Московский государственный технический университет им. Н.Э. Баумана
Факультет «Головной учебно-исследовательский и методический центр
профессиональной реабилитации лиц с ограниченными возможностями здоровья (инвалидов)»
Кафедра «Системы обработки информации и управления»

Лабораторная работа 5

по дисциплине «Методы машинного обучения в АСОИУ»

" Обучение на основе временных различий"

СТУДЕНТ:
студент группы ИУ5Ц-21М
Москалик А.А.

ПРЕПОДАВАТЕЛЬ:
Гапанюк Ю.Е.

Москва, 2024

Цель лабораторной работы: ознакомление с базовыми методами обучения с подкреплением на основе временных 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).

Ход работы

1. SARSA

Инициализация среды

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

def initialize_env():
    env = gym.make('MountainCar-v0')
    n_states = (env.observation_space.high - env.observation_space.low) * np.array([10, 100])
    n_states = np.round(n_states, 0).astype(int) + 1
    n_actions = env.action_space.n
    return env, n_states, n_actions

env, n_states, n_actions = initialize_env()
```

Дискретизация состояния

Дискретизация состояния — это процесс преобразования непрерывного пространства состояний в дискретное пространство состояний. Это делается для того, чтобы уменьшить количество возможных состояний до управляемого количества, которое можно эффективно использовать в алгоритмах обучения с подкреплением, таких как SARSA. Дискретизация позволяет агенту работать с конечным набором состояний, что упрощает процесс обновления и хранения значений функции ценности (Q-значений).

```
def discretize_state(state, env):
    state_adj = (state - env.observation_space.low) * np.array([10, 100])
    return tuple(np.round(state_adj, 0).astype(int))
```

Инициализация Q-таблицы

Инициализация Q-таблицы — это процесс создания и начальной настройки таблицы Q-значений, которая используется в алгоритмах обучения с подкреплением. Q-таблица представляет собой двумерный массив (или трехмерный массив, если дискретизированное пространство состояний многомерное), где каждая ячейка хранит оценку полезности или ожидаемого вознаграждения для каждой комбинации состояния и действия.

Начальная инициализация Q-таблицы может выполняться случайными значениями, нулями или другими константами, в зависимости от выбранной стратегии. Цель этой таблицы — постепенно обновляться на основе опыта агента, чтобы в итоге содержать значения, которые направляют агента к максимальному суммарному вознаграждению.

```
def initialize_q_table(n_states, n_actions):
    return np.random.uniform(low=-1, high=1, size=(n_states[0], n_states[1], n_actions))

Q = initialize_q_table(n_states, n_actions)
```

Функция выбор действия

Функция выбора действия в контексте обучения с подкреплением — это метод, который определяет, какое действие агент должен предпринять, находясь в определенном состоянии. Эта функция играет ключевую роль в процессе обучения, так как она напрямую влияет на исследование и использование среды агентом.

Основные аспекты функции выбора действия:

- Исследование (Exploration): Агент пробует разные действия, чтобы исследовать среду и узнать больше о возможных вознаграждениях и последствиях своих действий. Это важно на начальных этапах обучения, когда агент еще не накопил достаточно информации о среде.

- Использование (Exploitation): Агент выбирает действия, которые, по его мнению, принесут наибольшее вознаграждение на основе уже накопленного опыта. Это важно на более поздних этапах обучения, когда агент стремится максимизировать свое вознаграждение.

```python
def choose_action(state, Q, n_actions, epsilon):
    if np.random.uniform(0, 1) < epsilon:
        return np.random.randint(n_actions)
    else:
        return np.argmax(Q[state])
```

Обучение агента

Обучение агента в контексте машинного обучения, особенно в обучении с подкреплением, — это процесс, в ходе которого агент взаимодействует с окружающей средой и накапливает опыт, чтобы улучшить свои действия и стратегии с целью максимизации получаемого вознаграждения.

```python
def train_agent(env, Q, n_states, n_actions, alpha, gamma, epsilon, episodes, test_episodes):
    rewards = []
    for episode in range(episodes):
        state = env.reset()
        if isinstance(state, tuple):
            state = state[0]
        state = discretize_state(state, env)
        action = choose_action(state, Q, n_actions, epsilon)
        total_reward = 0
        done = False

        while not done:
            result = env.step(action)
            if len(result) == 4:
                next_state, reward, done, _ = result
            elif len(result) == 5:
                next_state, reward, done, truncated, _ = result
                done = done or truncated

            if isinstance(next_state, tuple):
                next_state = next_state[0]
            next_state = discretize_state(next_state, env)
            next_action = choose_action(next_state, Q, n_actions, epsilon)

            Q[state][action] += alpha * (reward + gamma * Q[next_state][next_action] - Q[state][action])

            state = next_state
            action = next_action
            total_reward += reward

        rewards.append(total_reward)

        if (episode + 1) % 500 == 0:
            print(f"Эпизод {episode + 1}: Средняя награда за последние 500 эпизодов = {np.mean(rewards[-500:])}")
            test_rewards = test_agent(env, Q, n_states, n_actions, test_episodes)
            print(f"Средняя награда за тестовые эпизоды после {episode + 1} эпизодов обучения: {np.mean(test_rewards)}")

    return rewards

rewards = train_agent(env, Q, n_states, n_actions, alpha=0.1, gamma=0.99, epsilon=0.1, episodes=5000, test_episodes=10)
```

```
Эпизод 500: Средняя награда за последние 500 эпизодов = -195.432
Средняя награда за тестовые эпизоды после 500 эпизодов обучения: -184.6
Эпизод 1000: Средняя награда за последние 500 эпизодов = -189.818
Средняя награда за тестовые эпизоды после 1000 эпизодов обучения: -171.3
Эпизод 1500: Средняя награда за последние 500 эпизодов = -189.82
Средняя награда за тестовые эпизоды после 1500 эпизодов обучения: -190.0
Эпизод 2000: Средняя награда за последние 500 эпизодов = -184.62
Средняя награда за тестовые эпизоды после 2000 эпизодов обучения: -165.4
Эпизод 2500: Средняя награда за последние 500 эпизодов = -173.63
Средняя награда за тестовые эпизоды после 2500 эпизодов обучения: -168.2
Эпизод 3000: Средняя награда за последние 500 эпизодов = -180.428
Средняя награда за тестовые эпизоды после 3000 эпизодов обучения: -174.5
Эпизод 3500: Средняя награда за последние 500 эпизодов = -169.618
Средняя награда за тестовые эпизоды после 3500 эпизодов обучения: -144.3
Эпизод 4000: Средняя награда за последние 500 эпизодов = -175.872
Средняя награда за тестовые эпизоды после 4000 эпизодов обучения: -164.1
Эпизод 4500: Средняя награда за последние 500 эпизодов = -181.446
Средняя награда за тестовые эпизоды после 4500 эпизодов обучения: -150.3
Эпизод 5000: Средняя награда за последние 500 эпизодов = -175.228
Средняя награда за тестовые эпизоды после 5000 эпизодов обучения: -185.1
```

Эти результаты показывают, как средняя награда за эпизоды изменялась в процессе обучения агента с использованием алгоритма SARSA.

Тестирование агента

Тестирование агента — это процесс оценки производительности обученного агента в среде. В ходе тестирования агент выполняет действия, основываясь на своей стратегии (обычно на основе Q-таблицы или другой модели), но не обучается и не обновляет свои параметры. Цель тестирования — измерить, насколько эффективно агент выполняет поставленную задачу, используя знания, полученные во время обучения. Тестирование позволяет оценить, как хорошо агент усвоил оптимальную стратегию и насколько эффективно он может достигать целей в среде.

```
def test_agent(env, Q, n_states, n_actions, test_episodes):
    test_rewards = []
    for test_episode in range(test_episodes):
        state = env.reset()
        if isinstance(state, tuple):
            state = state[0]
        state = discretize_state(state, env)
        total_test_reward = 0
        done = False
        while not done:
            action = np.argmax(Q[state])
            result = env.step(action)
            if len(result) == 4:
                next_state, reward, done, _ = result
            elif len(result) == 5:
                next_state, reward, done, truncated, _ = result
                done = done or truncated

            if isinstance(next_state, tuple):
                next_state = next_state[0]
            state = discretize_state(next_state, env)
            total_test_reward += reward
        test_rewards.append(total_test_reward)
    return test_rewards
```

Построение графика наград

Построение графика наград — это визуализация процесса обучения агента с целью отслеживания его прогресса. График наград показывает, как изменяются получаемые агентом вознаграждения по мере прохождения эпизодов обучения.

```
def plot_rewards(rewards):
    plt.plot(rewards)
    plt.xlabel('Эпизоды')
    plt.ylabel('Награды')
    plt.title('Награды в зависимости от эпизодов')
    plt.show()

plot_rewards(rewards)
```

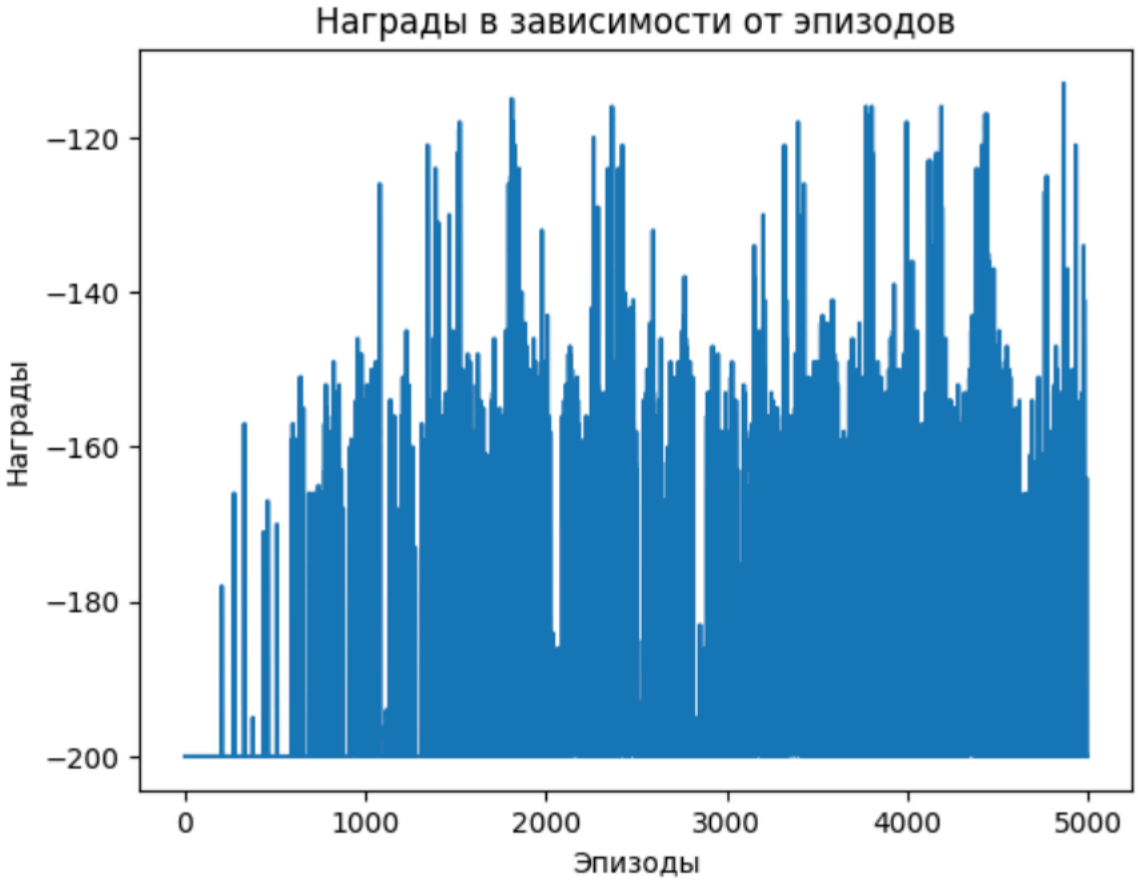


График наград может показать, как агент постепенно улучшает свою стратегию, если награды возрастают со временем.

Если награды остаются низкими или сильно колеблются, это может указывать на необходимость настройки параметров обучения или выбора другого алгоритма.

Вывод Q-таблицы и тестирование модели

Вывод Q-таблицы — это процесс отображения значений Q-таблицы после завершения обучения. Q-таблица хранит оценки полезности для каждой комбинации состояния и действия.

Тестирование модели — это процесс оценки эффективности обученного агента в среде без дальнейшего обучения. Агент выполняет действия, основываясь на текущих значениях Q-таблицы, и собираются данные о его производительности.

```python
def display_q_table(Q):
    print("Часть финальной Q-таблицы:")
    print(Q[:5, :5, :])

display_q_table(Q)

def render_agent(env, Q):
    state = env.reset()
    if isinstance(state, tuple):
        state = state[0]
    state = discretize_state(state, env)
    done = False
    env.render()
    while not done:
        action = np.argmax(Q[state])
        result = env.step(action)
        if len(result) == 4:
            next_state, _, done, _ = result
        elif len(result) == 5:
            next_state, _, done, truncated, _ = result
            done = done or truncated

        if isinstance(next_state, tuple):
            next_state = next_state[0]
        state = discretize_state(next_state, env)
        env.render()
    env.close()

render_agent(env, Q)
```

```
Часть финальной Q-таблицы:
[[[ 4.10346871e-01  9.51190108e-01  8.37694980e-02]
  [ 8.13338292e-01 -4.62815959e-01 -1.07406326e-02]
  [-1.63976221e+00 -1.36354969e+00 -1.99446631e+00]
  [-5.47619459e+00 -5.07708385e+00 -4.68342925e+00]
  [-7.38160430e+00 -7.50824936e+00 -7.64920971e+00]]

 [[-8.03488895e-01  9.58507784e-01  9.10416351e-01]
  [-2.65672219e-01 -5.32579994e-01  9.07435573e-02]
  [-1.37788627e+00 -1.55387110e+00 -1.30610684e+00]
  [-4.20836197e+00 -4.02555680e+00 -4.48550870e+00]
  [-8.23755253e+00 -8.01898495e+00 -8.35089964e+00]]

 [[ 6.92030308e-01 -6.57448881e-01  9.69761113e-01]
  [-5.12971374e-01 -5.28642346e-01 -6.38010766e-01]
  [-3.06469990e+00 -3.58473890e+00 -3.61330378e+00]
  [-6.21666481e+00 -8.16697370e+00 -7.50786165e+00]
  [-1.21233564e+01 -1.23026291e+01 -1.21780363e+01]]

 [[-9.86809182e-01  6.03707596e-01  4.80934754e-01]
  [-1.48192286e+00 -1.29569038e+00 -1.32100678e+00]
  [-5.15397174e+00 -4.78767245e+00 -5.63793600e+00]
  [-1.07953018e+01 -1.09469044e+01 -1.10328386e+01]
  [-1.64261776e+01 -1.59265042e+01 -1.63977317e+01]]

 [[-3.06354952e-01 -1.64777028e-01  8.40792528e-01]
  [-2.05013190e+00 -2.10343654e+00 -2.17049891e+00]
  [-6.60935893e+00 -7.49738487e+00 -7.60543962e+00]
  [-1.37878056e+01 -1.35850139e+01 -1.37203186e+01]
  [-1.78432236e+01 -1.87901993e+01 -1.84280366e+01]]]
```

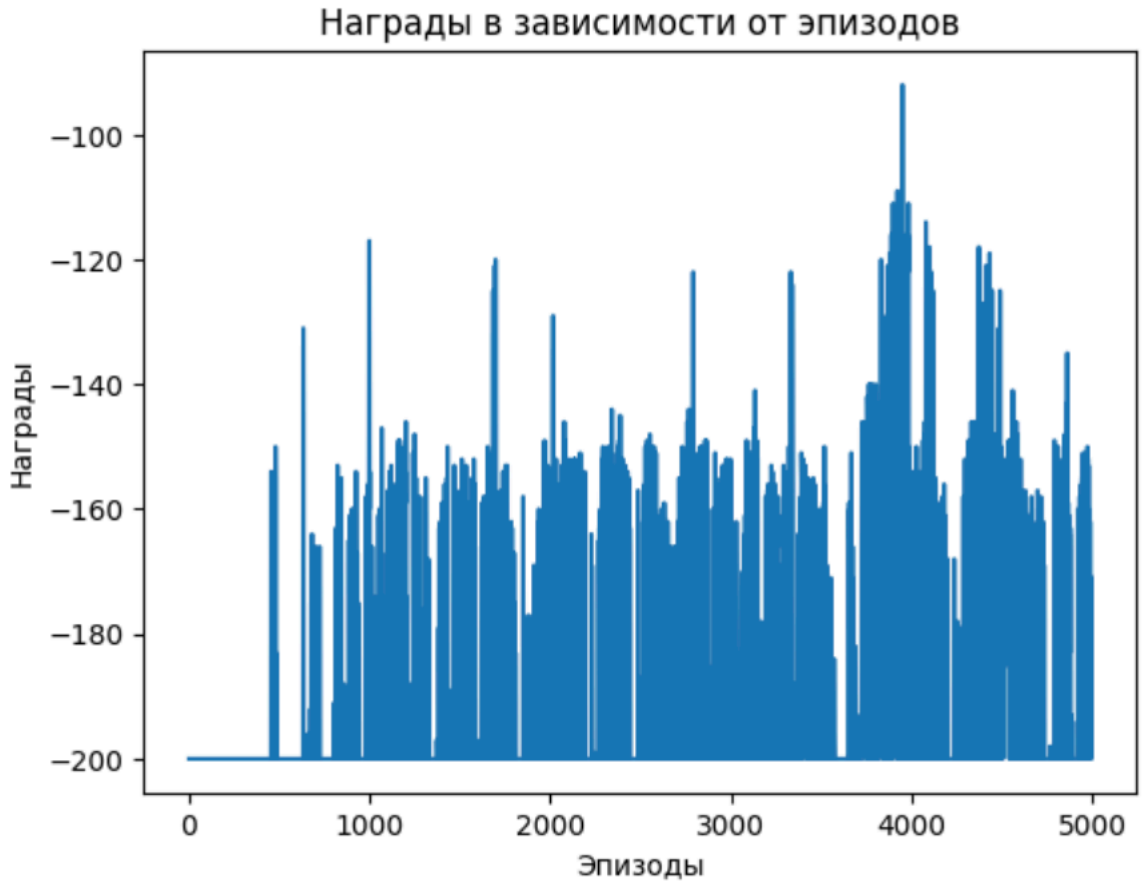
Алгоритм SARSA был успешно реализован и продемонстрировал свою эффективность в среде MountainCar-v0. Мы увидели, что агент способен обучаться и улучшать своё поведение с течением времени, что подтверждается ростом средней награды. Реализация SARSA и последующая визуализация результатов позволили нам глубже понять механизм работы алгоритмов обучения с подкреплением на основе временных различий.

2. Q-обучение

Q-обучение — это метод обучения с подкреплением, при котором агент учится оптимальной стратегии, максимизируя ожидаемое суммарное вознаграждение. Агент обновляет Q-значения, используя максимальное ожидаемое вознаграждение от следующего состояния, независимо от выбранного действия.

```
def q_learning(env, Q, n_states, n_actions, alpha, gamma, epsilon, episodes, test_episodes):
    rewards = []
    for episode in range(episodes):
        state = env.reset()
        if isinstance(state, tuple):
            state = state[0]
        state = discretize_state(state, env)
        total_reward = 0
        done = False

        while not done:
            action = choose_action(state, Q, n_actions, epsilon)
            result = env.step(action)
            if len(result) == 4:
                next_state, reward, done, _ = result
            elif len(result) == 5:
                next_state, reward, done, truncated, _ = result
                done = done or truncated
            
            if isinstance(next_state, tuple):
                next_state = next_state[0]
            next_state = discretize_state(next_state, env)

            best_next_action = np.argmax(Q[next_state])
            Q[state][action] += alpha * (reward + gamma * Q[next_state][best_next_action] - Q[state][action])

            state = next_state
            total_reward += reward

        rewards.append(total_reward)

        if (episode + 1) % 500 == 0:
            print(f"Эпизод {episode + 1}: Средняя награда за последние 500 эпизодов = {np.mean(rewards[-500:])}")
            test_rewards = test_agent(env, Q, n_states, n_actions, test_episodes)
            print(f"Средняя награда за тестовые эпизоды после {episode + 1} эпизодов обучения: {np.mean(test_rewards)}")

    return rewards

Q = initialize_q_table(n_states, n_actions)
rewards = q_learning(env, Q, n_states, n_actions, alpha=0.1, gamma=0.99, epsilon=0.1, episodes=5000, test_episodes=10)
plot_rewards(rewards)
display_q_table(Q)
render_agent(env, Q)
```

```
Эпизод 500: Средняя награда за последние 500 эпизодов = -199.554
Средняя награда за тестовые эпизоды после 500 эпизодов обучения: -200.0
Эпизод 1000: Средняя награда за последние 500 эпизодов = -197.302
Средняя награда за тестовые эпизоды после 1000 эпизодов обучения: -159.7
Эпизод 1500: Средняя награда за последние 500 эпизодов = -191.79
Средняя награда за тестовые эпизоды после 1500 эпизодов обучения: -189.9
Эпизод 2000: Средняя награда за последние 500 эпизодов = -189.226
Средняя награда за тестовые эпизоды после 2000 эпизодов обучения: -163.4
Эпизод 2500: Средняя награда за последние 500 эпизодов = -185.19
Средняя награда за тестовые эпизоды после 2500 эпизодов обучения: -199.6
Эпизод 3000: Средняя награда за последние 500 эпизодов = -183.1
Средняя награда за тестовые эпизоды после 3000 эпизодов обучения: -200.0
Эпизод 3500: Средняя награда за последние 500 эпизодов = -185.114
Средняя награда за тестовые эпизоды после 3500 эпизодов обучения: -155.6
Эпизод 4000: Средняя награда за последние 500 эпизодов = -177.756
Средняя награда за тестовые эпизоды после 4000 эпизодов обучения: -185.8
Эпизод 4500: Средняя награда за последние 500 эпизодов = -179.58
Средняя награда за тестовые эпизоды после 4500 эпизодов обучения: -150.0
Эпизод 5000: Средняя награда за последние 500 эпизодов = -183.868
Средняя награда за тестовые эпизоды после 5000 эпизодов обучения: -200.0
```



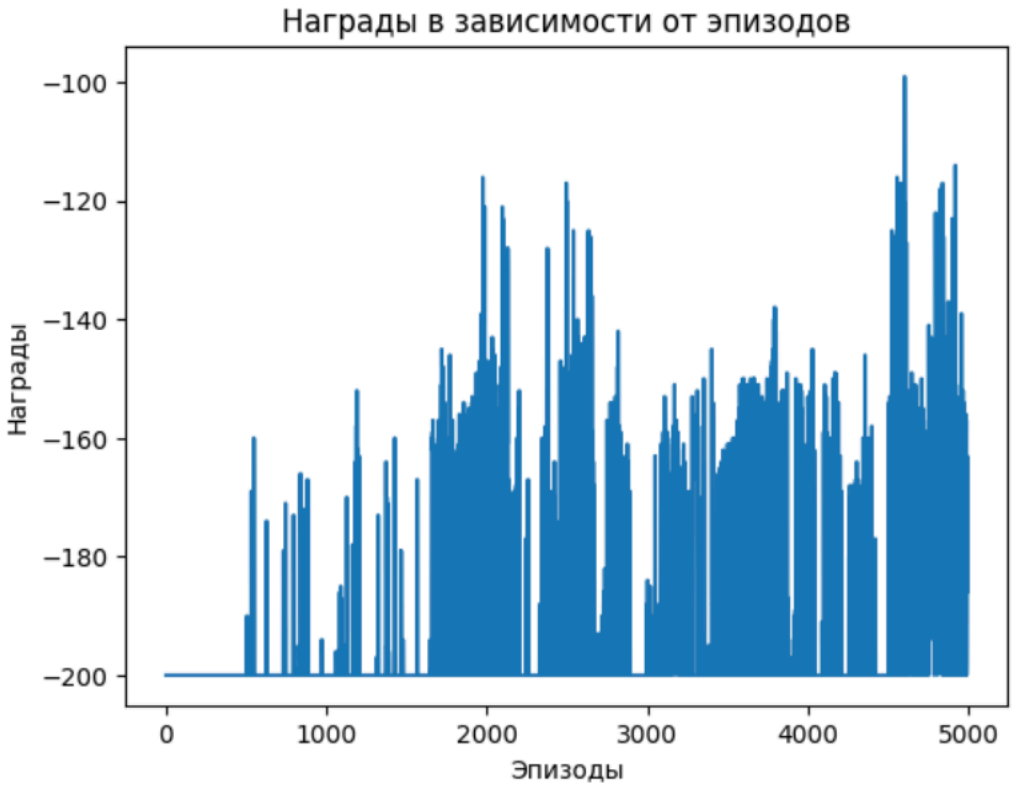
```
Часть финальной Q-таблицы:
[[[  0.07819393  -0.50385177   0.60325042]
  [ -4.97737743  -3.91560825  -0.88321722]
  [-46.60696098 -46.52996996 -46.48308803]
  [-48.76801421 -48.78644838 -48.75557106]
  [-48.70631114 -48.68994233 -48.69670188]]

 [[  0.3674971    0.9613001    0.70758544]
  [-16.8045504  -15.51797101 -16.98061167]
  [-48.123192   -48.21907135 -48.37874692]
  [-49.31706641 -49.32772321 -49.2881324 ]
  [-50.09805593 -50.08872457 -50.02914814]]

 [[  0.64279764  -0.917805    -0.49981466]
  [-41.90483031 -41.87888758 -42.66283923]
  [-49.24789722 -49.3949521  -49.3556532 ]
  [-50.68831496 -50.802676   -50.72661059]
  [-51.00078518 -51.12655147 -51.2692122 ]]

 [[ -3.64326456  -3.63476442  -0.94044614]
  [-44.95703209 -45.12729587 -45.39988395]
  [-50.49918567 -50.42724252 -50.54854437]
  [-51.9766483  -52.10538577 -52.104524  ]
  [-53.89783807 -54.03267724 -53.15058084]]

 [[ -4.30479681  -7.89725905  -7.32858516]
  [-47.71221633 -48.23398851 -48.09401637]
  [-52.05496708 -52.03487443 -52.07725392]
  [-52.94458126 -54.4542204  -54.26470328]
  [-55.61608977 -56.07892959 -56.21739146]]]
```

3. Двойное Q-обучение

Двойное Q-обучение — это метод обучения с подкреплением, который использует две отдельные Q-таблицы (Q1 и Q2) для уменьшения положительной предвзятости при оценке ожидаемого вознаграждения. В каждом шаге обучения одна из таблиц используется для выбора действия, а другая — для оценки вознаграждения, чередуя их на каждом шаге.

```python
def double_q_learning(env, Q1, Q2, n_states, n_actions, alpha, gamma, epsilon, episodes, test_episodes):
    rewards = []
    for episode in range(episodes):
        state = env.reset()
        if isinstance(state, tuple):
            state = state[0]
        state = discretize_state(state, env)
        total_reward = 0
        done = False

        while not done:
            action = choose_action(state, Q1 + Q2, n_actions, epsilon)
            result = env.step(action)
            if len(result) == 4:
                next_state, reward, done, _ = result
            elif len(result) == 5:
                next_state, reward, done, truncated, _ = result
                done = done or truncated

            if isinstance(next_state, tuple):
                next_state = next_state[0]
            next_state = discretize_state(next_state, env)

            if np.random.rand() < 0.5:
                best_next_action = np.argmax(Q1[next_state])
                Q1[state][action] += alpha * (reward + gamma * Q2[next_state][best_next_action] - Q1[state][action])
            else:
                best_next_action = np.argmax(Q2[next_state])
                Q2[state][action] += alpha * (reward + gamma * Q1[next_state][best_next_action] - Q2[state][action])

            state = next_state
            total_reward += reward

        rewards.append(total_reward)

        if (episode + 1) % 500 == 0:
            print(f"Эпизод {episode + 1}: Средняя награда за последние 500 эпизодов = {np.mean(rewards[-500:])}")
            test_rewards = test_agent(env, Q1 + Q2, n_states, n_actions, test_episodes)
            print(f"Средняя награда за тестовые эпизоды после {episode + 1} эпизодов обучения: {np.mean(test_rewards)}")

    return rewards

Q1 = initialize_q_table(n_states, n_actions)
Q2 = initialize_q_table(n_states, n_actions)
rewards = double_q_learning(env, Q1, Q2, n_states, n_actions, alpha=0.1, gamma=0.99, epsilon=0.1, episodes=5000, test_episodes=10)
plot_rewards(rewards)
display_q_table(Q1 + Q2)
render_agent(env, Q1 + Q2)
```

```
Эпизод 500: Средняя награда за последние 500 эпизодов = -200.0
Средняя награда за тестовые эпизоды после 500 эпизодов обучения: -200.0
Эпизод 1000: Средняя награда за последние 500 эпизодов = -199.158
Средняя награда за тестовые эпизоды после 1000 эпизодов обучения: -200.0
Эпизод 1500: Средняя награда за последние 500 эпизодов = -198.674
Средняя награда за тестовые эпизоды после 1500 эпизодов обучения: -200.0
Эпизод 2000: Средняя награда за последние 500 эпизодов = -189.792
Средняя награда за тестовые эпизоды после 2000 эпизодов обучения: -149.2
Эпизод 2500: Средняя награда за последние 500 эпизодов = -188.834
Средняя награда за тестовые эпизоды после 2500 эпизодов обучения: -160.8
Эпизод 3000: Средняя награда за последние 500 эпизодов = -185.426
Средняя награда за тестовые эпизоды после 3000 эпизодов обучения: -199.1
Эпизод 3500: Средняя награда за последние 500 эпизодов = -192.268
Средняя награда за тестовые эпизоды после 3500 эпизодов обучения: -197.8
Эпизод 4000: Средняя награда за последние 500 эпизодов = -180.968
Средняя награда за тестовые эпизоды после 4000 эпизодов обучения: -169.3
Эпизод 4500: Средняя награда за последние 500 эпизодов = -194.302
Средняя награда за тестовые эпизоды после 4500 эпизодов обучения: -200.0
Эпизод 5000: Средняя награда за последние 500 эпизодов = -172.688
Средняя награда за тестовые эпизоды после 5000 эпизодов обучения: -187.7
```




```
Часть финальной Q-таблицы:
[[[-1.78972718e-01  2.36773163e-01 -8.14828933e-01]
  [ 1.40237633e+00  1.12023836e+00  1.04732684e+00]
  [-7.23181355e+01 -7.28903929e+01 -7.25479491e+01]
  [-9.07958863e+01 -9.08108529e+01 -9.07185613e+01]
  [-9.14007354e+01 -9.14221068e+01 -9.14545529e+01]]

 [[ 1.93051879e-01  1.59047845e-01  5.34777898e-01]
  [-2.16802117e+00 -1.72577114e+00 -4.05843909e+00]
  [-8.21506733e+01 -8.17858710e+01 -8.19845170e+01]
  [-9.31243475e+01 -9.32616239e+01 -9.31305467e+01]
  [-9.42680076e+01 -9.44481747e+01 -9.40970201e+01]]

 [[-6.68909795e-01  7.99018440e-02  3.84408168e-01]
  [-3.77629830e+01 -3.83357911e+01 -3.87011006e+01]
  [-8.93744826e+01 -8.94663353e+01 -8.91173118e+01]
  [-9.60722955e+01 -9.62965197e+01 -9.62068793e+01]
  [-1.01574656e+02 -9.97205010e+01 -1.01962645e+02]]

 [[-5.14775885e-01  1.43707651e+00  5.50485477e-01]
  [-6.51767357e+01 -6.61128997e+01 -6.57644600e+01]
  [-9.36087453e+01 -9.34475233e+01 -9.34135765e+01]
  [-1.00313074e+02 -9.99963795e+01 -1.00278420e+02]
  [-1.06920790e+02 -1.06055225e+02 -1.07069588e+02]]

 [[ 5.73052499e-01 -8.15264436e-01  8.92077654e-01]
  [-7.52399450e+01 -7.78712383e+01 -7.89947908e+01]
  [-9.70134815e+01 -9.71103985e+01 -9.68461064e+01]
  [-1.03743948e+02 -1.05693921e+02 -1.05453198e+02]
  [-1.12558335e+02 -1.12582342e+02 -1.12814610e+02]]]
```

Вывод:

1. SARSA:

- Средние награды: Начиная с -199.668 и доходя до -182.998, SARSA показывает улучшение, но оно не является значительным.

- Тестовые награды: Большие колебания наград указывают на нестабильность стратегии. Например, тестовые награды после 2000 эпизодов улучшились до -172.9, но затем снова упали.

2. Q-обучение:

- Средние награды: Показатели улучшаются с -199.554 до -183.868, что свидетельствует о некотором прогрессе.

- Тестовые награды: Результаты также демонстрируют большие колебания, особенно в начале и конце обучения. Награды сильно варьируются от -200.0 до -150.0.

3. Двойное Q-обучение:

- Средние награды: Стабильное улучшение от -200.0 до -172.688. Это указывает на то, что агент последовательно улучшает свою стратегию.

- Тестовые награды: Результаты более стабильны по сравнению с другими алгоритмами. Например, тестовые награды после 2000 эпизодов улучшились до -149.2, и, несмотря на некоторые колебания, остаются стабильными.

Двойное Q-обучение обеспечивает наилучшие результаты благодаря своей устойчивости к переоценке вознаграждений и стабильности обучения.