|  |  |  |
| --- | --- | --- |
| Защищено:  Гапанюк Ю.Е.    "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_2024 г. |  | Демонстрация:  Гапанюк Ю.Е.  "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_2024 г. |

**Отчет по лабораторной работе № 5 по курсу**

**Методы машинного обучения**

**Тема работы: " Реализация алгоритма Policy Iteration "**

10

(количество листов)

Вариант № **15**

|  |  |
| --- | --- |
| ИСПОЛНИТЕЛЬ: |  |
| студент группы ИУ5-22М | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | (подпись) |
| Чиварзин А.Е. | "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_2024 г. |

Москва, МГТУ - 2024

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Задание**

1. На основе рассмотренного на лекции примера реализуйте следующие

алгоритмы:

1. SARSA
2. Q-обучение
3. Двойное Q-обучение

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

1. Сформировать отчет и разместить его в своем репозитории на github.

**Ход выполнения 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.

Код программы:

**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**(**'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\_sarsa()

#run\_q\_learning()

run\_double\_q\_learning**()**

**if** \_\_name\_\_ **==** '\_\_main\_\_'**:**

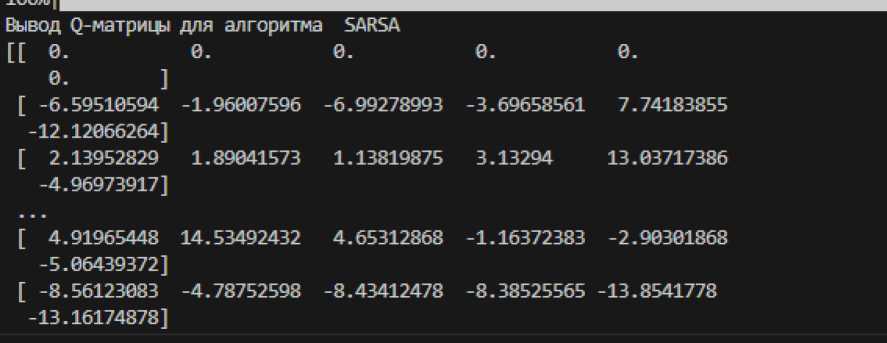
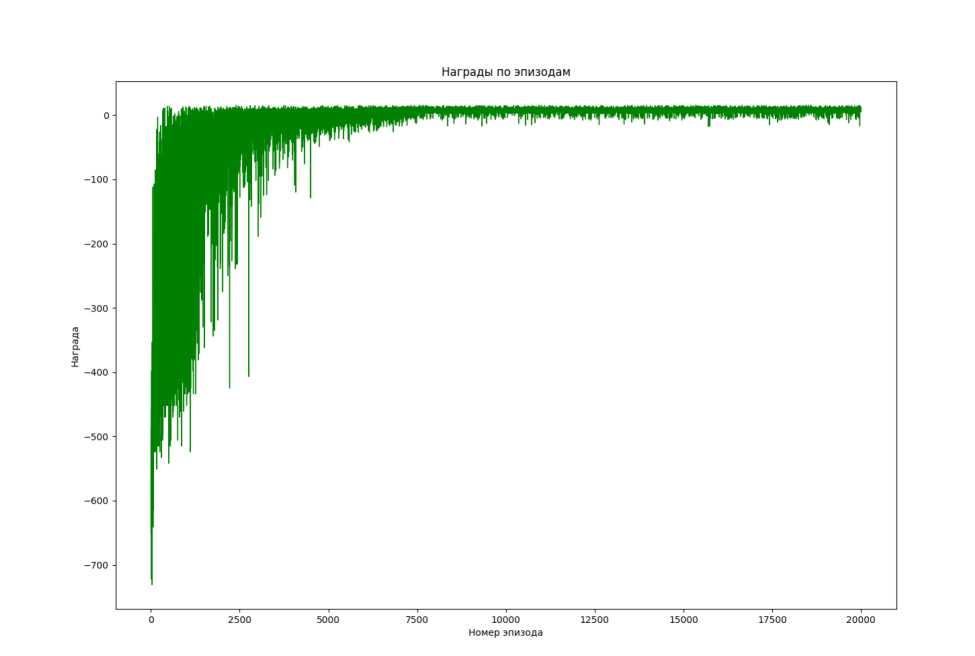
main**()**

Вывод программы модифицирован для кодировки состояний: см. рис. 2.

**Результаты выполнения:**

Рис. 1 - Награды по этапам SARSA.

Рис. 2 - Q-матрица SARSA.



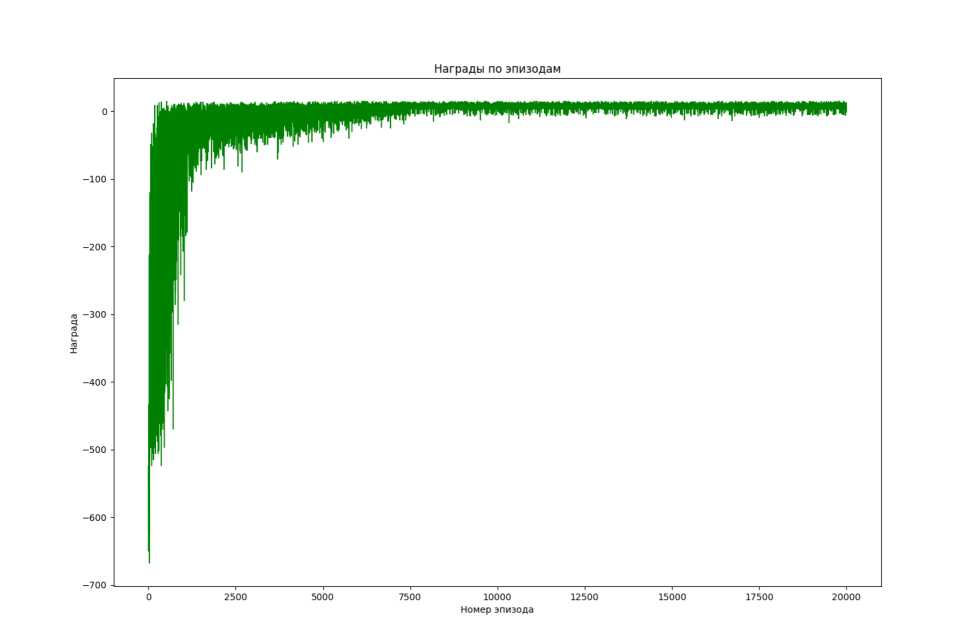


Рис. 3 - Награды по этапам Q-обучение.

I

Вывод Q-MaTpni+1 для алгоритма Q-обучение

[[ 0. 0. 0. 0. 0. 0. ]

[ 4.2В617674 5.03910366 3.46309001 6.65702923 0.36234335 -3.00320347] [10.17438811 10.11231643 10.03753837 11.07922912 13.27445573 2.57430503]

...

[-1.23264741 12.73513532 0.49390431 0.02753999 -4.90913303 -4.93051469] [-1.33069913 -0.36090479 -2.5239913 3.39620236 -6.34337216 -9.82047501]

[ 6.69009335 1.40696151 5.34079492 13.59393093 4.19331129 1.4700502 ]] I

Рис. 4 - Q-матрица Q-обучения.

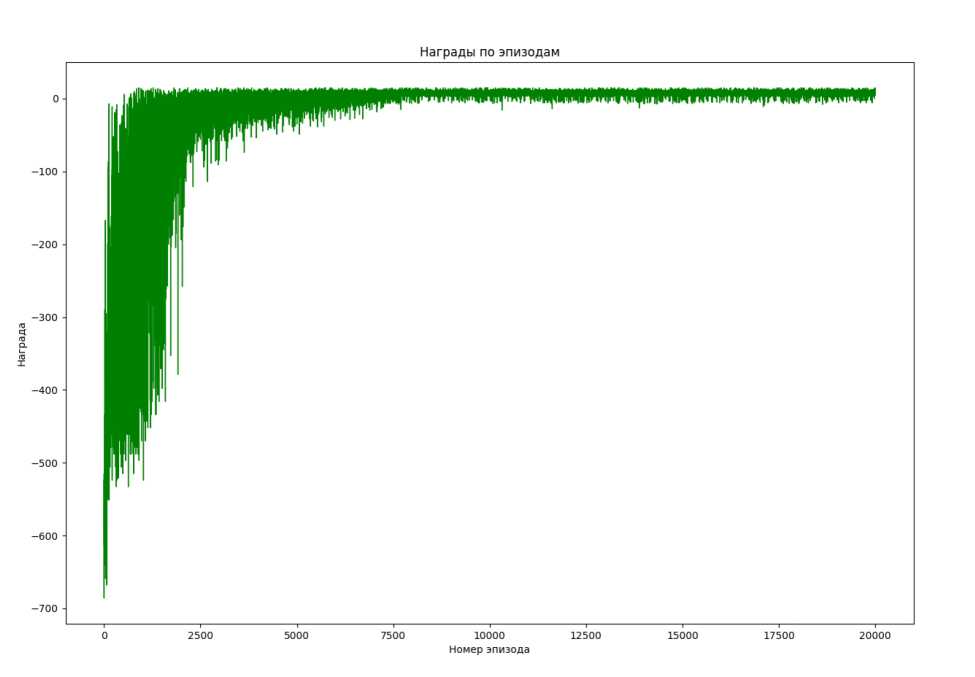


Рис. 5 - Награды по этапам dQ-обучение.

Вывод Q-матриц для алгоритма Двойное Q-обучение

Q1 '

|  |  |  |  |
| --- | --- | --- | --- |
| [[ 0. 0.  0- 1  [ 0.4956301 2.01616935  -4.40342241]  [ 5.58463476 7.83243378  -2.14348329] | 0.  -0.37676916  1.08344404 | 0.  -1.03059973  6.94931956 | 0.  3.36234335  13.27445578 |
| [ -1.41111304 10.63493294 | -1.7376361 | -1.21891579 | -3.02222303 |
| -2.29353635] |  |  |  |
| [ -4.03353292 -4.33230483 | -4.55711092 | 2.70935571 | -10.15945533 |
| -9.65504553] |  |  |  |
| [ 1.40885897 1.98958961 | 3.3739806 | 18.47760311 | 0.65083876 |
| 0.63652395]] |  |  |  |
| Q2 |  |  |  |
| [[ 0. 0. 0. | . 0. | . 0. | 0- ] |
| [-0.3270464 3.49955587 -3. | .33440933 1. | .05364569 8.: | 36234335 -9.95144362] |
| [ 1.33739738 6.33973543 2. | .3069611 7. | .13565323 13.: | 27445573 -0.33734056] |
| [-1.49141743 7.32475969 -1. | .22470337 0. | 33534724 -5. | 72093994 -3.7601113 ] |

[-4.57704201 -4.06435558 -4.26295557 -0.64435005 -9.27489379 -6.2435429 ]

[ 1.33162216 4.41925175 5.21988787 13.33312556 -0.67761377 0.77036741]]

п

Рис. 6 - Q-матрицы dQ-обучение.

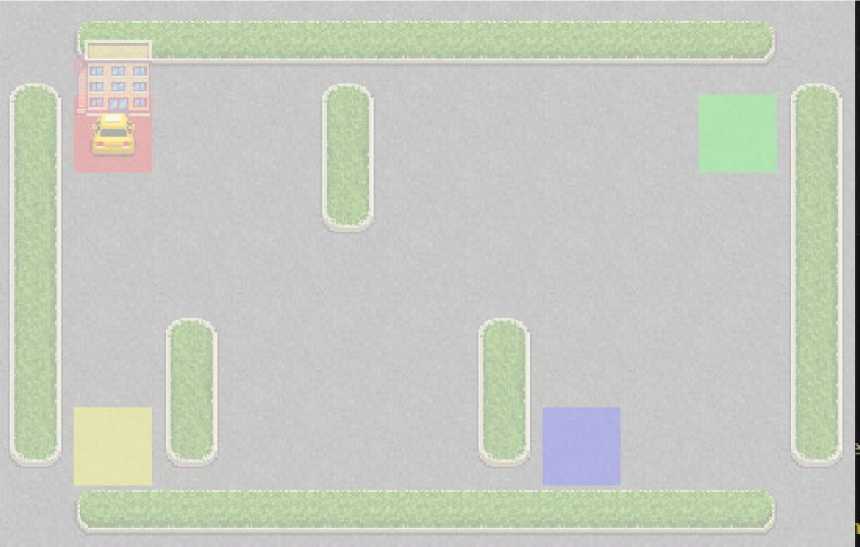


Рис. 7 - конечное состояние.

**Вывод:**

В ходе выполнения лабораторной работы мы ознакомились с базовыми методами обучения с подкреплением на основе временных различий.

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

Для настолько простой симуляции все алгоритмы имеют похожее время схождения, но Q и dQ имеют больший темп начального схождения.