The game I chose was “Acrobot-v1”

To very briefly explain the this game, on the OpenAI Gym documentation, it describes: “two blue links connected by two green joints. The joint in between the two links is actuated. The goal is to swing the free end of the outer-link to reach the target height (black horizontal line above system) by applying torque on the actuator.”
The actions are: apply -1, 0, or 1 torque to the actuated joint. Size 3
The state/observation space consists of the angles or the links and their velocities. Size 6
Every action has a reward of -1. Achieving the target height has a reward of 0. Minimum reward is when the game times out (500 steps = -500 reward).

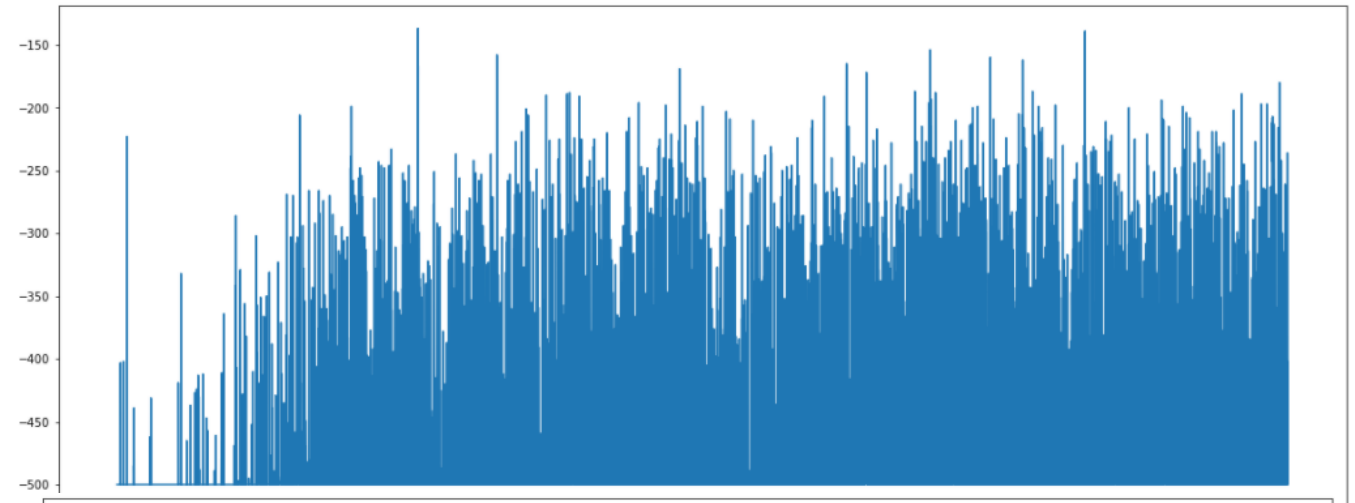
Before training the agent, I tested a random agent. That is, the agent takes completely random actions. Not surprisingly, this random agent had a very hard time completing the game (in fact, it never did). All trials have a total reward of -500, the minimum.

To train the agent, I first created a Q table with random numbers within the state space and action space. Then I discretized the state space for easier manipulation (locating specific values in the Q table). The training process follows Q-learning with epsilon greedy policy. Actions are chosen based on the condition: if np.random.random() > epsilon, choose the action that maximizes reward. Else, choose a random action. Then, the Q-table gets updated according to the Bellman equation. Finally, epsilon decays at a given rate.

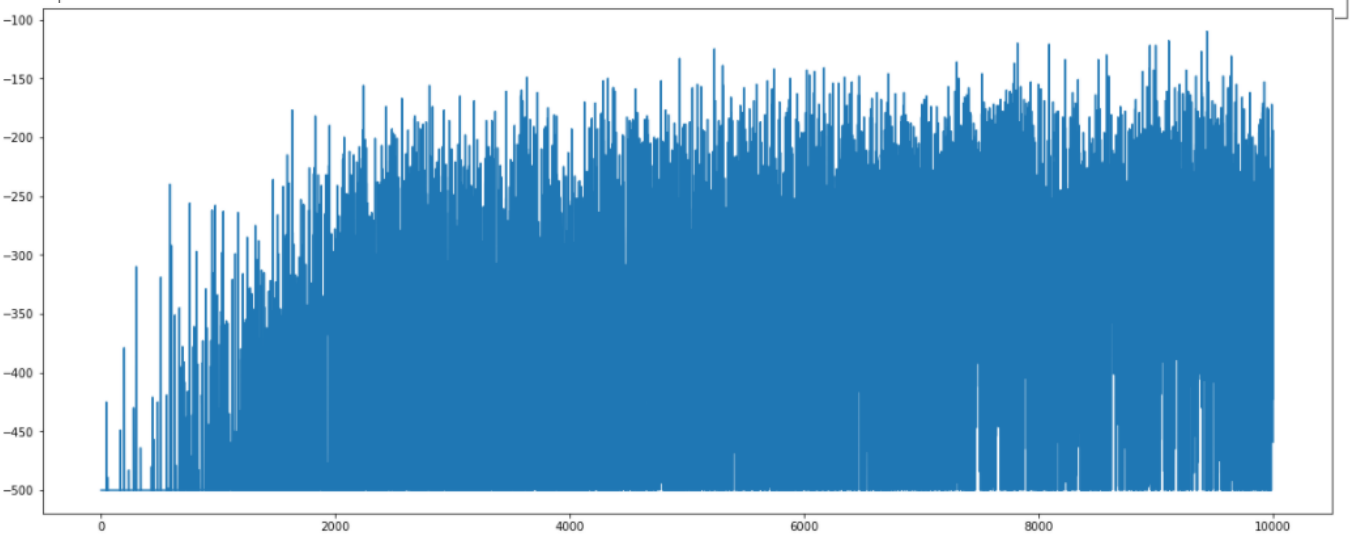
The rate of epsilon decay I set is constant:
epsDecayValue = epsilon / (Episodes // decayRate - 1).
epsilon = 1
Episodes, decayRate are given parameters.

Reward vs. Episode plot for learning rate = 0.05, 0,5 and 1, respectively, from top to bottom (everything else constant):

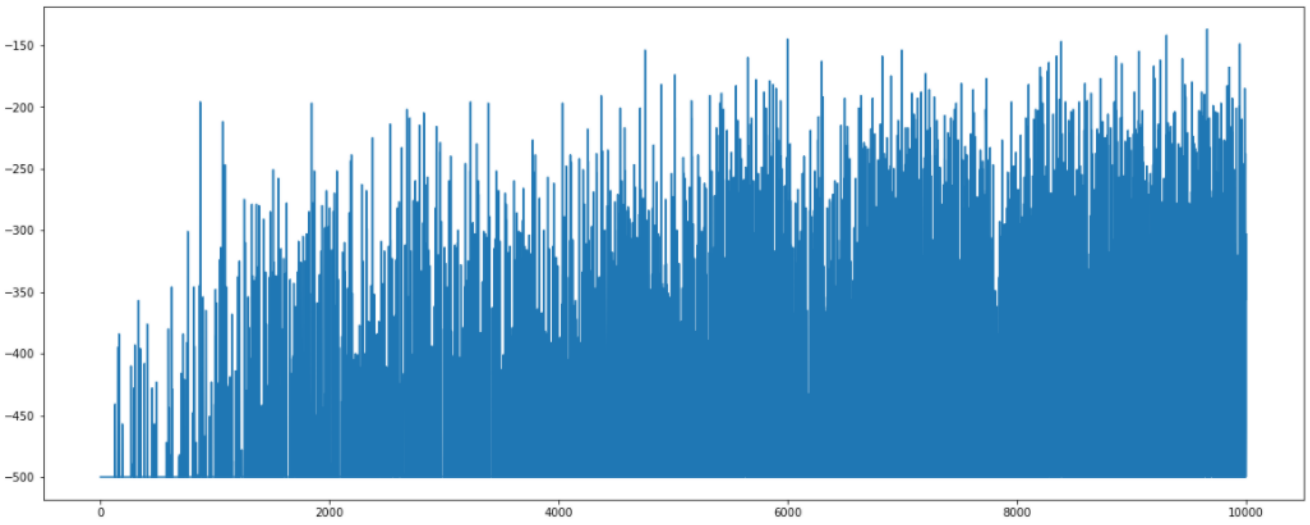
0.05:



0.5:



1:

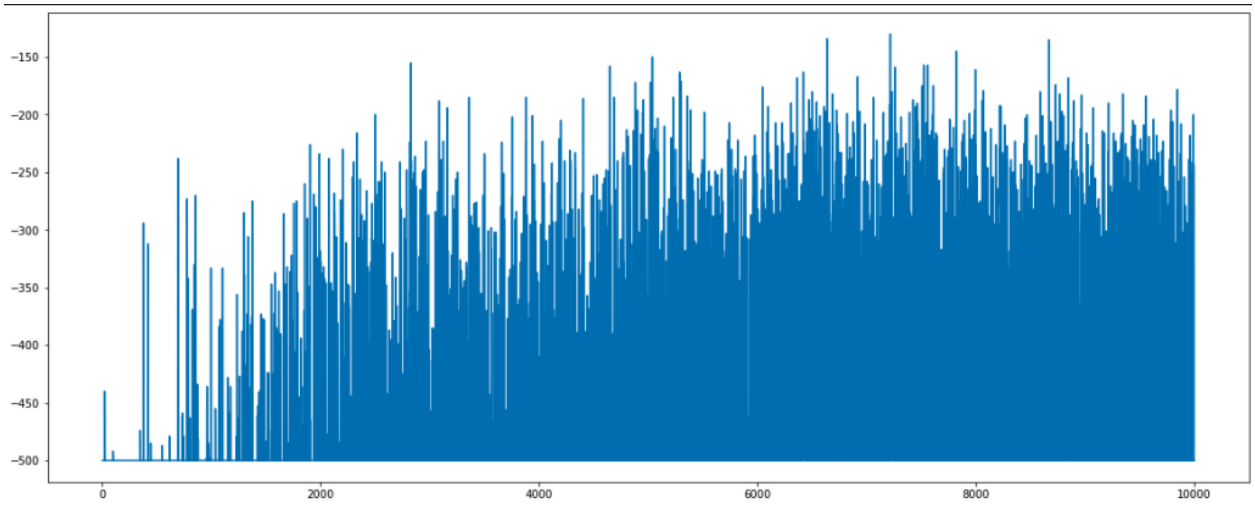


The “density” of the lines can illustrate the effect of learning rate. The more dense the plot, the more times the reward is -500. At learning rate 0.05 and 1, the agent achieves high reward much less frequently than 0.5 and they have larger fluctuations than 0.5, indicating that the learning rate was too small and too large, respectively. Furthermore, learning rate of 0.5 also produces higher maximum rewards, as indicated by the yaxis.

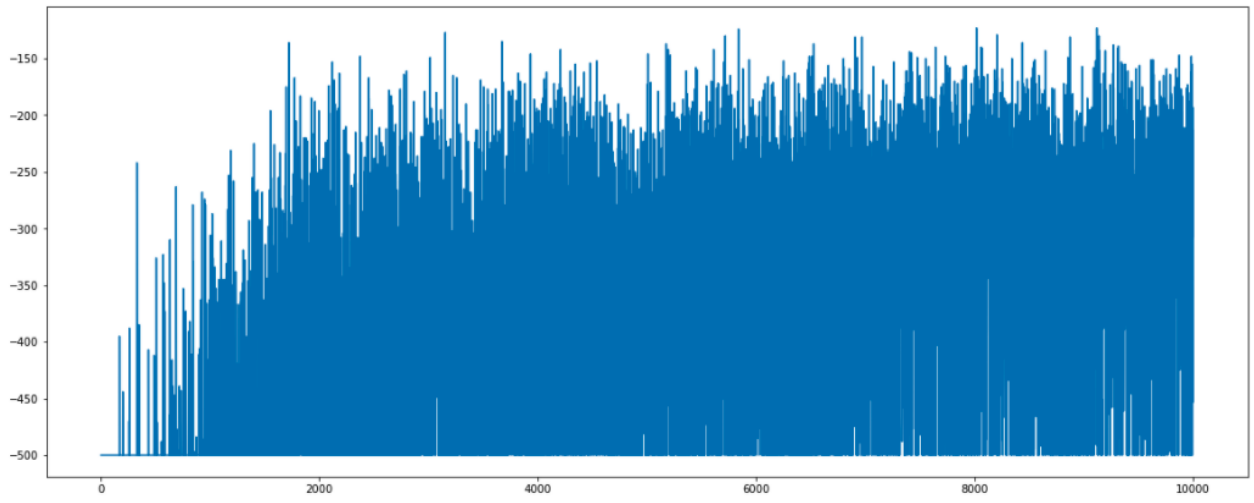
Intuitively, the discount rate should be set high because the agent only gets a non-negative reward when achieving the goal, so the agent should focus more on future reward.

Reward vs. Episode plot for discount = 0.01, 0.99, respectively, from top to bottom (everything else constant):

0.01:



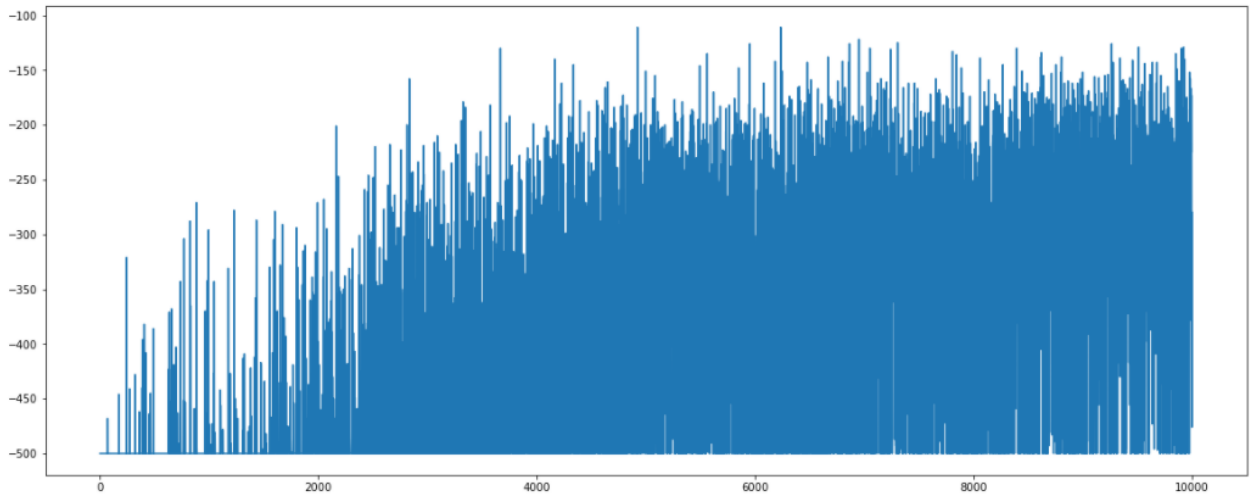
0.99



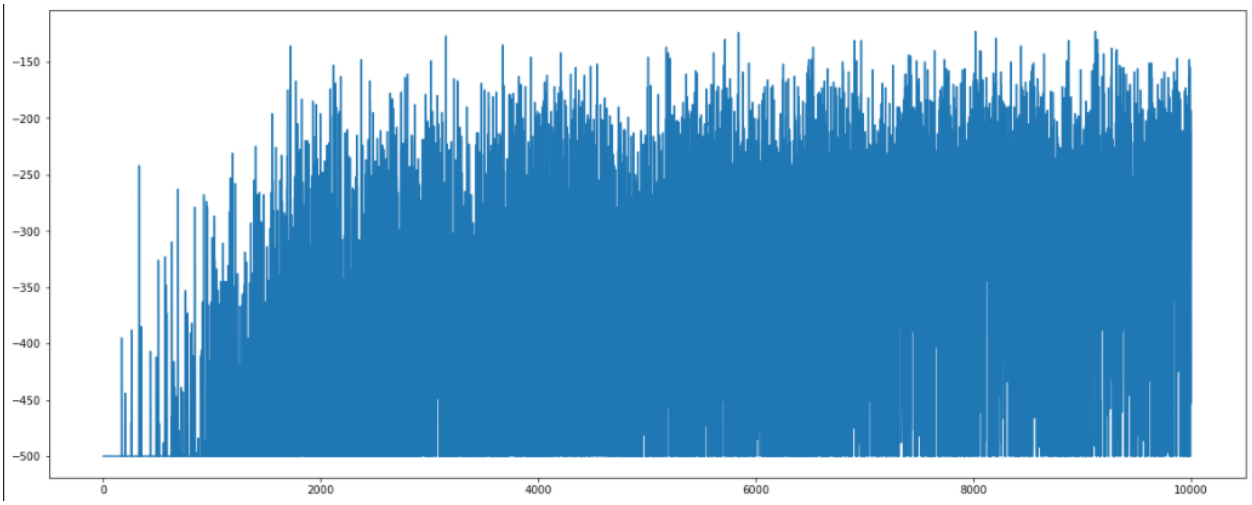
Setting a high discount value is better than low, as demonstrated in the difference of the density.

Reward vs. Episode plot for decayRate = 2, 5, 1000 respectively, from top to bottom (everything else constant):

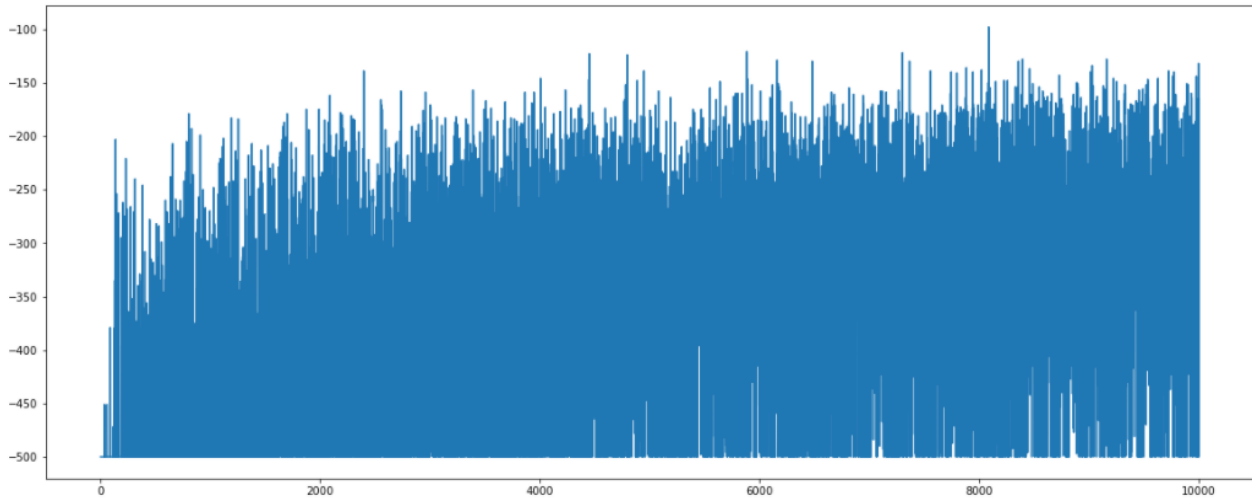
2:



5:

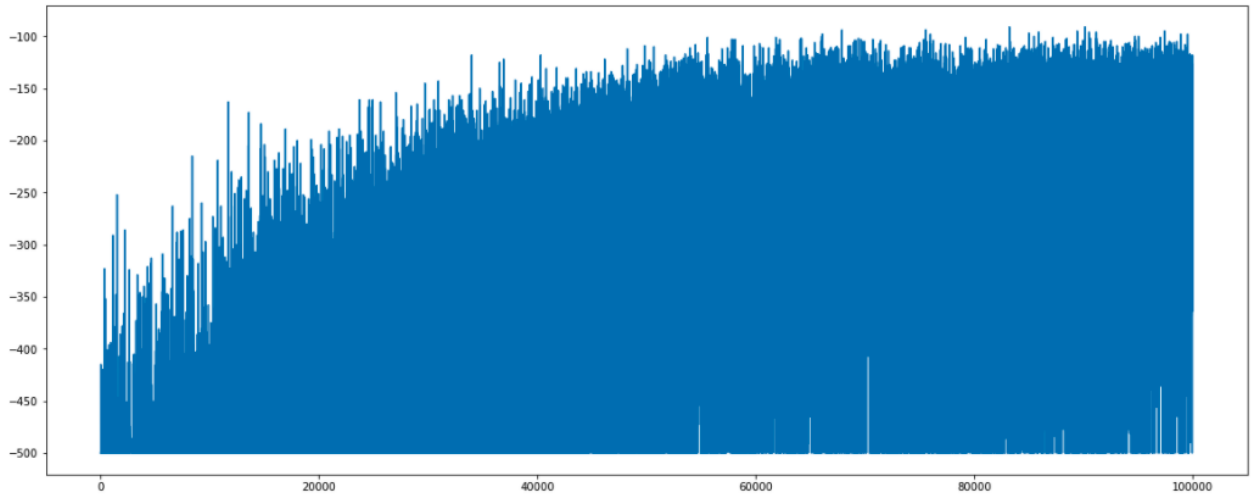


1000:

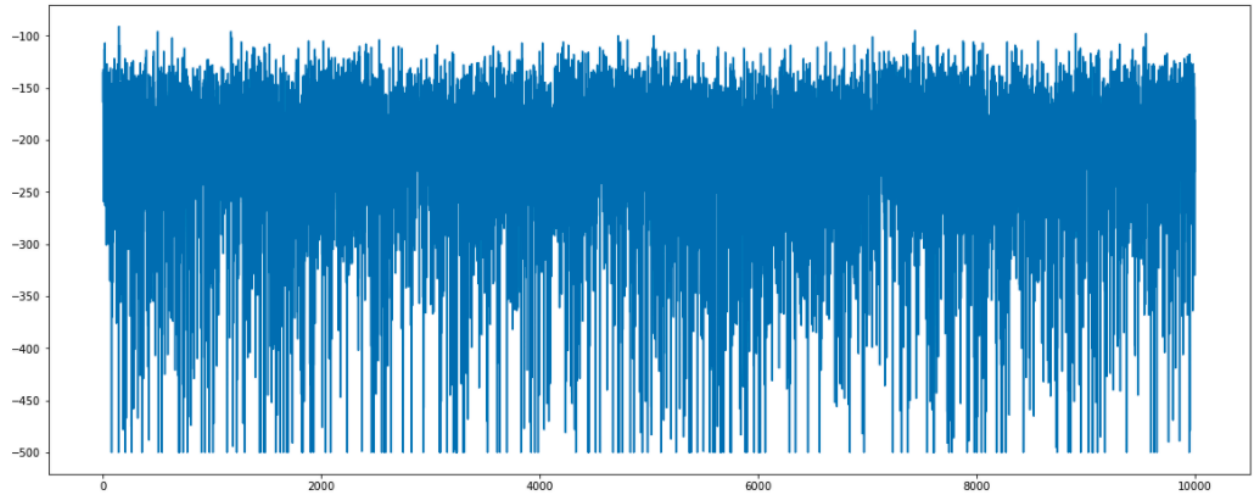


Here we can see the explore/exploit tradeoff. As we increase the rate of decay, the agent explores less frequently quicker. This is demonstrated in the general trends of the plots. When decayRate = 2, the exploring phase (low density in plot) is longer, and arrives at a relatively stable rewards slower than the other decayRates. However, we can see that the relatively stable rewards of decayRate = 2 and 1000 are very similar. This suggests that the sample space of "good actions" of Acrobot is relatively small.

Then, I allow the agent to train for 100,000 games (it took 3 hours!) to see if the reward can converge:



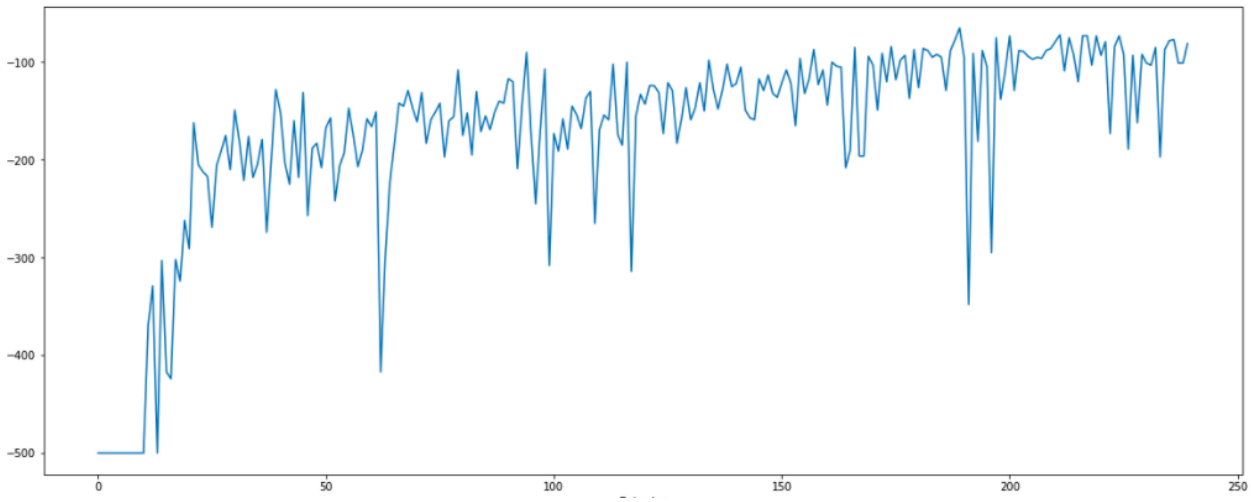
Last 10,000 episodes:



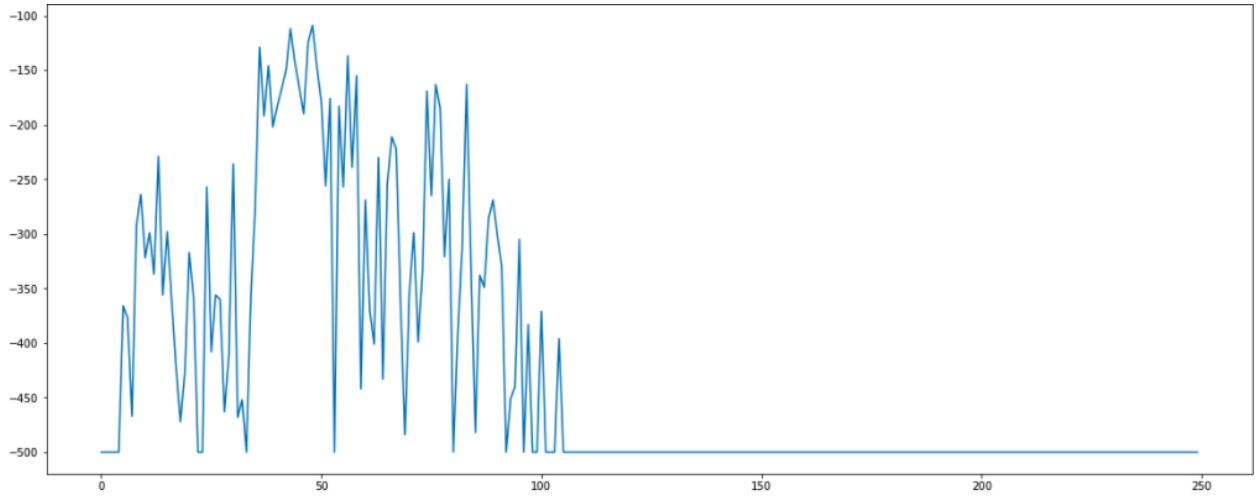
Although we can see the sign of convergence: the last 10,000 episodes have much less variation than any previous trainings, the agent is still receiving the minimum reward frequently.

It was very challenging for the agent to avoid the actions that lead to the worst results. *Perhaps this is due to the fact that every action comes with negative rewards, so "good" rewards come very infrequently.*

This section is included only for completeness, not to show what I did for this HW.
Finally, just out of curiosity, I slightly modified and ran the model from [this source] to see how gradient policy can enhance the agent's performance. The result was stunning (to me):



In less than 250 episodes, the reward reached above -100.
However, there is a lot of randomness in this model.



For example, I ran the model again and this was the result.