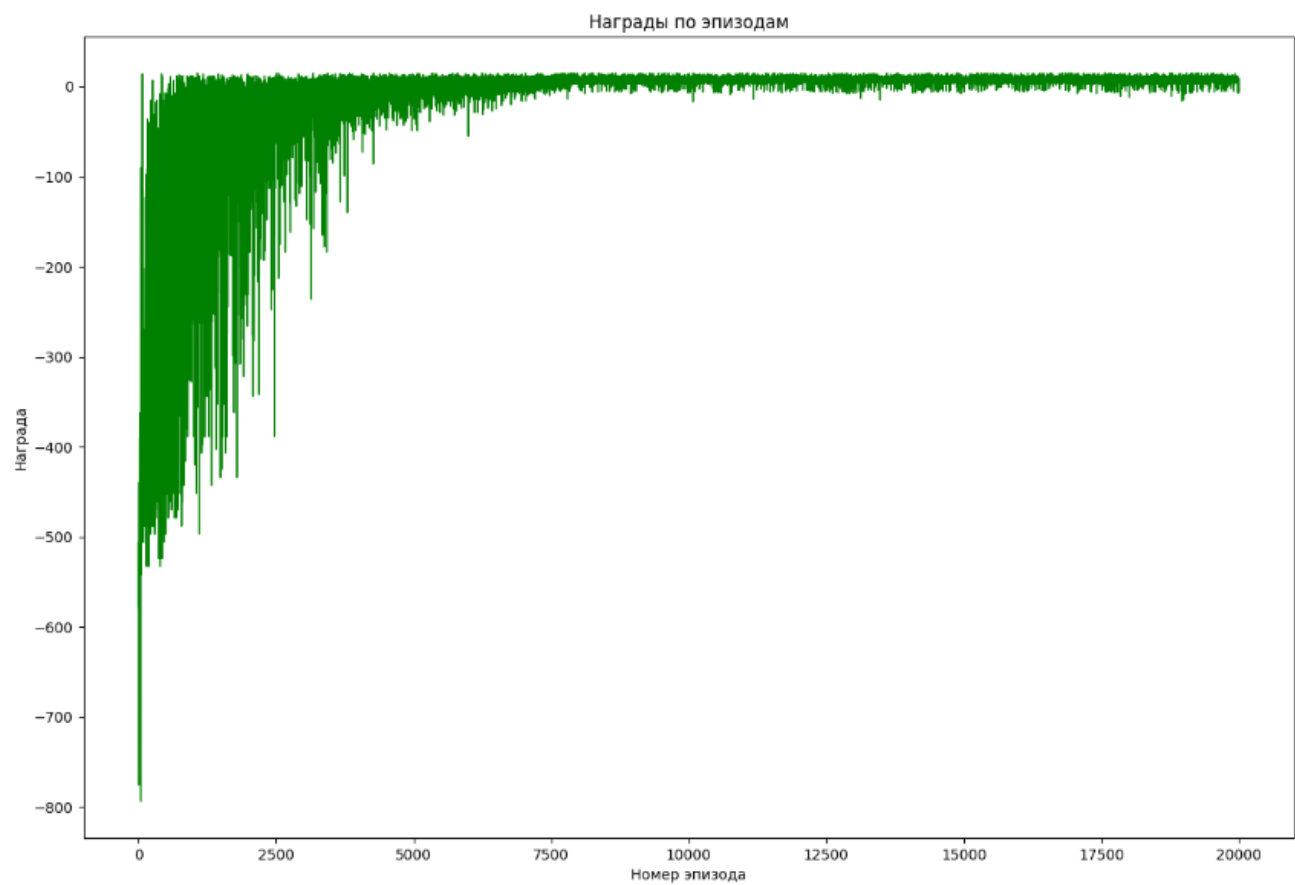
Q2

```
[[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00  
 0.00000000e+00  0.00000000e+00]  
 [ 5.17987161e-03 -1.80363209e+00 -4.58076168e+00  2.04730552e+00  
 8.36234335e+00 -4.29104368e+00]  
 [ 5.56596834e+00  7.80888479e+00  3.54967900e+00  9.47751186e+00  
 1.32744558e+01 -1.80929173e-01]  
 ...  
 [ 8.85848661e+00  1.45657712e+01  7.75419242e+00  5.31600895e+00  
 2.67109603e-01 -2.03947872e+00]  
 [-4.61752088e+00 -4.45744416e+00 -4.16808700e+00  6.31669213e+00  
 -8.51151633e+00 -8.78763830e+00]  
 [ 3.52284065e+00  4.49186663e+00  2.62503908e+00  1.84189537e+01  
 5.21719516e-01  4.29845406e-01]]
```


q_learning



sarsa



double_q_learning

