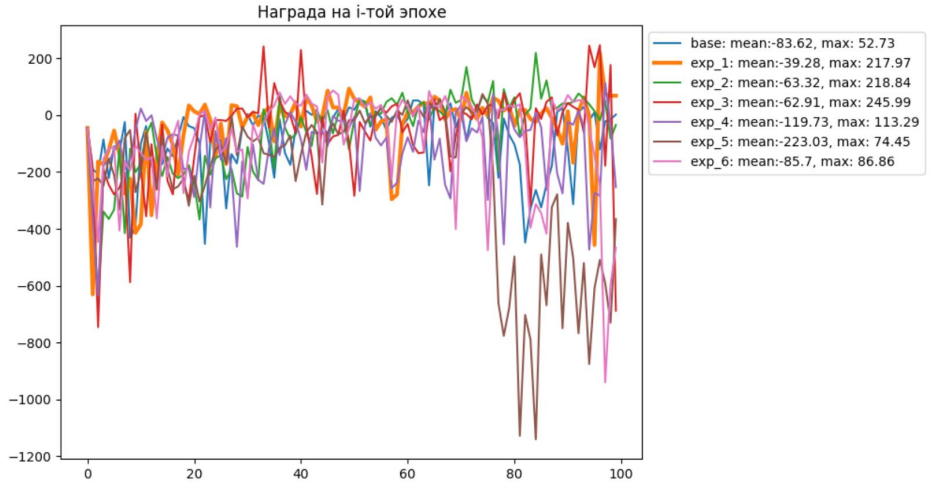
Задание 5.

Task_1. (Обучить Агента решать Acrobot-v1, MountainCar-v0, или LunarLander-v2 (одну на выбор) методом DQN. Найти оптимальные гиперпараметры. Сравнить с алгоритмом Deep Cross-Entropy на графиках.)

Я выбраз задачу LunarLander-v2 и провел несколько экспериментов с базовыми параметрами: Но, перед запуском кода, я провел эксперимент о кол-ве операций которые хранятся в переменной memory_size. Прочтя вспомогательный материал в интернете, я изменил кол-во итераций, которые способны храниться в памяти до 15000, так как неуверен, что есть необходимость хранить все попытки в памяти. (А их очень много накапливается к 100ому эпизоду).

```
episode: 83, memory_size: 29693
episode: 84, memory_size: 30193
episode: 85, memory_size: 30693
episode: 86, memory_size: 31193
episode: 87, memory_size: 31693
episode: 88, memory_size: 32193
episode: 89, memory_size: 32693
episode: 90, memory_size: 33193
episode: 91, memory_size: 33693
episode: 92, memory_size: 34193
episode: 93, memory_size: 34693
episode: 94, memory_size: 35193
episode: 95, memory_size: 35693
episode: 96, memory_size: 36193
episode: 97, memory_size: 36693
episode: 98, memory_size: 37193
episode: 99, memory_size: 37693
```

Номер эксперимента	***gamma***	***lr***	***Batch_size***	***Epsilon_descrease***	***Epilon_min***	***Max награда?***
base	0.99	0.001	64	0.01	0.01	52.7286285494651
Exp_1	0.99	0.001	64	0.03	0.03	217.97083258482144
Exp_2	0.99	0.001	128	0.03	0.03	218.8449635335151
Exp_3	0.99	0.001	128	0.05	0.05	245
Exp_4	0.7	0.001	128	0.02	0.02	113.28577555064831
Exp_5	0.99	0.002	128	0.025	0.025	74.44871860763892
Exp_6 (128 нейронов сеть)	0.99	0.002	128	0.025	0.025	86.8571943965004

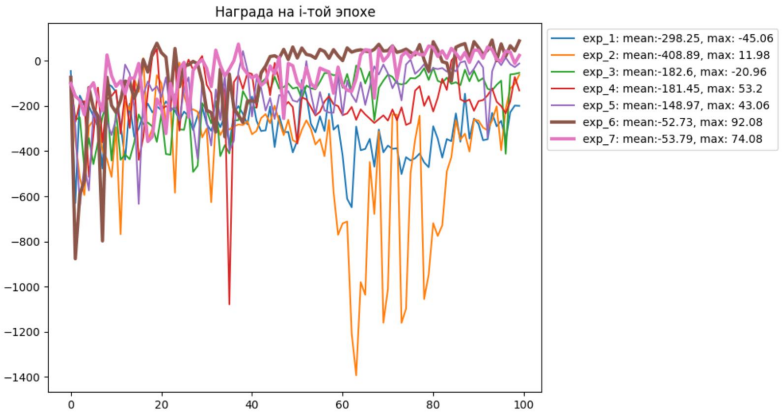


Получается, что exp_1 был самым стабильным (среднее значение) а exp_3 принес наибольшие результаты.

Task_2.

DQN_HardTargetUpdate. Для эксперемента возьмем параметры наилучшего обучения из DQN (exp_1) И попробуем подтюнить частоту обновления модели.

Номер эксперимента	***gamma***	***lr***	***Batch_size***	***Epsilon_decsrease***	***Epilon_min***	***Max награда?***	***Update_TIME***
Exp_1	0.99	0.001	64	0.03	0.03	-45.05	10
Exp_2	0.99	0.001	64	0.03	0.03	11.98	30
Exp_3	0.99	0.001	64	0.03	0.03	-20.95	50
Exp_4	0.99	0.001	64	0.03	0.03	53.20	100
Exp_5	0.99	0.001	64	0.03	0.03	43.05	150
Exp_6	0.99	0.001	64	0.03	0.03	92.07	200
Exp_7	0.99	0.001	64	0.03	0.03	74.08	250



Довольно интересный эксперимент. Мы недостигли максимальных значений как в первом случае, но мы видим, что в случае exp_6 и exp_7 графики вышли в итоге на довольно «стабильное плато», что показывает положительную тенденцию. Интерсно, что в этих двух эксперементах использовался наибольший step для обновления весов в основной модели.

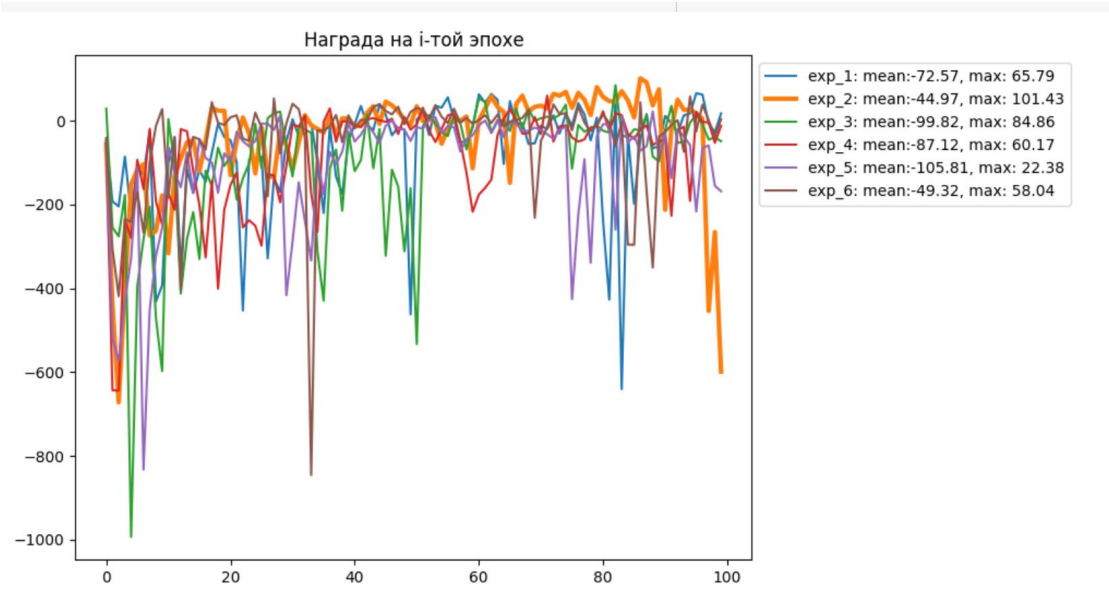
DQN_SOFTTargetUpdate.
Первое, мне пришлось переписать метод инициализации второй сети. Если инициализировать стартово две модели при зафиксированном seed, то базовые веса будут отличаться в моделях. (См. Скриншоты)

```
        [-0.1110,  0.0277, -0.1206, -0.0410,  0.0138,  0.0751,  0.1089, -0.0779,
         -0.0609, -0.0511,  0.0965,  0.0437,  0.1054, -0.0657, -0.0271,  0.0624,
         -0.0405,  0.0246, -0.0946, -0.0699, -0.1058,  0.0201,  0.0011,  0.0880,
         -0.1078, -0.0306,  0.1146,  0.0887, -0.0704,  0.0627,  0.0417, -0.0069,
         -0.0398, -0.0892,  0.0111,  0.1003, -0.0469, -0.0776,  0.0216,  0.0448,
          0.0374, -0.1112,  0.1178, -0.0723, -0.0005,  0.1023,  0.0835,  0.0432,
          0.0366,  0.0903,  0.1159, -0.0643, -0.0152,  0.0073, -0.0003, -0.0771,
          0.0157,  0.0640, -0.0747, -0.0702,  0.0929, -0.0688, -0.0998,  0.0416]])),
 ('linear_3.bias', tensor([ 0.0546, -0.1069,  0.0528, -0.0348]))])
```

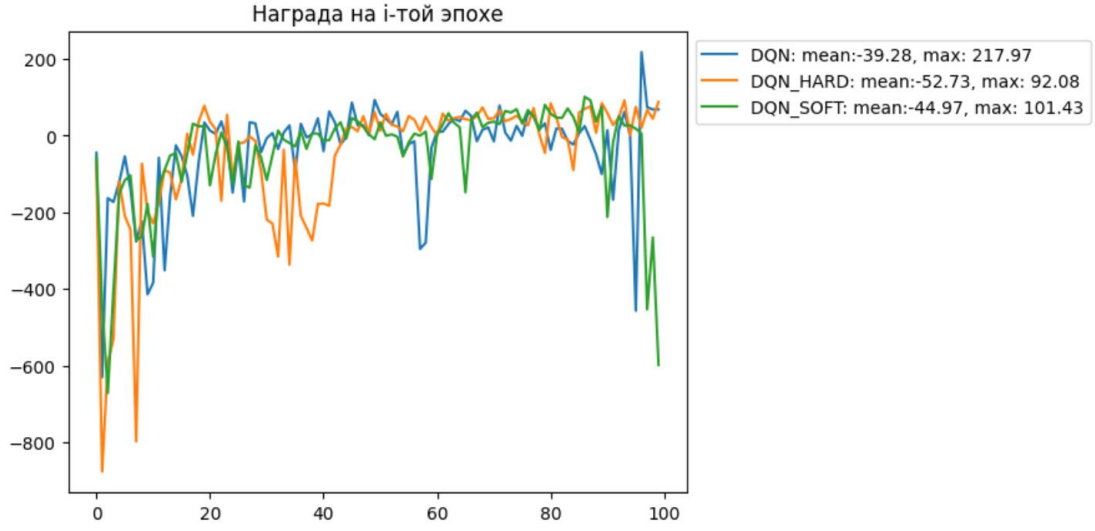
```
        [-0.0807, -0.0119,  0.1103,  0.1173,  0.1122,  0.0898, -0.0232,  0.0800,
          0.1074, -0.0600,  0.1005, -0.0068, -0.0999,  0.0601,  0.1211,  0.0314,
         -0.0046,  0.1238, -0.0468,  0.0877, -0.0149,  0.0442, -0.1235,  0.0700,
          0.1247,  0.0559, -0.0970, -0.0640, -0.0255,  0.1104, -0.0042,  0.0598,
         -0.1029, -0.0795, -0.0129,  0.0258,  0.0601,  0.0609,  0.0705, -0.0431,
          0.0271, -0.0979,  0.0159,  0.0804, -0.0740, -0.0514,  0.0234,  0.0688,
         -0.0069,  0.0457,  0.0726,  0.0719, -0.0227, -0.0033, -0.0850,  0.0435,
          0.0321, -0.0799,  0.1006, -0.0463, -0.1190, -0.1071,  0.0350, -0.0839]])),
 ('linear_3.bias', tensor([-0.0420, -0.0177,  0.0263, -0.0924]))])
```

Поэтому пришлось при инициализации второй сети. Наследоваться от первой
Я выбрал опять теже базовые параметры, и менял только коэффициент изменения весов (thau) .

Номер эксперимента	***gamma***	***lr***	***Batch_size***	***Epsilon_decrease***	***Epilon_min***	***Max награда?***	***coefficient***
Exp_1	0.99	0.001	64	0.03	0.03	65.78	0.001
Exp_2	0.99	0.001	64	0.03	0.03	101.43036	0.005
Exp_3	0.99	0.001	64	0.03	0.03	84.85671076905162	0.01
Exp_4	0.99	0.001	64	0.03	0.03	60.173	0.02
Exp_5	0.99	0.001	64	0.03	0.03	22.38142	0.015
Exp_6	0.99	0.001	64	0.03	0.03	58.03592833255635	0.1



Построим общий график, лучших попыток из каждой серии



По данному графику сложно определить, какой алгоритм оказался наилучшим. Я бы сказал, что DQN_HARD показался мне наиболее стабильным алгоритмом для выполнения данной задачи. Хотя пиковых значений мы смогли достигнуть только благодаря DQN графику. Возможно, нам стоило увеличить кол-во итераций, и тогда мы смогли бы получить лучшие решения.