# Лабораторная работа №5: Обучение на основе временны’х различий

## Задание:

На основе рассмотренного на лекции примера реализуйте следующие алгоритмы:

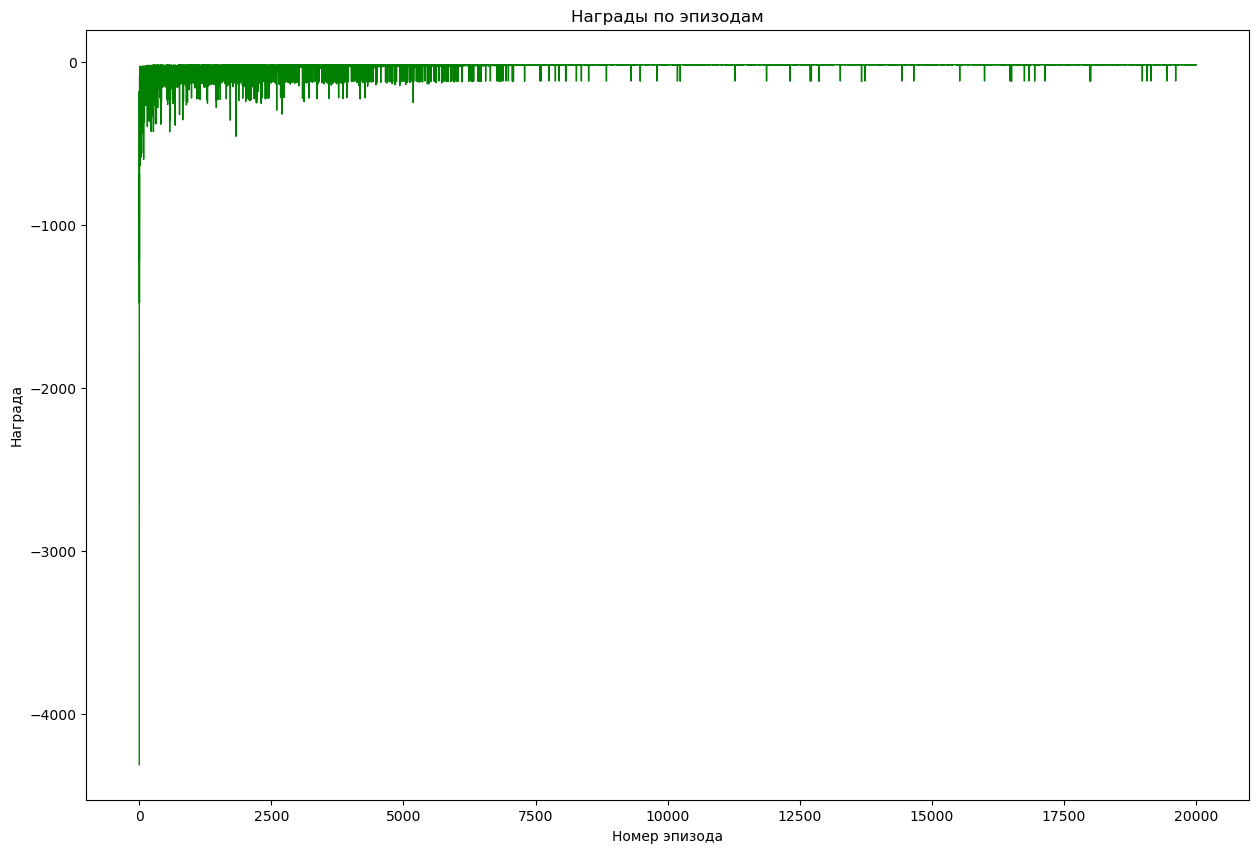
* SARSA
* Q-обучение
* Двойное Q-обучение

для любой среды обучения с подкреплением (кроме рассмотренной на лекции среды Toy Text / Frozen Lake) из библиотеки Gym (или аналогичной библиотеки).

import numpy as np  
import matplotlib.pyplot as plt  
import 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  
 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)  
  
# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Двойное Q-обучение \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
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('CliffWalking-v0', 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('CliffWalking-v0')  
 agent = SARSA\_Agent(env)  
 agent.learn()  
 agent.print\_q()  
 agent.draw\_episodes\_reward()  
 play\_agent(agent)  
  
  
def run\_q\_learning():  
 env = gym.make('CliffWalking-v0')  
 agent = QLearning\_Agent(env)  
 agent.learn()  
 agent.print\_q()  
 agent.draw\_episodes\_reward()  
 play\_agent(agent)  
  
  
def run\_double\_q\_learning():  
 env = gym.make('CliffWalking-v0')  
 agent = DoubleQLearning\_Agent(env)  
 agent.learn()  
 agent.print\_q()  
 agent.draw\_episodes\_reward()  
 play\_agent(agent)  
  
  
def main():  
 run\_sarsa()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

100%|███████████████████████████████████| 20000/20000 [00:02<00:00, 6925.43it/s]

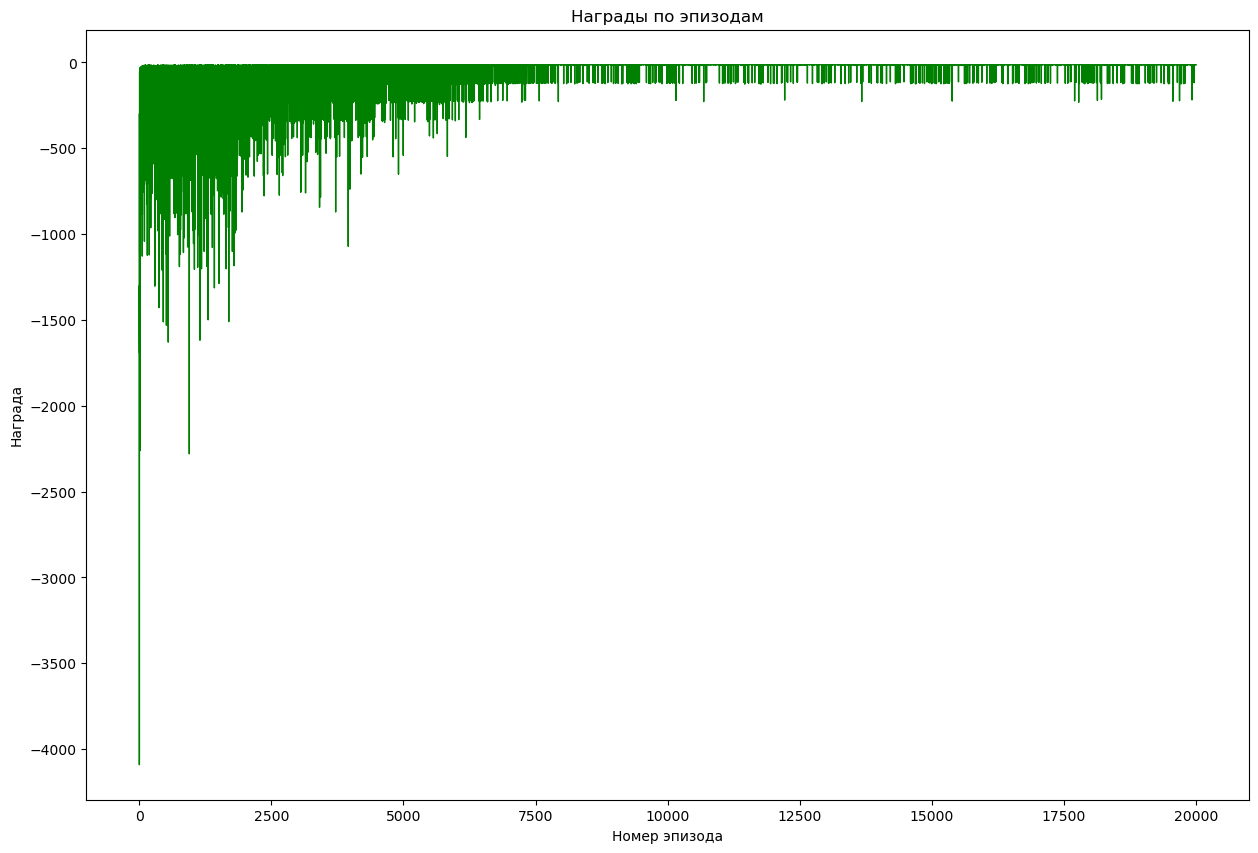
Вывод Q-матрицы для алгоритма SARSA  
[[ -13.25173696 -12.39605989 -14.11623231 -13.25136681]  
 [ -12.5580069 -11.62013456 -13.2919406 -13.55456177]  
 [ -11.69320238 -10.84054371 -12.64144122 -12.60778355]  
 [ -10.89023458 -10.04936099 -11.78937143 -11.88847835]  
 [ -10.15532438 -9.22701619 -11.10834298 -11.07496311]  
 [ -9.26565819 -8.37592232 -10.13306003 -10.27620006]  
 [ -8.380709 -7.5113468 -9.58319642 -9.53399545]  
 [ -7.535188 -6.62410798 -8.56319484 -8.61688964]  
 [ -6.64941925 -5.72206402 -8.31299248 -7.81793856]  
 [ -5.74459465 -4.81122249 -5.24027371 -6.85260578]  
 [ -4.85755447 -3.90127971 -4.33264622 -6.00426343]  
 [ -3.93743148 -3.91822446 -3.00919637 -5.04552495]  
 [ -13.1609511 -13.90915597 -14.86164694 -13.98546187]  
 [ -12.44870221 -16.60849789 -24.99495609 -17.78441873]  
 [ -11.73565197 -16.42963026 -20.72688677 -18.7202329 ]  
 [ -10.8836935 -16.49689852 -30.1364691 -16.9689756 ]  
 [ -10.2299048 -15.03967135 -26.6729615 -15.80129025]  
 [ -9.23276881 -12.90306417 -23.79774664 -15.75006934]  
 [ -8.49153126 -11.49773202 -23.06328649 -14.65430206]  
 [ -7.60427986 -12.31920882 -24.28043646 -14.31971775]  
 [ -6.70926264 -10.50553504 -21.62930622 -13.34867186]  
 [ -7.22485747 -4.02055151 -11.83557384 -10.255277 ]  
 [ -5.80161757 -2.94040004 -16.41028908 -7.12135942]  
 [ -4.11595568 -2.95526673 -2.08509961 -4.55277986]  
 [ -13.91126616 -14.2989152 -16.56712335 -14.72208772]  
 [ -13.27576384 -32.79550902 -128.8020534 -21.60886379]  
 [ -16.34130013 -35.39085524 -124.00422117 -27.00329459]  
 [ -18.38393633 -22.01420098 -121.13199141 -21.52745703]  
 [ -16.25947431 -21.8518423 -103.06984206 -25.73728781]  
 [ -15.05150806 -26.24910789 -112.68559858 -16.80734128]  
 [ -12.84690395 -18.31240623 -130.56049014 -30.12178866]  
 [ -11.83842925 -22.41450962 -102.25532708 -19.62675245]  
 [ -10.10174063 -14.69686842 -124.13174534 -15.06873733]  
 [ -7.81191338 -12.19794119 -95.84443189 -15.97130621]  
 [ -6.27466028 -1.98683675 -121.69205295 -15.49616552]  
 [ -3.17535019 -1.98134832 -1. -3.01097945]  
 [ -14.65128533 -114.47446818 -15.43328319 -15.45303013]  
 [ 0. 0. 0. 0. ]  
 [ 0. 0. 0. 0. ]  
 [ 0. 0. 0. 0. ]  
 [ 0. 0. 0. 0. ]  
 [ 0. 0. 0. 0. ]  
 [ 0. 0. 0. 0. ]  
 [ 0. 0. 0. 0. ]  
 [ 0. 0. 0. 0. ]  
 [ 0. 0. 0. 0. ]  
 [ 0. 0. 0. 0. ]  
 [ 0. 0. 0. 0. ]]



run\_q\_learning()

100%|███████████████████████████████████| 20000/20000 [00:03<00:00, 5984.23it/s]

Вывод Q-матрицы для алгоритма Q-обучение  
[[ -12.43482348 -12.30962461 -12.30887957 -12.40687036]  
 [ -11.8328893 -11.54854076 -11.54852367 -12.46654886]  
 [ -11.25071315 -10.76414645 -10.76414772 -11.94980249]  
 [ -10.67447145 -9.96343 -9.96342987 -11.45824971]  
 [ -9.94838775 -9.14635938 -9.14635936 -10.74631663]  
 [ -9.12667664 -8.31261185 -8.31261185 -9.93345333]  
 [ -8.2783984 -7.46184887 -7.46184887 -9.12484228]  
 [ -7.44169269 -6.59372334 -6.59372334 -8.30661175]  
 [ -6.57984565 -5.70788096 -5.70788096 -7.45900362]  
 [ -5.69977421 -4.80396016 -4.80396016 -6.56353153]  
 [ -4.79847642 -3.881592 -3.881592 -5.67819343]  
 [ -3.86793198 -3.85994091 -2.9404 -4.79859761]  
 [ -13.05425142 -11.54888054 -11.54888054 -12.31778539]  
 [ -12.31655313 -10.76416381 -10.76416381 -12.31790206]  
 [ -11.5488065 -9.96343246 -9.96343246 -11.5488803 ]  
 [ -10.76415199 -9.14635966 -9.14635966 -10.7641638 ]  
 [ -9.96343082 -8.31261189 -8.31261189 -9.96343244]  
 [ -9.14635945 -7.46184887 -7.46184887 -9.14635964]  
 [ -8.31261187 -6.59372334 -6.59372334 -8.31261188]  
 [ -7.46184887 -5.70788096 -5.70788096 -7.46184887]  
 [ -6.59372333 -4.80396016 -4.80396016 -6.59372332]  
 [ -5.70788094 -3.881592 -3.881592 -5.70788094]  
 [ -4.80396016 -2.9404 -2.9404 -4.80396015]  
 [ -3.881592 -2.9404 -1.98 -3.881592 ]  
 [ -12.31790293 -10.76416381 -12.31790293 -11.54888054]  
 [ -11.54888054 -9.96343246 -111.31790293 -11.54888054]  
 [ -10.76416381 -9.14635966 -111.31790293 -10.76416381]  
 [ -9.96343246 -8.31261189 -111.31790293 -9.96343246]  
 [ -9.14635966 -7.46184887 -111.31790293 -9.14635966]  
 [ -8.31261189 -6.59372334 -111.31790293 -8.31261189]  
 [ -7.46184887 -5.70788096 -111.31790293 -7.46184887]  
 [ -6.59372334 -4.80396016 -111.31790293 -6.59372334]  
 [ -5.70788096 -3.881592 -111.31790293 -5.70788096]  
 [ -4.80396016 -2.9404 -111.31790293 -4.80396016]  
 [ -3.881592 -1.98 -111.31790293 -3.881592 ]  
 [ -2.9404 -1.98 -1. -2.9404 ]  
 [ -11.54888054 -111.31790293 -12.31790293 -12.31790293]  
 [ 0. 0. 0. 0. ]  
 [ 0. 0. 0. 0. ]  
 [ 0. 0. 0. 0. ]  
 [ 0. 0. 0. 0. ]  
 [ 0. 0. 0. 0. ]  
 [ 0. 0. 0. 0. ]  
 [ 0. 0. 0. 0. ]  
 [ 0. 0. 0. 0. ]  
 [ 0. 0. 0. 0. ]  
 [ 0. 0. 0. 0. ]  
 [ 0. 0. 0. 0. ]]



run\_double\_q\_learning()

100%|███████████████████████████████████| 20000/20000 [00:03<00:00, 5645.08it/s]

Вывод Q-матриц для алгоритма Двойное Q-обучение  
Q1  
[[ -13.41085121 -12.32039715 -13.22085441 -13.46495809]  
 [ -12.54291913 -12.47561857 -11.54889273 -13.40871041]  
 [ -12.22432693 -11.86887411 -10.77165602 -12.40171682]  
 [ -11.16322267 -10.82534065 -9.96365824 -12.05249325]  
 [ -11.05205443 -10.41639727 -9.1492258 -10.94800023]  
 [ -9.52649573 -10.21246777 -8.3446886 -10.28149595]  
 [ -9.80824246 -8.2362672 -7.47375959 -8.94535926]  
 [ -7.20712046 -6.77046104 -8.92376595 -8.28890266]  
 [ -6.54453082 -5.91904761 -5.69441497 -6.60453439]  
 [ -4.55872983 -4.86585995 -4.80428923 -6.22828448]  
 [ -4.00220499 -4.0351628 -3.86762016 -4.91395969]  
 [ -3.05974744 -3.83402984 -2.94038794 -3.72068814]  
 [ -13.09663389 -11.56363323 -11.54888054 -12.32233877]  
 [ -12.32489122 -10.76693189 -10.76416381 -12.32127054]  
 [ -11.67875891 -9.96343246 -9.98199815 -11.55600029]  
 [ -10.76491544 -9.1463998 -9.14635966 -10.76571002]  
 [ -10.06740771 -8.36021917 -8.31261189 -9.97056818]  
 [ -9.23612528 -7.48694881 -7.46184887 -9.1475695 ]  
 [ -10.13592084 -7.35396399 -6.59372334 -8.3502819 ]  
 [ -7.79677298 -5.70788096 -5.58136271 -7.42623006]  
 [ -6.73954514 -4.83656972 -4.80396016 -7.41959922]  
 [ -5.9093011 -3.881592 -3.881592 -5.56637289]  
 [ -4.91325692 -2.9404 -2.9404 -4.75224101]  
 [ -4.17759405 -2.92799458 -1.98 -3.85377166]  
 [ -12.31790293 -10.76416381 -12.31790293 -11.54888054]  
 [ -11.54888054 -9.96343246 -111.31790293 -11.54888054]  
 [ -10.76416381 -9.14635966 -111.31790293 -10.76416381]  
 [ -9.96343246 -8.31261189 -111.31790293 -9.96343246]  
 [ -9.14635966 -7.46184887 -111.31790293 -9.14635966]  
 [ -8.31261189 -6.59372334 -111.31790293 -8.31261189]  
 [ -7.46184887 -5.70788096 -111.31790293 -7.46184887]  
 [ -7.00752026 -4.80396016 -111.31790293 -6.59372334]  
 [ -5.70788096 -3.881592 -111.31790293 -5.70788096]  
 [ -4.80396016 -2.9404 -111.31790293 -4.80396016]  
 [ -3.881592 -1.98 -111.31790293 -3.881592 ]  
 [ -2.9404 -1.98 -1. -2.9404 ]  
 [ -11.54888054 -111.31790293 -12.31790293 -12.31790293]  
 [ 0. 0. 0. 0. ]  
 [ 0. 0. 0. 0. ]  
 [ 0. 0. 0. 0. ]  
 [ 0. 0. 0. 0. ]  
 [ 0. 0. 0. 0. ]  
 [ 0. 0. 0. 0. ]  
 [ 0. 0. 0. 0. ]  
 [ 0. 0. 0. 0. ]  
 [ 0. 0. 0. 0. ]  
 [ 0. 0. 0. 0. ]  
 [ 0. 0. 0. 0. ]]  
Q2  
[[ -13.52899396 -12.32032186 -12.6645909 -13.60071139]  
 [ -12.67669951 -11.96662703 -11.54891178 -13.31685 ]  
 [ -12.06706714 -11.93089044 -10.78086937 -12.4742596 ]  
 [ -11.58349454 -10.69016243 -9.96366437 -11.69300314]  
 [ -10.69695831 -10.4314907 -9.1621448 -11.08584944]  
 [ -9.99939755 -10.00591974 -8.32261303 -9.95651751]  
 [ -8.21090883 -9.41804826 -8.93762021 -9.2638147 ]  
 [ -8.19683209 -6.87491528 -5.41251622 -8.02754711]  
 [ -5.77303005 -5.71169519 -5.76280659 -8.94001014]  
 [ -5.15084343 -4.81342664 -4.77772572 -5.57175425]  
 [ -4.54013115 -3.66599526 -3.86569303 -4.29058777]  
 [ -3.51743242 -3.29295308 -2.93833744 -3.9786991 ]  
 [ -13.08445256 -11.55654407 -11.54888054 -12.32385535]  
 [ -12.32624308 -10.7728287 -10.76416381 -12.31862476]  
 [ -11.61468246 -9.96343246 -9.97352297 -11.55823585]  
 [ -10.76962475 -9.15002451 -9.14635966 -10.76515281]  
 [ -9.99219895 -8.32132977 -8.31261189 -9.9805818 ]  
 [ -9.2535756 -9.02193235 -7.46184887 -9.17637278]  
 [ -10.39418475 -6.55593639 -6.59372334 -8.21750712]  
 [ -9.63536166 -5.73987064 -6.12864401 -9.19786596]  
 [ -6.81357084 -4.80216128 -4.80396016 -6.56501198]  
 [ -5.59132891 -3.881592 -3.881592 -5.88147357]  
 [ -4.95369677 -2.9404 -2.9404 -4.70156161]  
 [ -3.86803839 -2.92889781 -1.98 -3.86472264]  
 [ -12.31790293 -10.76416381 -12.31790293 -11.54888054]  
 [ -11.54888054 -9.96343246 -111.31790293 -11.54888054]  
 [ -10.76416381 -9.14635966 -111.31790293 -10.76416381]  
 [ -9.96343246 -8.31261189 -111.31790293 -9.96343246]  
 [ -9.14635966 -7.46184887 -111.31790293 -9.14635966]  
 [ -8.31261189 -6.59372334 -111.31790293 -8.31261189]  
 [ -8.22166437 -5.70788096 -111.31790293 -7.46184887]  
 [ -6.59372334 -4.80396016 -111.31790293 -6.59372334]  
 [ -5.73987783 -3.881592 -111.31790293 -5.70788096]  
 [ -4.80396016 -2.9404 -111.31790293 -4.80396016]  
 [ -3.881592 -1.98 -111.31790293 -3.881592 ]  
 [ -2.9404 -1.98 -1. -2.9404 ]  
 [ -11.54888054 -111.31790293 -12.31790293 -12.31790293]  
 [ 0. 0. 0. 0. ]  
 [ 0. 0. 0. 0. ]  
 [ 0. 0. 0. 0. ]  
 [ 0. 0. 0. 0. ]  
 [ 0. 0. 0. 0. ]  
 [ 0. 0. 0. 0. ]  
 [ 0. 0. 0. 0. ]  
 [ 0. 0. 0. 0. ]  
 [ 0. 0. 0. 0. ]  
 [ 0. 0. 0. 0. ]  
 [ 0. 0. 0. 0. ]]

