Exercise 0
Gregory Attra
02.20.2022
CS 7180 - Prof. Amato
Code Setup:

1. Unzipped the 'ex4.zip' file
2. 'cd' into the 'code' directory
3. run 'source ./init' to setup the pyenv

Questions:

1. (a)
   - Prior to the infinite loop, create a dictionary for storing the number of times state s has been reached:
     $N(s) = 0 \forall s \in S$
   - After the check for "first-visit", increment the number of times state s has been reached and use the updated $N(s)$ value to average the return at that state:
     $N(s) = N(s) + 1$
     $V(s) = V(s) + \frac{1}{N(s)}[G - V(s)]$

   (b)
   - Prior to the infinite loop, create a dictionary for storing the number of times action a has been taken at state s:
     $N(s, a) = 0 \forall s \in S \forall a \in A$
   - After the check for "first-visit", increment the number of times action a has been taken in state s and use the updated $N(s, a)$ value to average the return for that state-action pair:
     $N(s, a) = N(s, a) + 1$
     $Q(s, a) = Q(s, a) + \frac{1}{N(s,a)}[G - Q(s, a)]$

2. (a) I would expect the average return for a state to be the same. When using every-visit MC, we include returns which are computing later in the episode, in other words: smaller returns. This brings our average lower that had we used first-visit, as we would have a higher likelihood of encountering a state earlier in the episode, and thus having a higher return. However, in blackjack, episodes are incredibly brief with often fewer than five steps per episode. The probability of encountering the same state twice in a given episode is low, and even still, the brief nature of each episode means the expected return when a given state is first entered will be roughly equal, but slightly greater, than the expected return when that state is reached later in the episode.

   (b) i. First Visit:
   T - 1: $G = \gamma G + r = 1$
   $Q(T, a) = Q(T, a) + \frac{1}{N(T,a)=1}[G - Q(T, a)] = 1$
   T - 2 $\rightarrow$ 9 :G $= \gamma G + r$

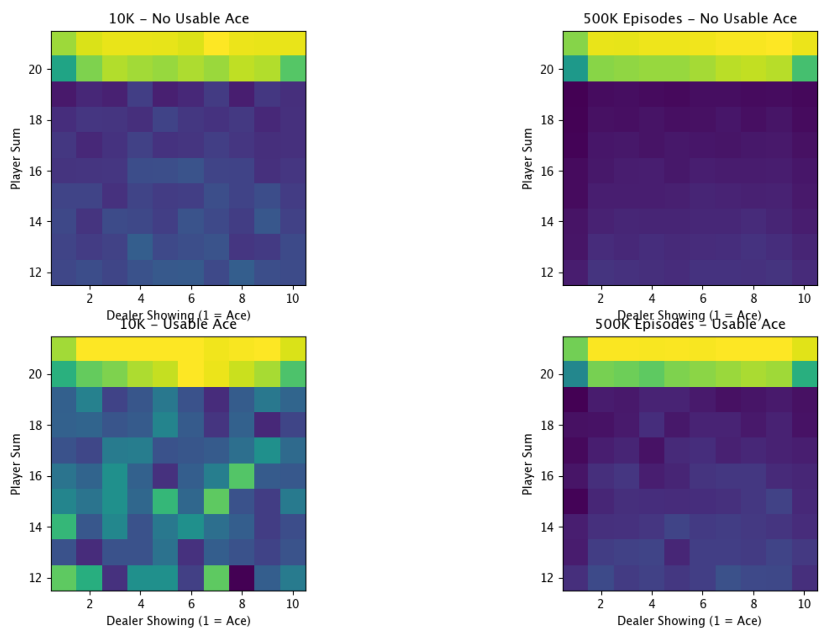
T - 10: $G = G + r = 9 + 1 = 10$ $Q(NT,a) = Q(NT,a) + \frac{1}{N(T,a)=1}[G - Q(NT,a)] = 10$
The estimate for $Q(NT,a) = \frac{1}{N(s,a)=1}[10 - Q(NT,a)] = 10$

ii. Every Visit:
$Q(T,a) = 1$
$Q(NT,a) = \frac{G_{T-2}+G_{T-3}...+G_{T-10}}{N(NT,a)} = \frac{2+3+4...+10}{9} = 6$

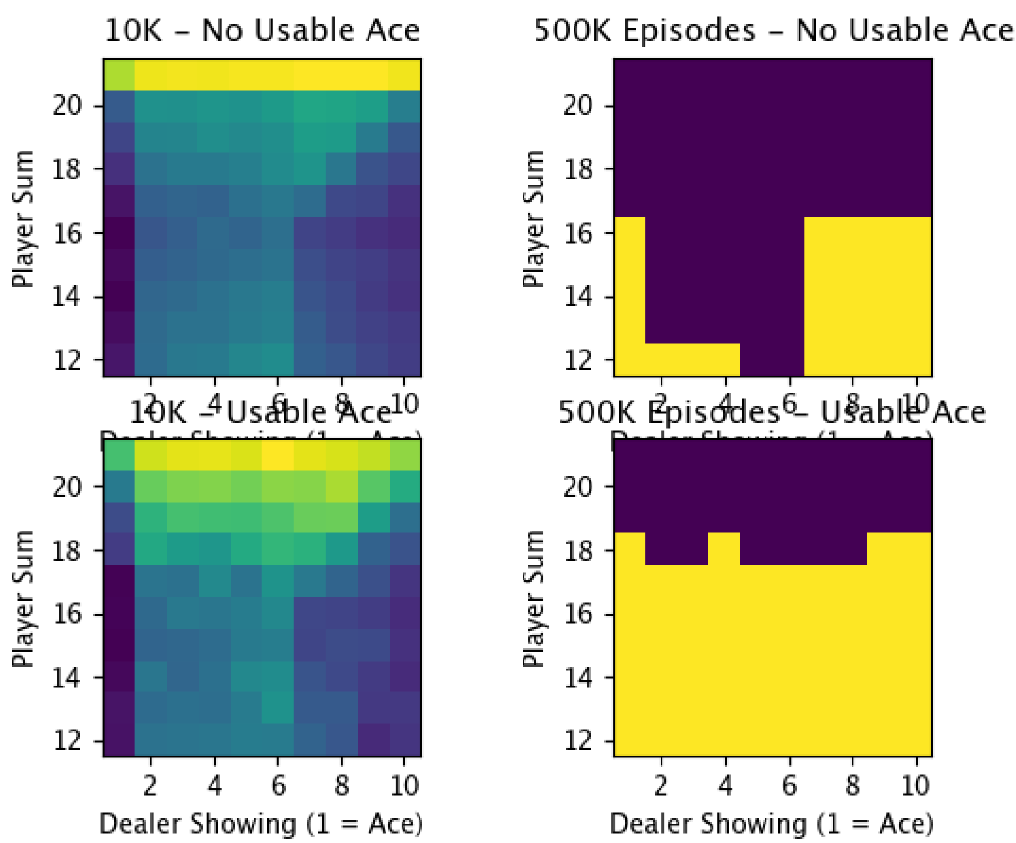
3. Blackjack:
The algorithms for MC control live in ‘blackjack.py‘

   (a) To run Blackjack without exploring starts run: ‘python src/monte_carlo/run_blackjack.py‘

   (b) To run Blackjack with exploring starts run: ‘python src/monte_carlo/run_blackjack_es.py‘



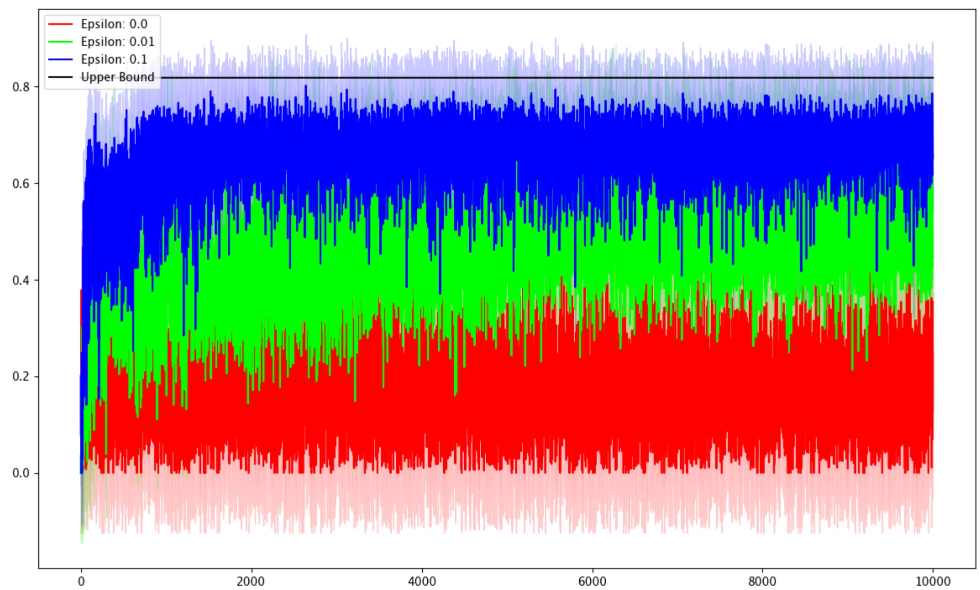
Blackjack First-Visit MC



Blackjack First-Visit MC w/ Exploring Starts

4. Four Rooms:
   The algorithms for MC Control live in ‘algorithms.py‘. The environment for the Four Rooms problem lives in ‘env.py‘.

   (a) To run the four rooms problem: ‘python src/monte_carlo/run_four_rooms.py‘



Four Rooms MC with Epsilon Soft Policy

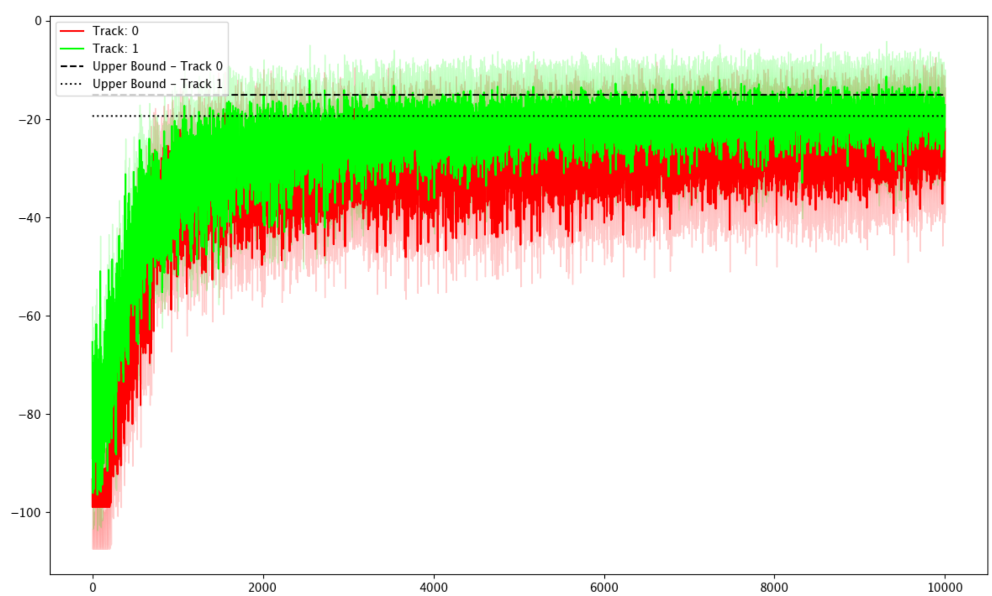
5. (a)

   (b) Because if $A_t \neq \pi(S_t)$, we exit and do not update the weight. It is therefore redundant to check $\pi(A_t|S_t)$ as we know it is 1 (since π is a greedy policy), else we would have exited.

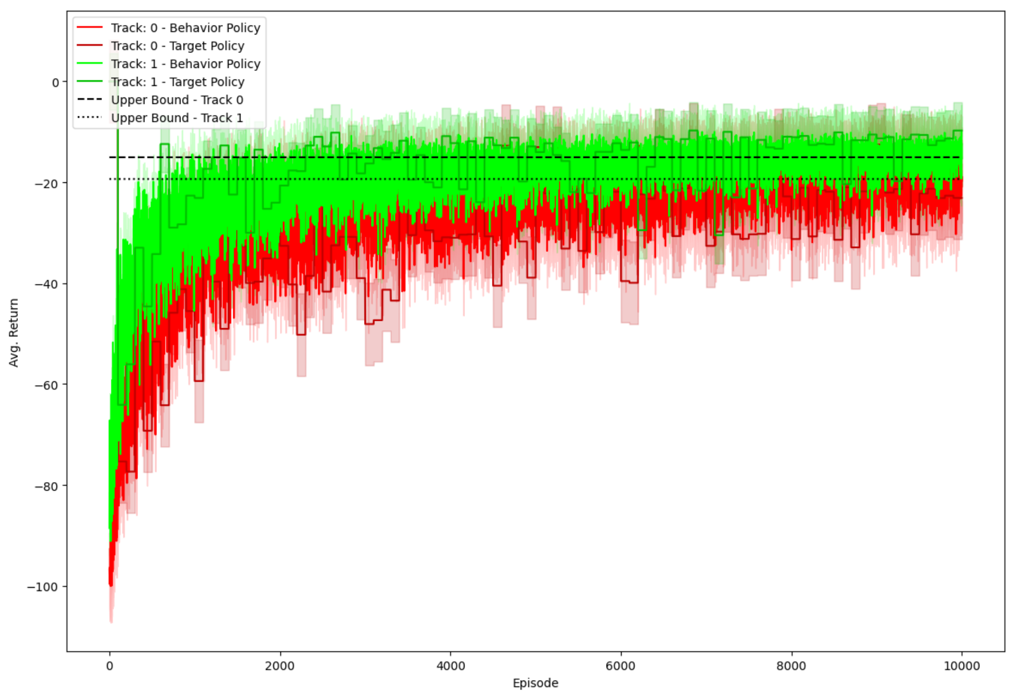
6. Racetracks:
   The algorithms for on-policy and off-policy MC Control live in 'algorithms.py'. The environment lives in 'racetracks.py'

   (a) To run the racetracks problem using On-Policy MC Control: 'python src/monte_carlo/run_on_policy_racetrack.py'

   (b) To run the racetracks problem using Off-Policy MC Control: 'python src/monte_carlo/run_off_policy_racetrack.py'



On-Policy MC Control for Racetrack Problem



Off-Policy MC Control for Racetrack Problem