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 СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ

Лабораторная работа №5

по дисциплине: «Методы машинного обучения»

Студент	Ваганов Даниил Дмитриевич
Группа	ИУ5-24М
Название	Обучение на основе временны́х различий

Студент	_____	**Ваганов Д.Д.**
	подпись, дата	*фамилия, и.о.*
Преподаватель	_____	**Гапанюк Ю.Е.**
	подпись, дата	*фамилия, и.о.*

Оценка _____

Москва, 2024 г.

ЗАДАНИЕ

1. 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).

Выполнение лабораторной работы

Для реализации была выбрана среда Taxi-v3 из библиотеки Gym.

По документации: 500 состояний – 5*5 карта, 4 возможных локации точки выхода, 5 состояний пассажира (4 выхода и в такси).

6 действий – 4 движения и взять/высадить пассажира.

Из 500 состояний в рамках 1 итерации достижимо 400 – исключаются состояния, где пассажир там же, где и здание.

Программа:

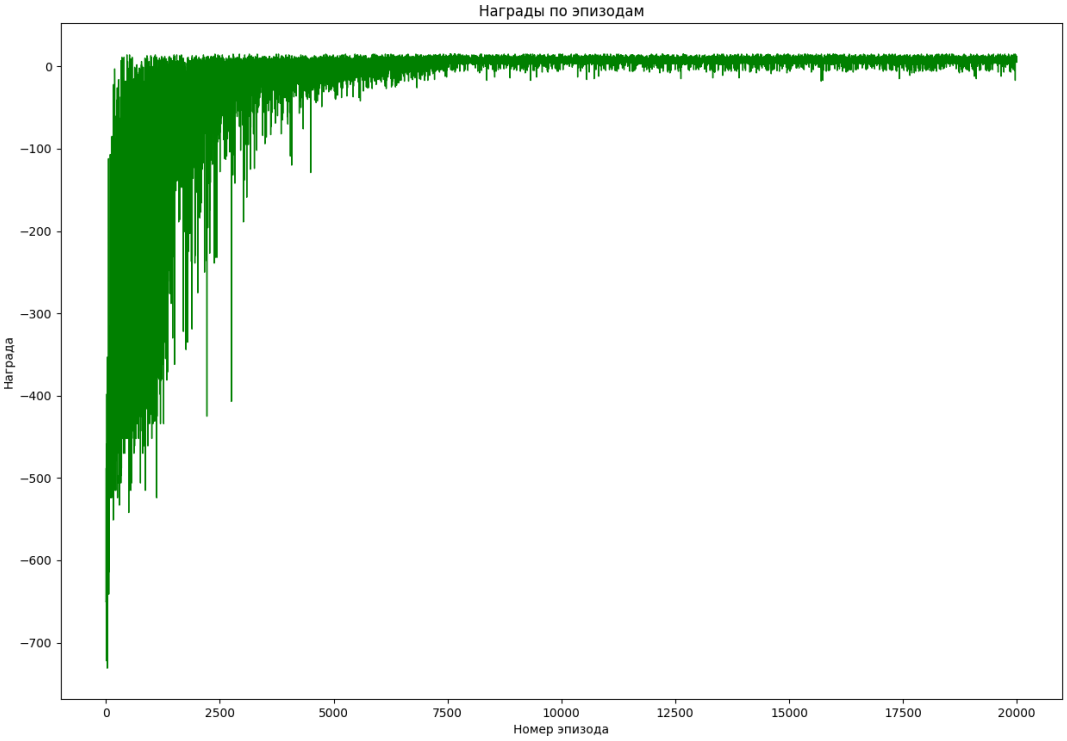
```
import gym
import numpy as np
import matplotlib.pyplot as plt
from pprint import pprint
import pandas as pd
from gym.envs.toy_text.taxi import TaxiEnv
def print_full(x):
  pd.set_option('display.max_rows', len(x))
  print(x)
  pd.reset_option('display.max_rows')

''' Класс, эмулирующий работу агента '''
class PolicyIterationAgent:
  def __init__(self, env):
    self.env = env
    # Пространство состояний
    self.observation_dim = 500
    # Массив действий в соответствии с документацией self.actions_variants =
np.array([0,1,2,3,4,5])
    # Задание стратегии (политики)
    self.policy_probs = np.full((self.observation_dim, len(self.actions_variants)),
0.16666666) # Начальные значения для v(s)
    self.state_values = np.zeros(shape=(self.observation_dim))
    # Начальные значения параметров
    self.maxNumberOfIterations = 1000
    self.theta=1e-6
    self.gamma=0.99
  ''' Вывод матриц стратегии '''
  def print_policy(self):
    '''Вывод матриц стратегии '''
    if self.policy_probs[0][0] != 0.16666666: #np.set_printoptions(threshold=np.inf) x
= TaxiEnv()
      pos = {0:'R', 1:'G',2:'Y', 3:'B', 4:'T'}
      print('''
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
```

```
|Y| : |B: |
+---------+
            ''')
      print('состояние: x,y,пассажир,назначение')
      print('Стратегия:')
      for i in range(len(self.policy_probs)):
        t_x,t_y,passeng,dest = x.decode(i)
        print((t_x,t_y,pos[passeng],pos[dest]), self.policy_probs[i])
#np.set_printoptions(threshold=False)
    else:
      print('Стратегия:')
      pprint(self.policy_probs)

  def policy_evaluation(self):
    '''Оценивание стратегии'''
    # Предыдущее значение функции ценности valueFunctionVector = self.state_values
    for iterations in range(self.maxNumberOfIterations):
      # Новое значение функции ценности
      valueFunctionVectorNextIteration=np.zeros(shape=(self.observation_dim))
      # Цикл по состояниям
      for state in range(self.observation_dim):
        # Вероятности действий
        action_probabilities = self.policy_probs[state] # Цикл по действиям
        outerSum=0
        for action, prob in enumerate(action_probabilities):
          innerSum=0
          # Цикл по вероятностям действий
          for probability, next_state, reward, isTerminalState in
self.env.P[state][action]:

innerSum=innerSum+probability*(reward+self.gamma*self.state_values[next_state])
            outerSum=outerSum+self.policy_probs[state][action]*innerSum
            valueFunctionVectorNextIteration[state]=outerSum
            if(np.max(np.abs(valueFunctionVectorNextIteration-
valueFunctionVector))<self.theta):
              # Проверка сходимости алгоритма
              valueFunctionVector=valueFunctionVectorNextIteration
              break
            valueFunctionVector=valueFunctionVectorNextIteration
          return valueFunctionVector

  def policy_improvement(self):
    '''Улучшение стратегии'''
    qvaluesMatrix=np.zeros((self.observation_dim, len(self.actions_variants)))
    improvedPolicy=np.zeros((self.observation_dim, len(self.actions_variants))) # Цикл
по состояниям
    for state in range(self.observation_dim):
      for action in range(len(self.actions_variants)):
        for probability, next_state, reward, isTerminalState in
self.env.P[state][action]:

qvaluesMatrix[state,action]=qvaluesMatrix[state,action]+probability*(reward+self.gamma
*self.state_values[next_state])
```

```
        # Находим лучшие индексы
        bestActionIndex=np.where(qvaluesMatrix[state,:]==np.max(qvaluesMatrix
[state,:]))
        # Обновление стратегии
        improvedPolicy[state,bestActionIndex]=1/np.size(bestActionIndex)
    return improvedPolicy

  def policy_iteration(self, cnt):
    '''Основная реализация алгоритма '''
    policy_stable = False
    for i in range(1, cnt+1):
      self.state_values = self.policy_evaluation()
      self.policy_probs = self.policy_improvement()
    print(f'Алгоритм выполнился за {i} шагов.')

  def play_agent(agent):
    env2 = gym.make('Taxi-v3',render_mode='human')
    state = env2.reset()[0]
    done = False
    while not done:
      p = agent.policy_probs[state]
      if isinstance(p, np.ndarray):
        action = np.random.choice(len(agent.actions_variants), p=p)
      else:
        action = p
      next_state, reward, terminated, truncated, _ = env2.step(action)
      env2.render()
      state = next_state
      if terminated or truncated:
        done = True

def main():
  # Создание среды
  env = gym.make('Taxi-v3')
  env.reset()
  # Обучение агента
  agent = PolicyIterationAgent(env)
  agent.print_policy()
  agent.policy_iteration(1000)
  agent.print_policy()
# Проигрывание сцены для обученного агента
  play_agent(agent)

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

Результаты выполнения

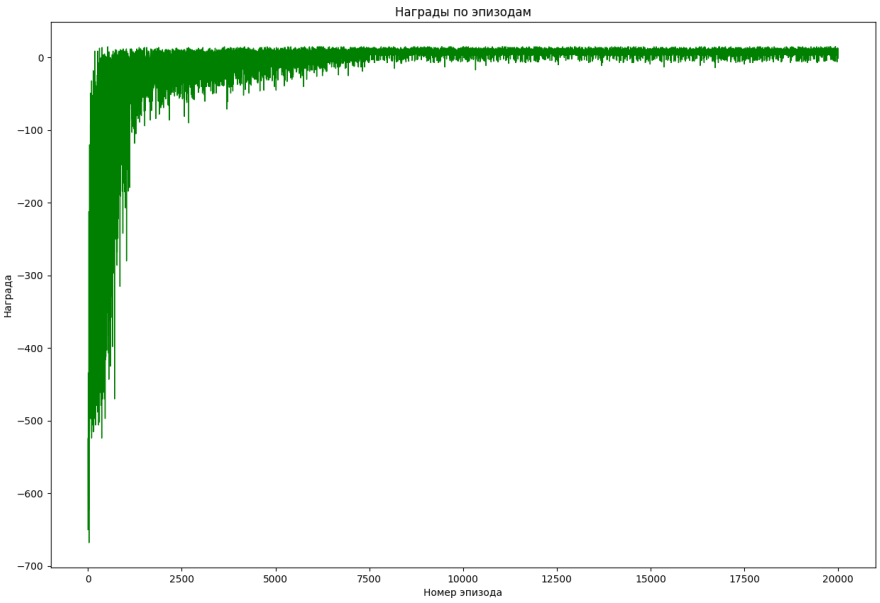
Награда по этапам SARSA



Q-матрица SARSA

```
Вывод Q-матрицы для алгоритма  SARSA
[[  0.           0.           0.           0.           0.
    0.        ]
 [ -6.59510594  -1.96007596  -6.99278993  -3.69658561   7.74183855
  -12.12066264]
 [  2.13952829   1.89041573   1.13819875   3.13294     13.03717386
   -4.96973917]
 ...
 [  4.91965448  14.53492432   4.65312868  -1.16372383  -2.90301868
   -5.06439372]
 [ -8.56123083  -4.78752598  -8.43412478  -8.38525565 -13.8541778
  -13.16174878]
```

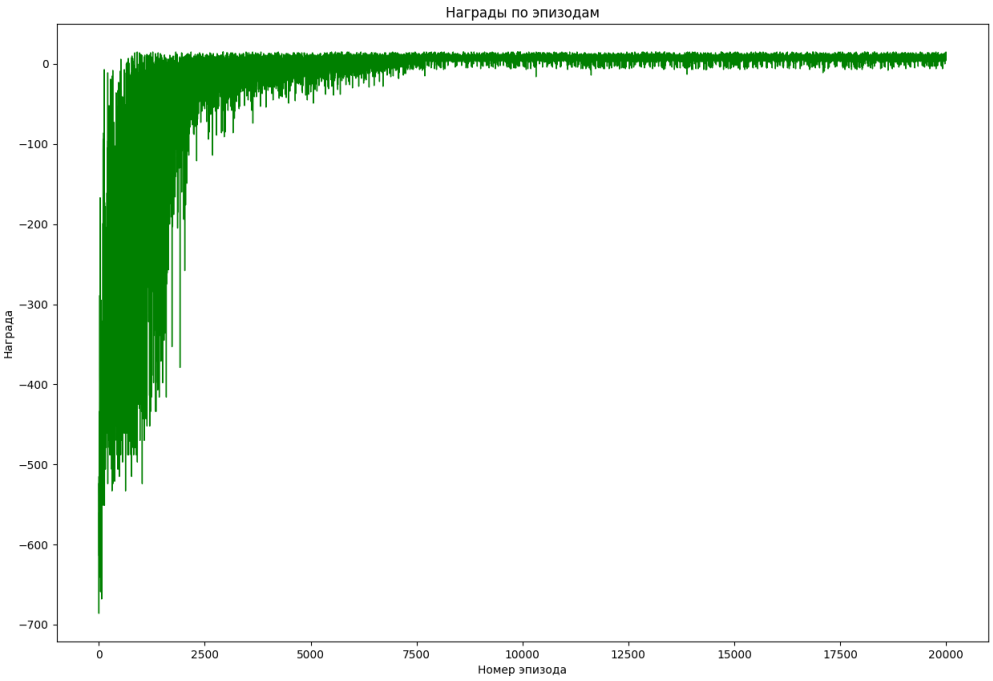
Награды по этапам Q-обучение



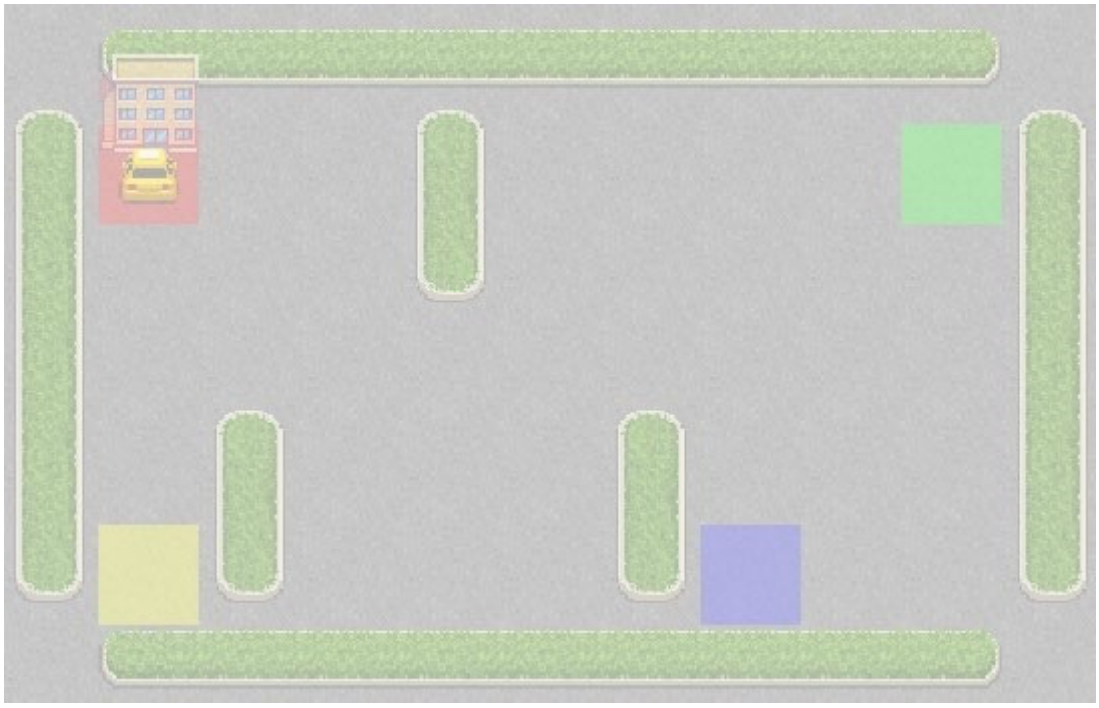
Q-матрица Q-обучения

```
100%|
Вывод Q-матрицы для алгоритма  Q-обучение
[[ 0.          0.          0.          0.          0.          0.        ]
 [ 4.28617674  5.03910366  3.46309801  6.65702923  8.36234335 -3.00320847]
 [10.17488811 10.11231643 10.03753887 11.07922912 13.27445578  2.57480508]
 ...
 [-1.28264741 12.73518582  0.49890481  0.02758999 -4.90913803 -4.93051469]
 [-1.88069918 -0.86090479 -2.5239913   8.89620286 -6.84887216 -9.82047501]
 [ 6.69089885  1.40696151  5.84079492 18.59893093  4.19881129  1.4700502 ]]
```

Награды по этапам dQ-обучение



Конечное состояние



Метод SARSA оказался самым быстрым, 2000 итераций в сек. Против 1900 и 1700итераций в Q и dQ обучении. Вероятно, это связано с max() в Q и появлением промахов кэша в dQ.