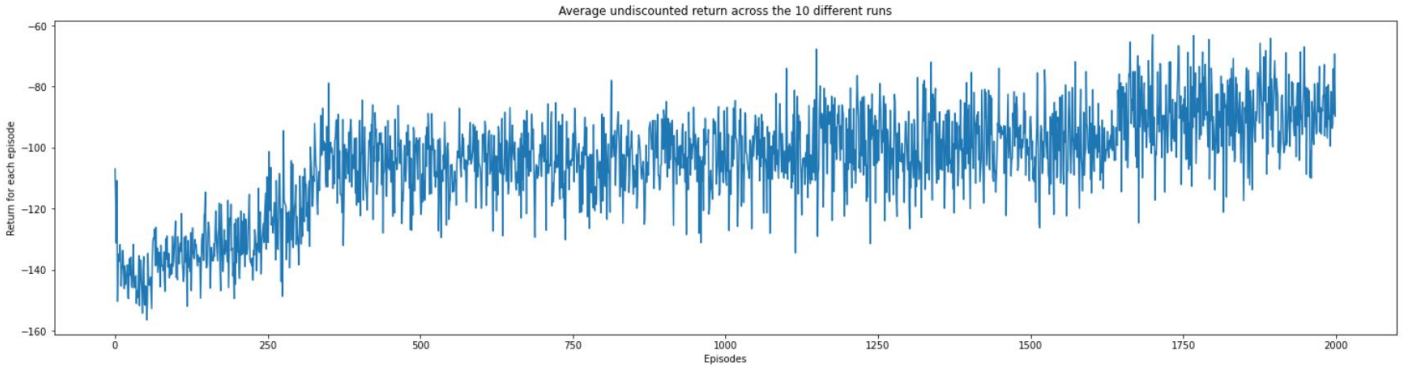
Report
INF8953DE (Fall 2021) : Reinforcement Learning - Assignment #2
Amine EL AMERI - Matricule: 2164634

1 Monte Carlo Methods

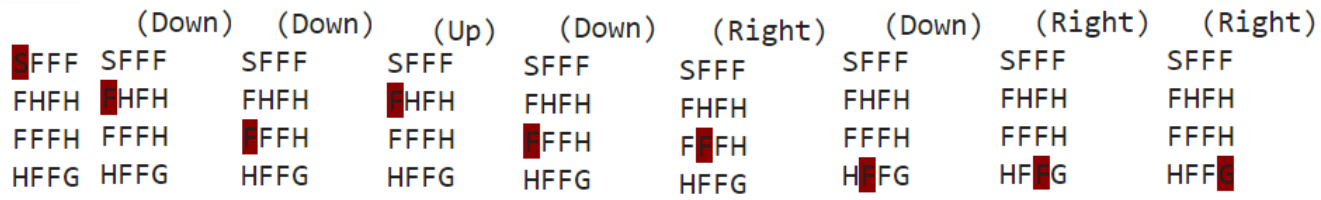
1- I implemented the generate_episode function, taking a policy as input.

2- After implementing the first visit Monte Carlo control algorithm for eps-soft policies

a) The average undiscounted return across the 10 different runs with respect to the number of episodes



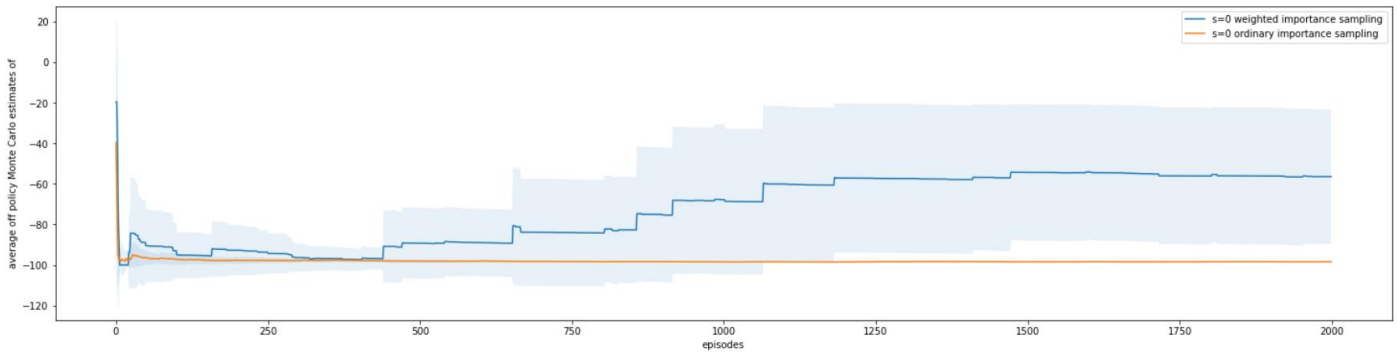
b) Sampling actions from the policy found with the first visit MC Control for epsilon soft policies gives the following visualization:

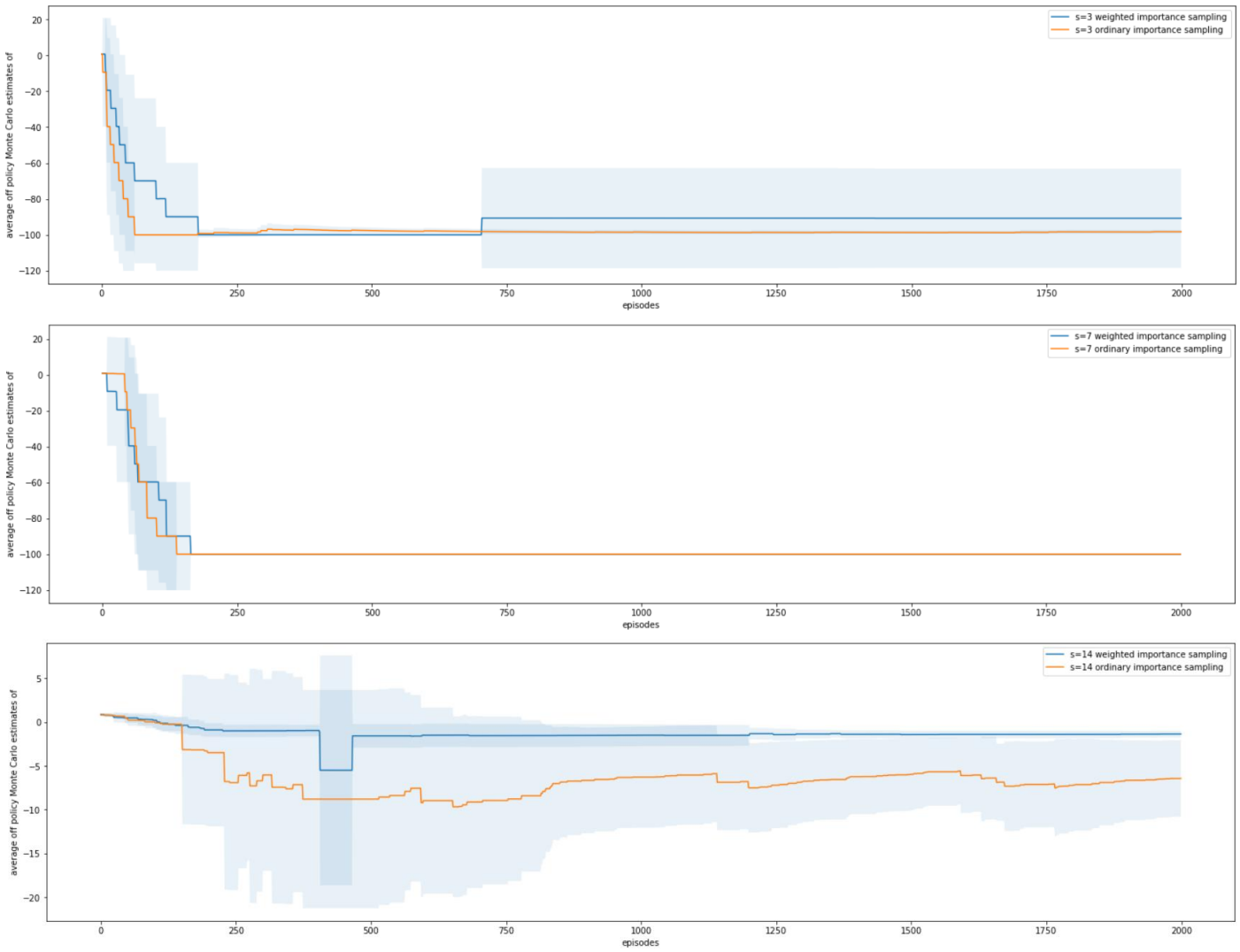


c)
The average undiscounted return for the last 100 episodes of the 10 runs is -88.65, which is small because of the policy learned with first visit MC algorithm is only an approximate optimal policy and because we have a slip rate.

3- After implementing ordinary importance sampling MC prediction and weighted importance sampling, and using a random behavior policy b(a|s) = 0.25 for every a and s, I have the following plots:

a)



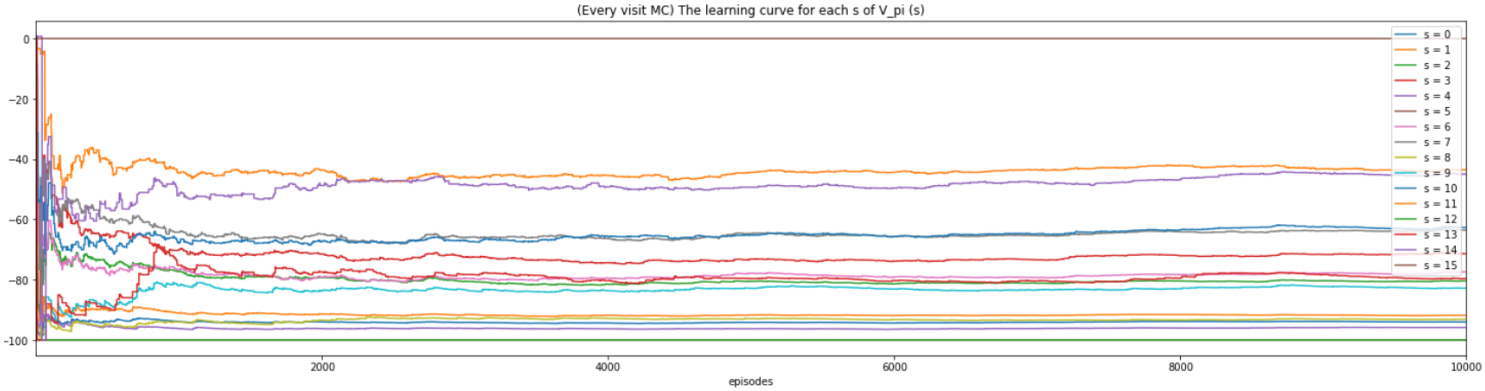


b) I expect the weighted importance sampling to produce a lower variance estimates because if a rare event occurs, ordinary importance sampling will vary a lot, but the weighted importance sampling, due to the fact that the new large weight appears also in the denominator will not vary that much.

2 Prediction: Unifying Monte Carlo methods and Temporal Difference Learning

1- After I implemented the Every visit MC prediction algorithm

a. The plot of the learning curves for each s of $V^{\pi}(s)$

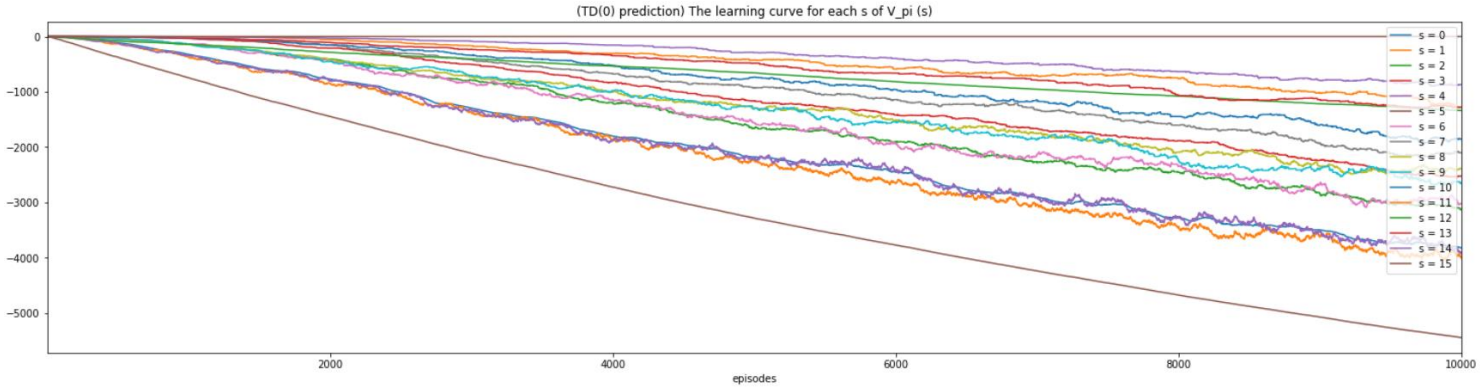


b. The final learned value function: which makes sense because the closer we get to the terminal state, the larger the values become, and that's because when we are close to the terminal state, the number of steps left before the end of the episode is small and so the number of -100 rewards we are going to get is small also.

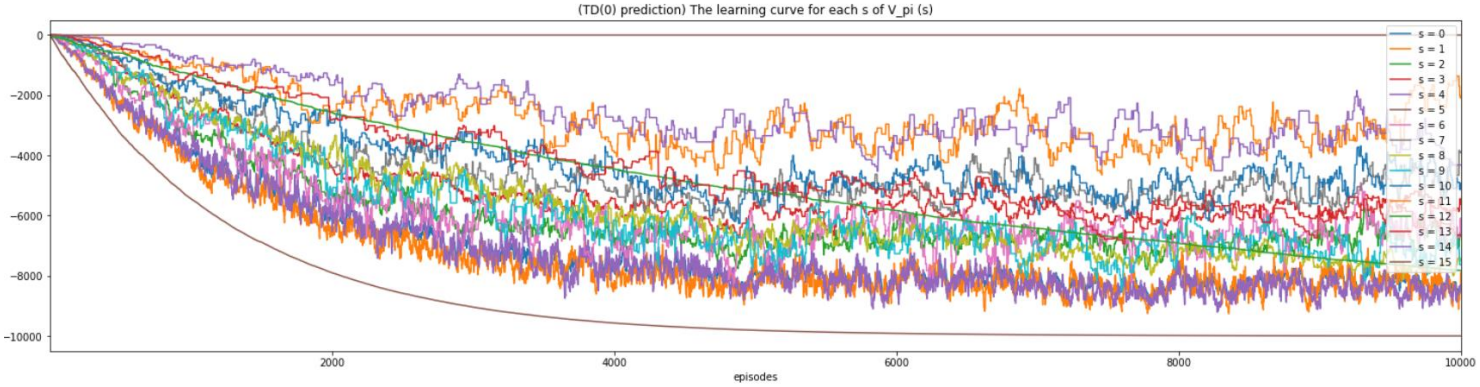
```
The final learned value function (Every visit MC):
array([ -93.99418235,  -91.80662109,  -80.41664888,  -71.3911572 ,
        -95.82809095, -100.        ,  -77.42978042,  -63.470762  ,
        -93.08358187,  -82.79899131,  -62.72978113,  -43.50660752,
       -100.        ,  -79.64026002,  -44.96923683,    0.        ])
```

2- I After I implemented the TD(0) prediction algorithm:

With $\alpha = 0.01$

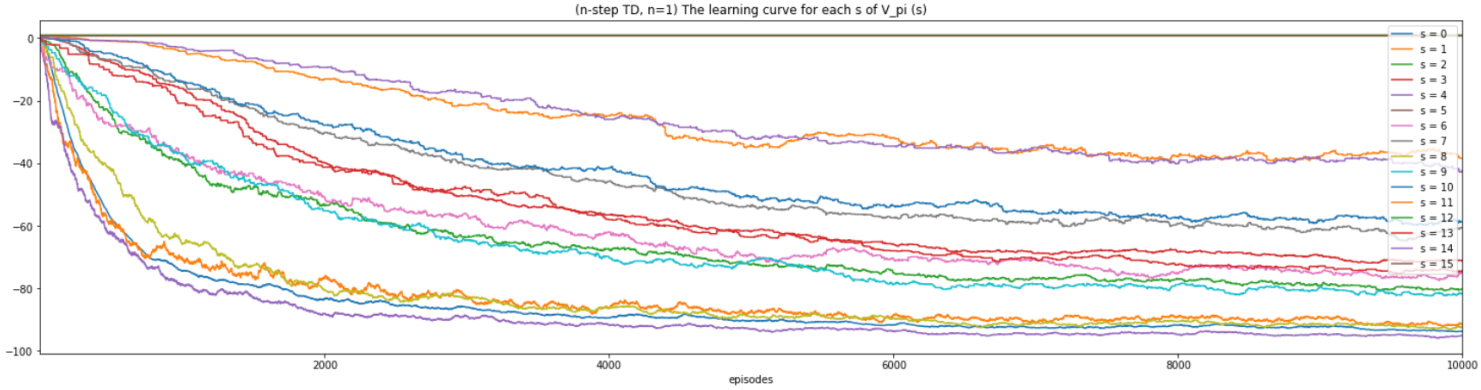


With $\alpha = 0.1$



3- After implementing the n-step TD algorithm:

a.

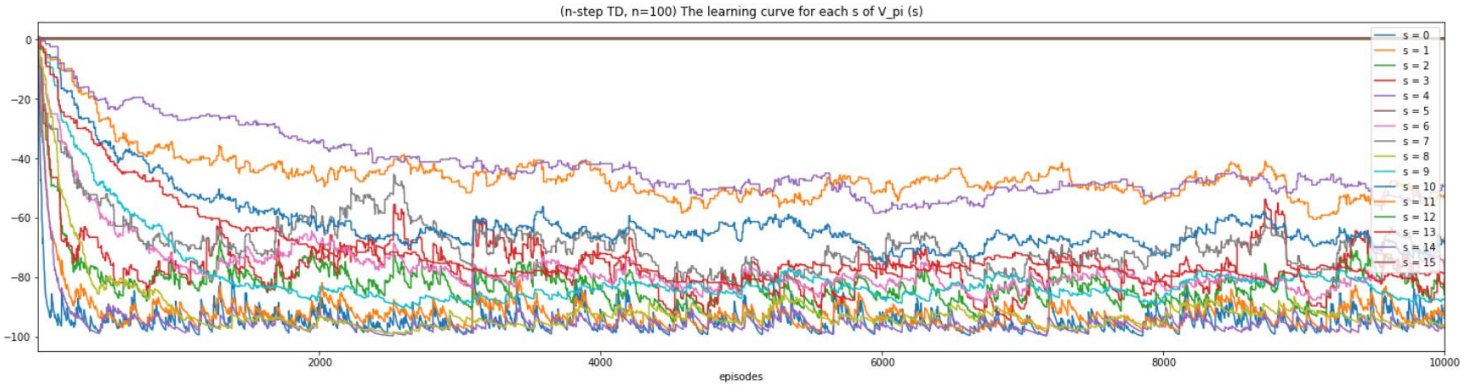


b. I expected the 1-step TD to look similar to TD(0), because the only difference between the two is that 1-step TD uses the accumulated sum of discounted rewards in its update

rule, while TD(0) uses the immediate reward. And we see that the plots of TD(0) and 1-step TD are pretty close especially if we take $\alpha = 0.1$ for TD(0).

4- Using the same implementation of n-step TD

a.



b. The plot doesn't look similar to Every visit MC, because we're using the same step-size in all the steps of an episode and in all episodes.

5-

a. I will change the constant α with $\alpha(S)$: the number of times the value of state S is updated. Thus $\alpha(S)$ is incremented every time $V(S_t)$ is updated.

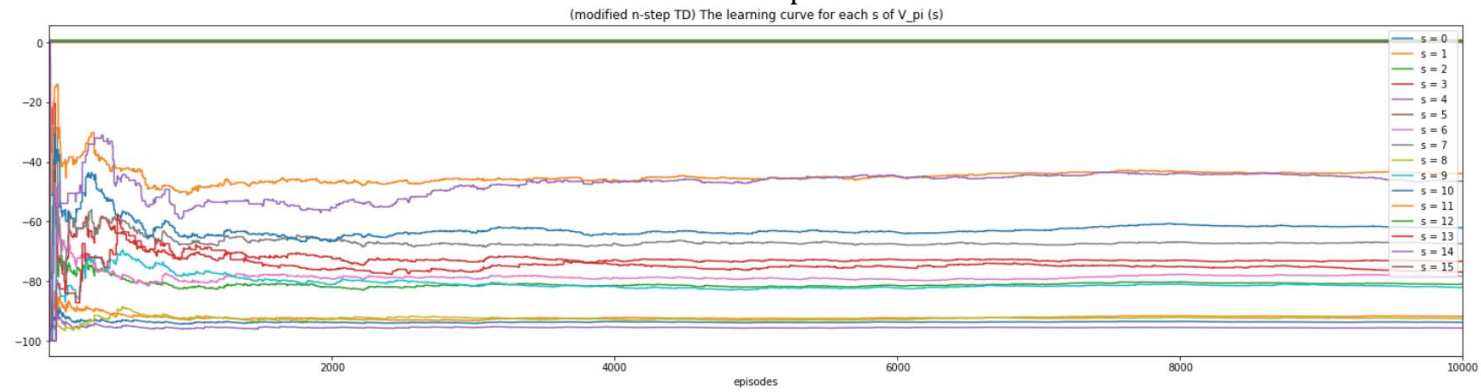
In pseudo-code it looks like this

```
ALGO modified_n_step_TD
  ...
  alphas = np.ones(number_of_states)

  Loop for each episode
    ...
    Loop for t = 0, 1, 3, ...
      ...
      if tau >= 0
        V(S_tau) = V(S_tau) + (1/alphas(S_tau))*[G - V(S_tau)]
        alphas(S_tau) = alphas(S_tau) + 1
```

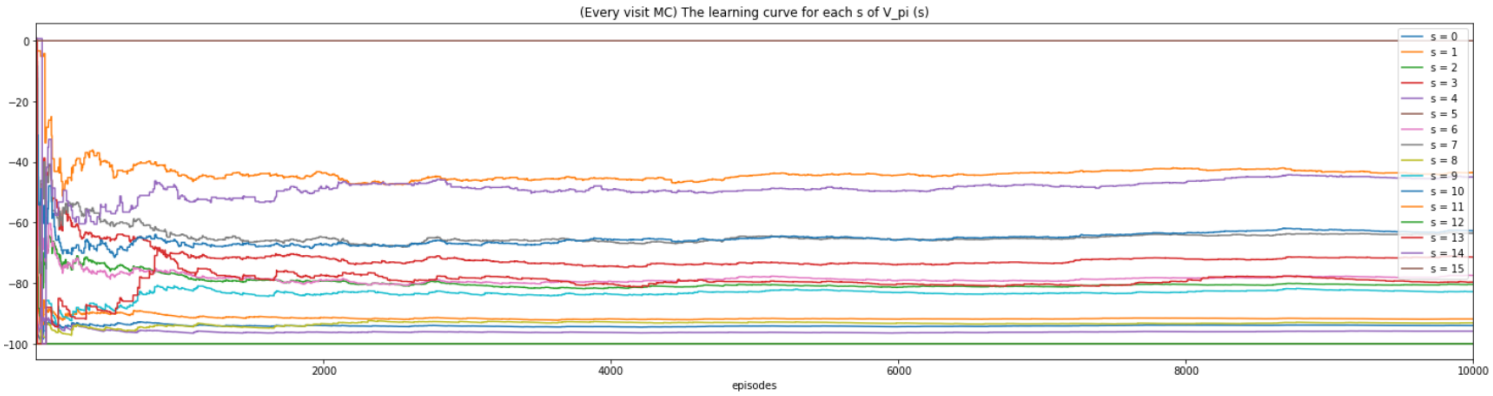
b. The plot of the modified n-step TD

Modified n-step TD



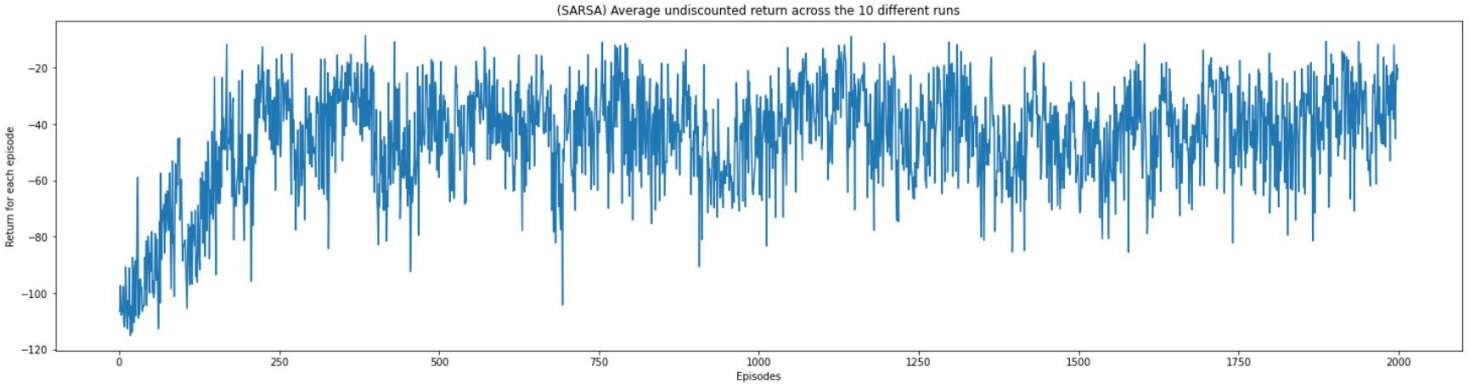
c. The new plot looks very similar to one of Every visit MC prediction, because this time, the step-size is (number of updated times)-dependent

Every Visit MC (for comparison with the modified n-step TD)



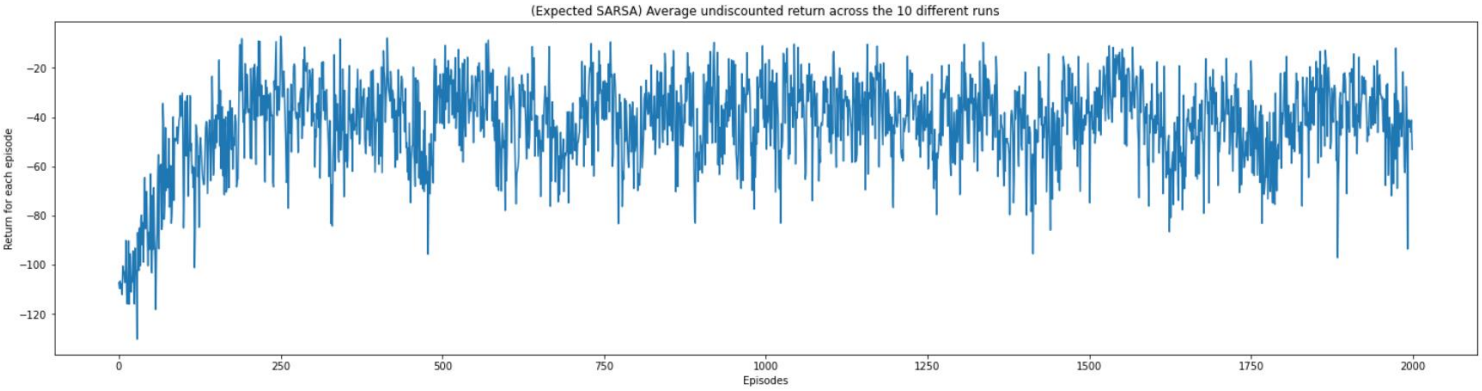
3 Temporal Difference Control Methods

1- After I implemented the SARSA control algorithm



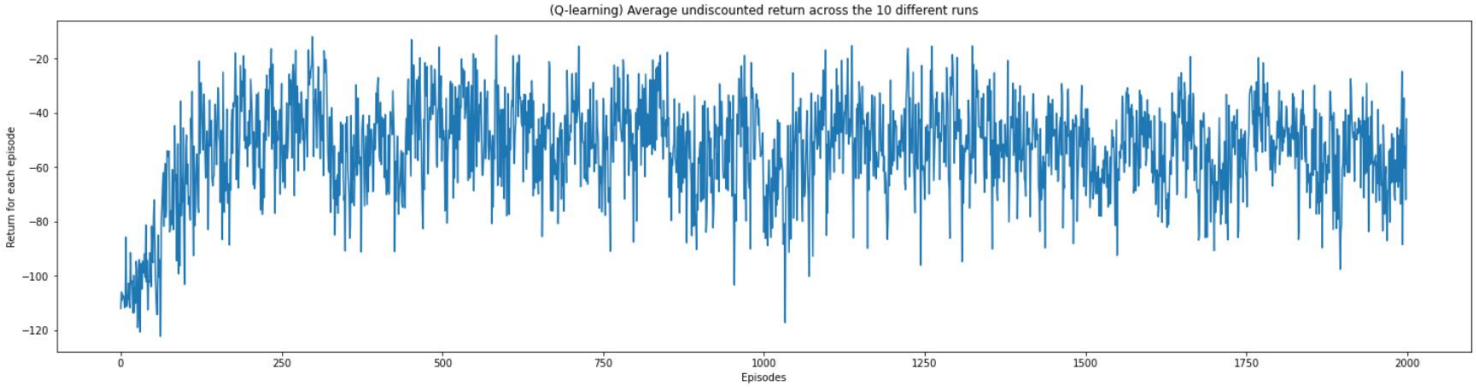
and the rendering of one of the policies learned (I used the 10th one) proves its optimality

2- After I implemented the Expected SARSA control algorithm



and the rendering of one of the policies learned (I used the 10th one) proves its optimality

3- After I implemented the Q-learning control algorithm



and the rendering of one of the policies learned (I used the 10th one) proves its optimality

4- Comparing the performances of MC methods and that of TD methods, I see that TD converges faster than MC, because MC needs to wait until the end of an episode to update while TD doesn't have to do this. And that MC has a higher variance than TD, because each update of MC is done using a real sample of Q(s, a) which reduces the bias but increases the variance (bias-variance tradeoff).