Домашнее задание №4

4.1 Реализовать Q-Learning и сравнить его результаты с реализованными ранее алгоритмамы: Cross-Entropy, Monte Carlo, SARSA в задачу Taxi-v3. Для сравнения как минимум нужно использовать графики обучения.

При выполнении данного задания файлы .py названы по названию алгоритмов (за исклчением AcrobotDiscrete - дискретизированной среды), поскольку они используются и в 1-м и 2-м задании.

Импорт необходимых библиотек. Алгоритмы реализованы в соответствующих .py файлах.

In [2]:
```
import gym
from CrossEntropy import CrossEntropy
from MonteCarlo import MonteCarlo
from SARSA import SARSA
from Qlearning import Qlearning
import matplotlib.pyplot as plt
from AcrobotDiscrete import AcrobotDiscrete
import numpy as np
from DeepCrossEntropy import DeepCrossEntropy
```

Поскольку алгоритму кросс-энтропии необходимо минимум 6400 траекторий, запустим остальные алгоритмы с таким же количеством траекторий.

In [2]:
```
ce = CrossEntropy(env=gym.make("Taxi-v3"), q=0.5, n_trajectories=400, n_episode=16)
ce.fit()
```

In [3]:
```
mc = MonteCarlo(gym.make("Taxi-v3"), gamma=0.99, n_episode=6400, eps=1., eps_decay=0.995)
mc.fit()
```

In [10]:
```
sarsa = SARSA(gym.make("Taxi-v3"), gamma=0.99, alpha=0.5, n_episode=6400, n_episode_discount=400)
sarsa.fit()
```

In [11]:
```
q_learning = Qlearning(gym.make("Taxi-v3"), gamma=0.99, alpha=0.8, n_episode=6400, n_episode_discount=400)
q_learning.fit()
```

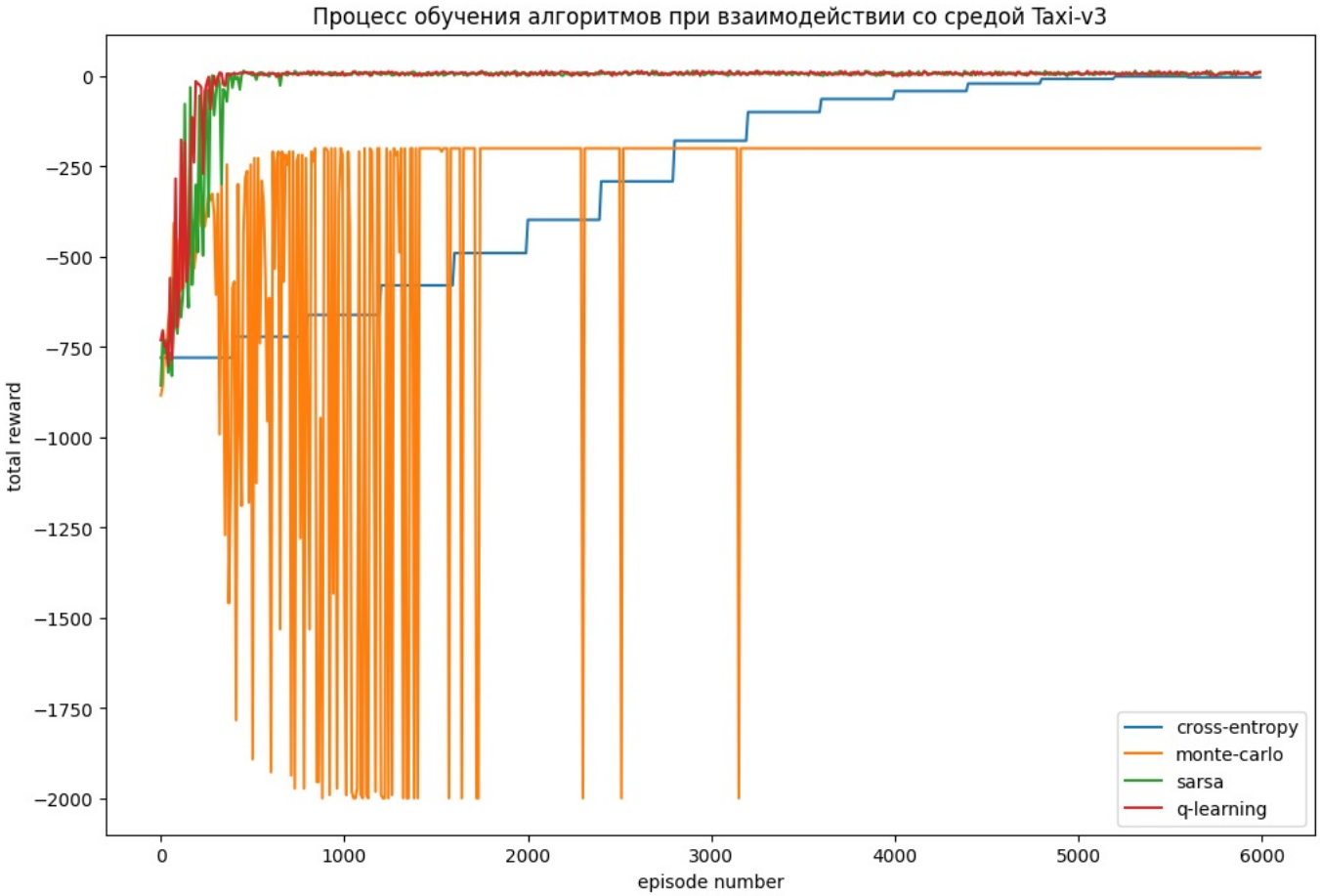
Награды для алгоритма кросс-энтропии необходимо преобразовать, поскольку для получения каждой оценки необходимо 400 траекторий.

In [12]:
```
ce_mean_total_rewards_6000 = []
for reward in ce.mean_total_rewards:
    for _ in range(400):
        ce_mean_total_rewards_6000.append(reward)
```

В целях приемлемой визуализации выведем на отображение награду на каждой 10 итерации

In [13]:
```
ce_mean_total_rewards = []
mc_mean_total_rewards = []
sarsa_mean_total_rewards = []
q_learning_mean_total_rewards = []
x = []
for i in range(6000):
    if i % 10 == 0:
        ce_mean_total_rewards.append(ce_mean_total_rewards_6000[i])
        mc_mean_total_rewards.append(mc.mean_total_rewards[i])
        sarsa_mean_total_rewards.append(sarsa.mean_total_rewards[i])
        q_learning_mean_total_rewards.append(q_learning.mean_total_rewards[i])
        x.append(i)
```

In [14]:
```
plt.figure(figsize = (12, 8))
plt.plot(x, ce_mean_total_rewards, label='cross-entropy')
plt.plot(x, mc_mean_total_rewards, label='monte-carlo')
plt.plot(x, sarsa_mean_total_rewards, label='sarsa')
plt.plot(x, q_learning_mean_total_rewards, label='q-learning')
plt.title('Процесс обучения алгоритмов при взаимодействии со средой Taxi-v3')
plt.xlabel('episode number')
plt.ylabel('total reward')
plt.legend()
plt.show()
```



Выводы по заданию 1:

По представленному графику видим, что алгоритмы SARSA и Q-learning сходятся примерно на 500 траекториях, алгоритм Monte Carlo сходится к локальному минимуму (не оптимальному решению) на 2500 траекториях, а алгоритм Cross-Entropy на 6000 траекторий. Это демонстрирует преимущество алгоритмов SARSA и Q-learning с точки зрения минимально необходимых траекторий при взаимодействии со средой.

4.2 Дискретизировать (можно использовать numpy.round()) пространство состояний и обучить агента решать CartPole-v1, Acrobot-v1, MountainCar-v0, или LunarLander-v2 (одну на выбор) методами Monte Carlo, SARSA и Q-Learning. Сравнить результаты этих алгоритмов и реализованного ранее алгоритма Deep Cross-Entropy на графиках.

В качестве среды выбрана Acrobot-v1.

Дискретизация реализована в классе AcrobotDiscrete (файл AcrobotDiscrete.py) и выполнена путем задания диапазонов в 10 шагов от нуля до максимума и до минимума. Таким образом (с учетом нуля) получили размерность 21 на каждую входную переменную и 21^6 для среды в целом. P.S. в процессе испытана среда CartPole-v1 и Acrobot-v1 с меньшей дискретизацией (размерность 11^6), результаты схожи с представленными.

Обучать будем на 10000 траекторий.

In [25]:
```
%%time
mc = MonteCarlo(AcrobotDiscrete(), gamma=0.9, n_episode=10000, eps=1., eps_decay=0.995)
mc.fit()
```

```
CPU times: user 30min 29s, sys: 3.87 s, total: 30min 33s
Wall time: 30min 33s
```

In [15]:
```
%%time
sarsa = SARSA(AcrobotDiscrete(), gamma=0.9, alpha=0.05, n_episode=10000, n_episode_discount=5000)
sarsa.fit()
```

```
CPU times: user 24min 19s, sys: 1.17 s, total: 24min 20s
Wall time: 24min 20s
```

In [16]:
```
%%time
q_learning = Qlearning(AcrobotDiscrete(), gamma=0.9, alpha=0.05, n_episode=10000, n_episode_discount=5000)
q_learning.fit()
```

```
CPU times: user 24min 58s, sys: 252 ms, total: 24min 58s
Wall time: 24min 58s
```

In [17]:
```
%%time
ce = CrossEntropy(AcrobotDiscrete(), q=0.5, n_trajectories=500, n_episode=20, is_print=False, init_model=np.arra
ce.fit()

ce_mean_total_rewards_10000 = []
for reward in ce.mean_total_rewards:
    for _ in range(500):
        ce_mean_total_rewards_10000.append(reward)
```

```
CPU times: user 3h 50min 45s, sys: 40.3 s, total: 3h 51min 26s
Wall time: 3h 51min 26s
```

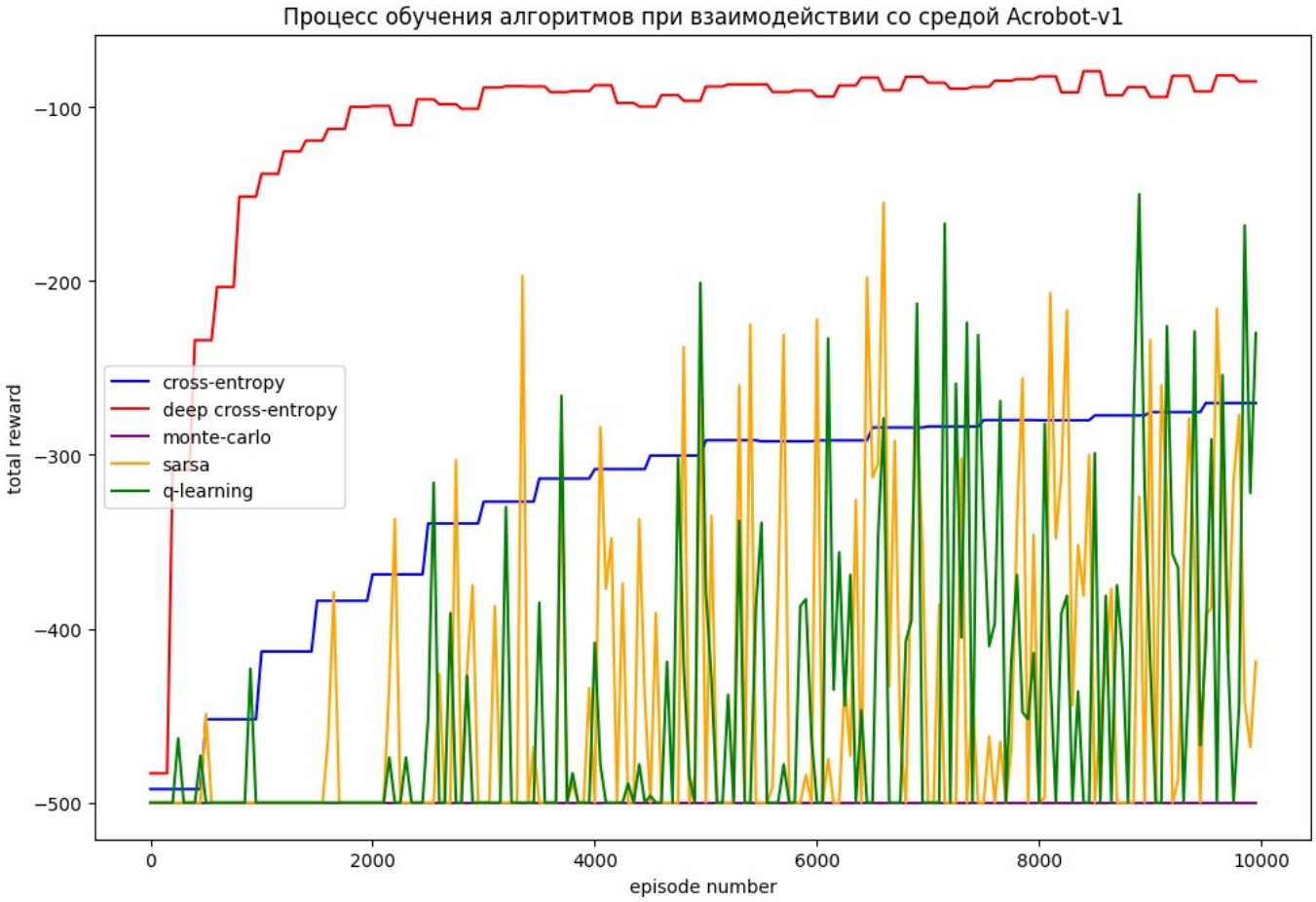
In [18]:
```
%%time
dce = DeepCrossEntropy(gym.make("Acrobot-v1"), q=0.9, n_trajectories=50, n_episode=200, n_neurons=128, eps=1,
                       lr=0.01, is_print=False, init_model=np.array([0.5, 0, 0.5]))
dce.fit()

dce_mean_total_rewards_10000 = []
for reward in dce.mean_total_rewards:
    for _ in range(200):
        dce_mean_total_rewards_10000.append(reward)
```

```
CPU times: user 4min 11s, sys: 220 ms, total: 4min 12s
Wall time: 4min 3s
```

In [26]:
```
ce_mean_total_rewards = []
dce_mean_total_rewards = []
mc_mean_total_rewards = []
sarsa_mean_total_rewards = []
q_learning_mean_total_rewards = []
x = []
for i in range(10000):
    if i % 50 == 0:
        ce_mean_total_rewards.append(ce_mean_total_rewards_10000[i])
        dce_mean_total_rewards.append(dce_mean_total_rewards_10000[i])
        mc_mean_total_rewards.append(mc.mean_total_rewards[i])
        sarsa_mean_total_rewards.append(sarsa.mean_total_rewards[i])
        q_learning_mean_total_rewards.append(q_learning.mean_total_rewards[i])
        x.append(i)
```

In [29]:
```
plt.figure(figsize = (12, 8))
plt.plot(x, ce_mean_total_rewards, label='cross-entropy', color='b')
plt.plot(x, dce_mean_total_rewards, label='deep cross-entropy', color='r')
plt.plot(x, mc_mean_total_rewards, label='monte-carlo', color='purple')
plt.plot(x, sarsa_mean_total_rewards, label='sarsa', color='orange')
plt.plot(x, q_learning_mean_total_rewards, label='q-learning', color='g')
plt.title('Процесс обучения алгоритмов при взаимодействии со средой Acrobot-v1')
plt.xlabel('episode number')
plt.ylabel('total reward')
plt.legend()
plt.show()
```



Выводы по заданию 2:

- лучший алгоритм - алгоритм на среде без дискредитации deep cross-entropy, он сходится к оптимальному значению наград до 2000 итераций
- алгоритм кросс энтропии (не глубокий) на дискретизированной среде имеет тенденцию к росту и имеет потенциал к увелечению награды при дальнейшем увеличении количества итераций (но испытывать не хотелость, итак ждать пришлось около 4 часов)
- алгоритмы SARSA и Q-Learning показывают примерно схожую картину: имеют положительный тренд, но при этом все равно в достаточном количестве траекторий имеем награду -500
- алгоритм Monte-Carlo вообще не приблизился к решению задачи

В итоге делаем вывод о плохом решении данного класса задач алгоритмами Monte-Carlo, SARSA и Q-Learning и ждем алгоритм DQN.

4.3 Придумать стратегию для выбора epsilon позволяющую агенту наилучшим образом решать Taxi-v3 алгоритмом Monte Carlo.

Применено 3 варианта формирования epsilon:

1. $eps = 1 / (episode + 1)$, где episode - шаг итерации
2. $eps = 1 - episode / nEpisodes$, где episode - шаг итерации, nEpisodes - всего итераций.
3. $eps = eps * epsDecay$, где eps_decay - коэффициент убывания epsilon, $epsDecay \in (0, 1)$

In [10]:
```
%%time
mc_1 = MonteCarlo(gym.make("Taxi-v3"), gamma=0.99, n_episode=10000, eps_version=0)
mc_1.fit()
```

```
CPU times: user 1min 46s, sys: 176 ms, total: 1min 46s
Wall time: 1min 46s
```

In [14]:
```
%%time
mc_2 = MonteCarlo(gym.make("Taxi-v3"), gamma=0.99, n_episode=10000, eps_version=1)
mc_2.fit()
```

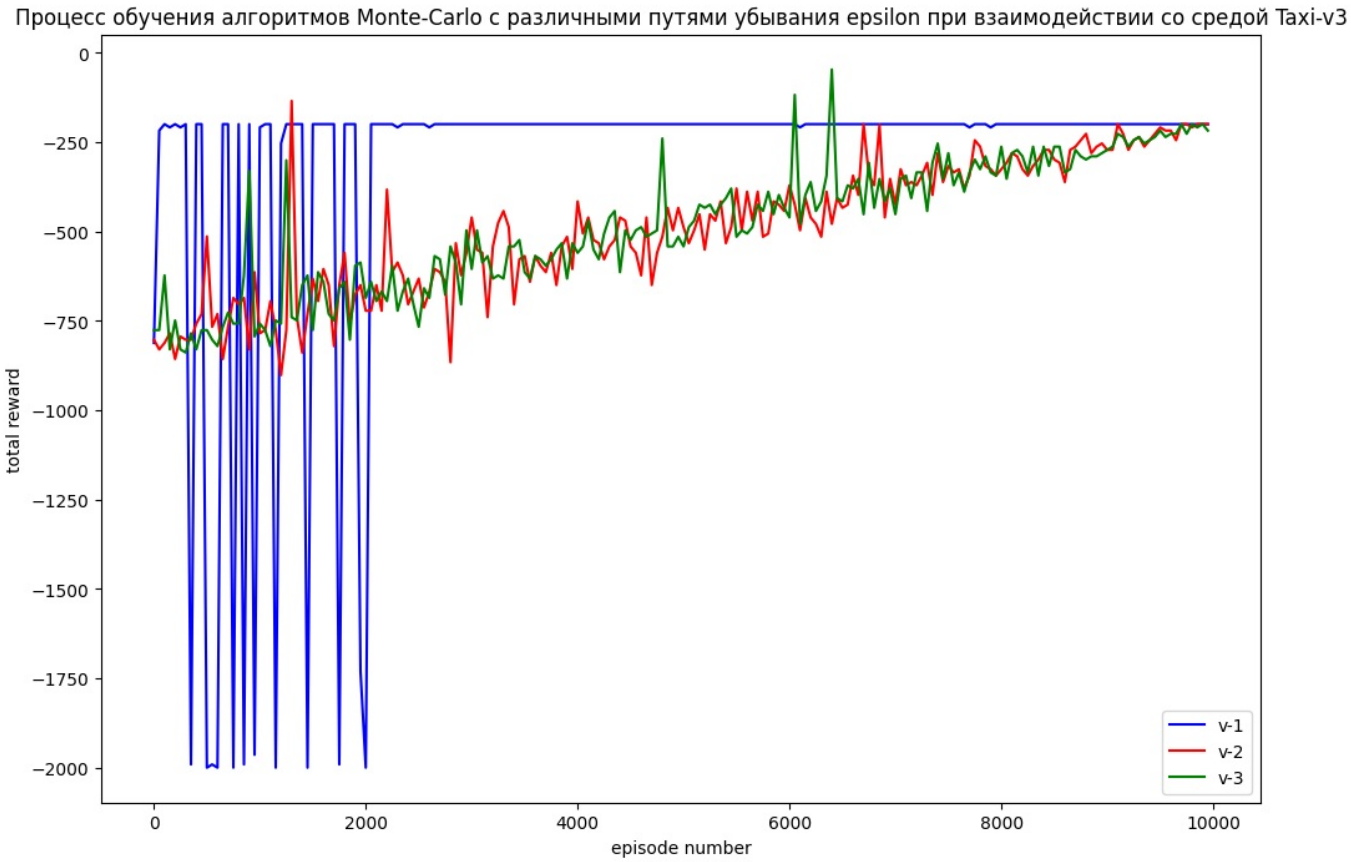
```
CPU times: user 1min 46s, sys: 84.1 ms, total: 1min 46s
Wall time: 1min 46s
```

In [15]:
```
%%time
mc_3 = MonteCarlo(gym.make("Taxi-v3"), gamma=0.99, n_episode=10000, eps_version=1)
mc_3.fit()
```

```
CPU times: user 1min 45s, sys: 144 ms, total: 1min 45s
Wall time: 1min 45s
```

In [16]:
```
mc1_mean_total_rewards = []
mc2_mean_total_rewards = []
mc3_mean_total_rewards = []
x = []
for i in range(10000):
    if i % 50 == 0:
        mc1_mean_total_rewards.append(mc_1.mean_total_rewards[i])
        mc2_mean_total_rewards.append(mc_2.mean_total_rewards[i])
        mc3_mean_total_rewards.append(mc_3.mean_total_rewards[i])
        x.append(i)
```

In [17]:
```
plt.figure(figsize = (12, 8))
plt.plot(x, mc1_mean_total_rewards, label='v-1', color='b')
plt.plot(x, mc2_mean_total_rewards, label='v-2', color='r')
plt.plot(x, mc3_mean_total_rewards, label='v-3', color='g')
plt.title('Процесс обучения алгоритмов Monte-Carlo с различными путями убывания epsilon при взаимодействии со ср
plt.xlabel('episode number')
plt.ylabel('total reward')
plt.legend()
plt.show()
```



Выводы по заданию 3:

Ни однин из примененных убываний epsilon не обеспечило достижения оптимального значения наград (сошлись к награде -250), при этом лучшую скорость сходимости показал 1-й вариант $eps = 1 / (episode + 1)$.

In [ ]: