Reinforcement Learning Assignment-1

Logic For Environment

- Using gym as abstract class for my environment
- action_space = discrete values including [0,0],[0,1],[1,0],[1,1],[0,-1],[-1,0],[-1,1],[-1,-1]
- Observation_space = all pair of discrete value of positions in state + all pair of velocities (total = 6048)
- reset() :-
  - choose a random start state
- Collision Detector :- I am assuming that the car is moving in a straight line connecting its initial position and position after translating with v velocity. If the object hits the wall or is outside the track and not finished, we reset. If it finish, we will stop the episode.

On Policy Algorithm Used

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$
Initialize:
  $\pi \leftarrow$ an arbitrary ε-soft policy
  $Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
  $Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):
  Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow \gamma G + R_{t+1}$
    Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:
      Append G to $Returns(S_t, A_t)$
      $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
      $A^* \leftarrow \text{argmax}_a\, Q(S_t, a)$ (with ties broken arbitrarily)
      For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

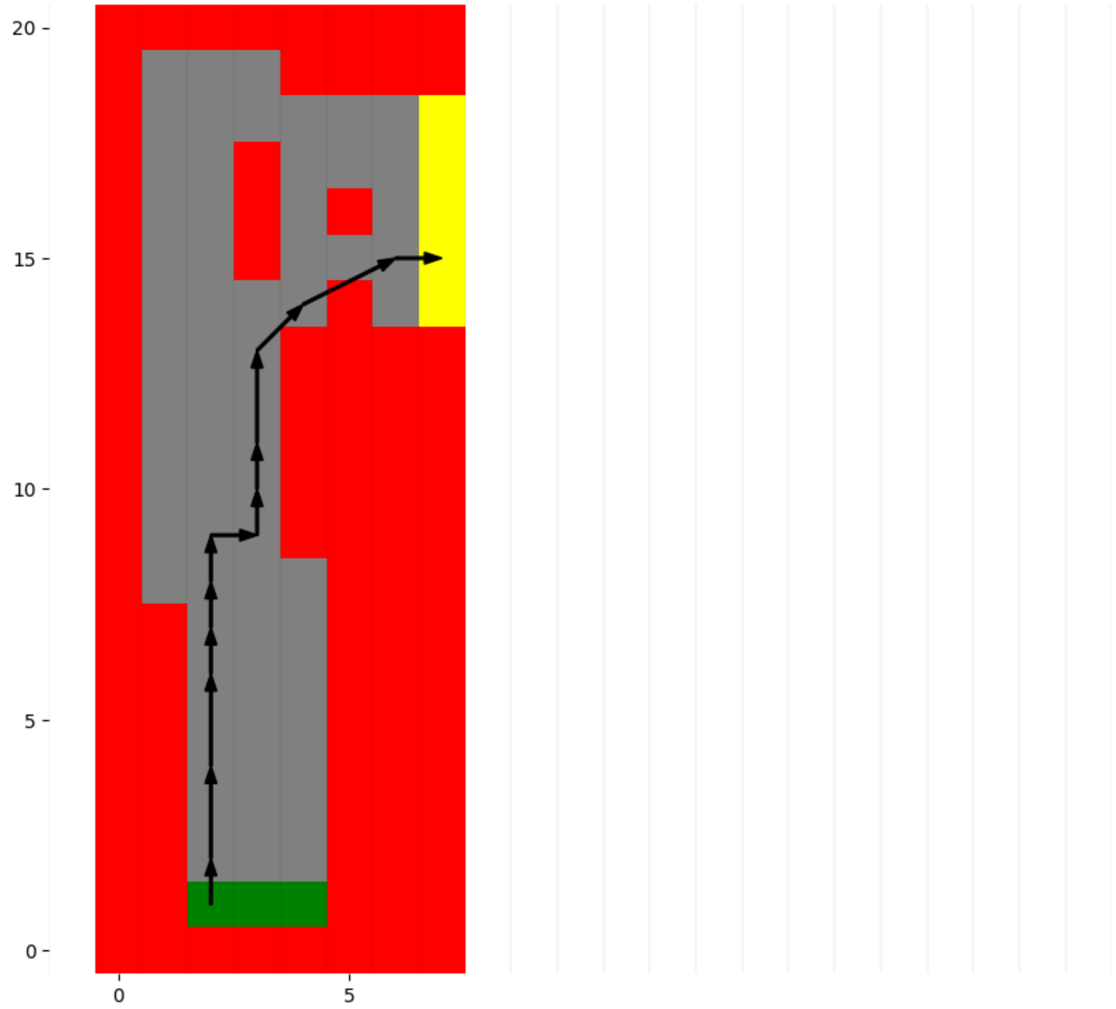
- I initialized the q as np.ones() of dimension (number of states, number of actions)
- I initialized the pi as np.ones()/number_of_actions of dimenstion (number of states, number of actions)
- I am choosing epsilon as 0.1 for my algorithm and gamma as 0.1
- number_of_episodes = 1,00,000 (time taken:- 2.5 hrs average)
- for each episode
  - returns,state_indexes,action_indexes = run episode
  - returns is the sum G without discount
  - for each timestep
    - q[state,action] = q[state,action] + gamma*(return at that timestep - q[state,action])

- greedyActions = np.agrmax of q for each state
- set value of pi at all state,greedy action as 1 - epsilon + epsilon/number_of_actions
- set all other as epsilon/number_of_actions

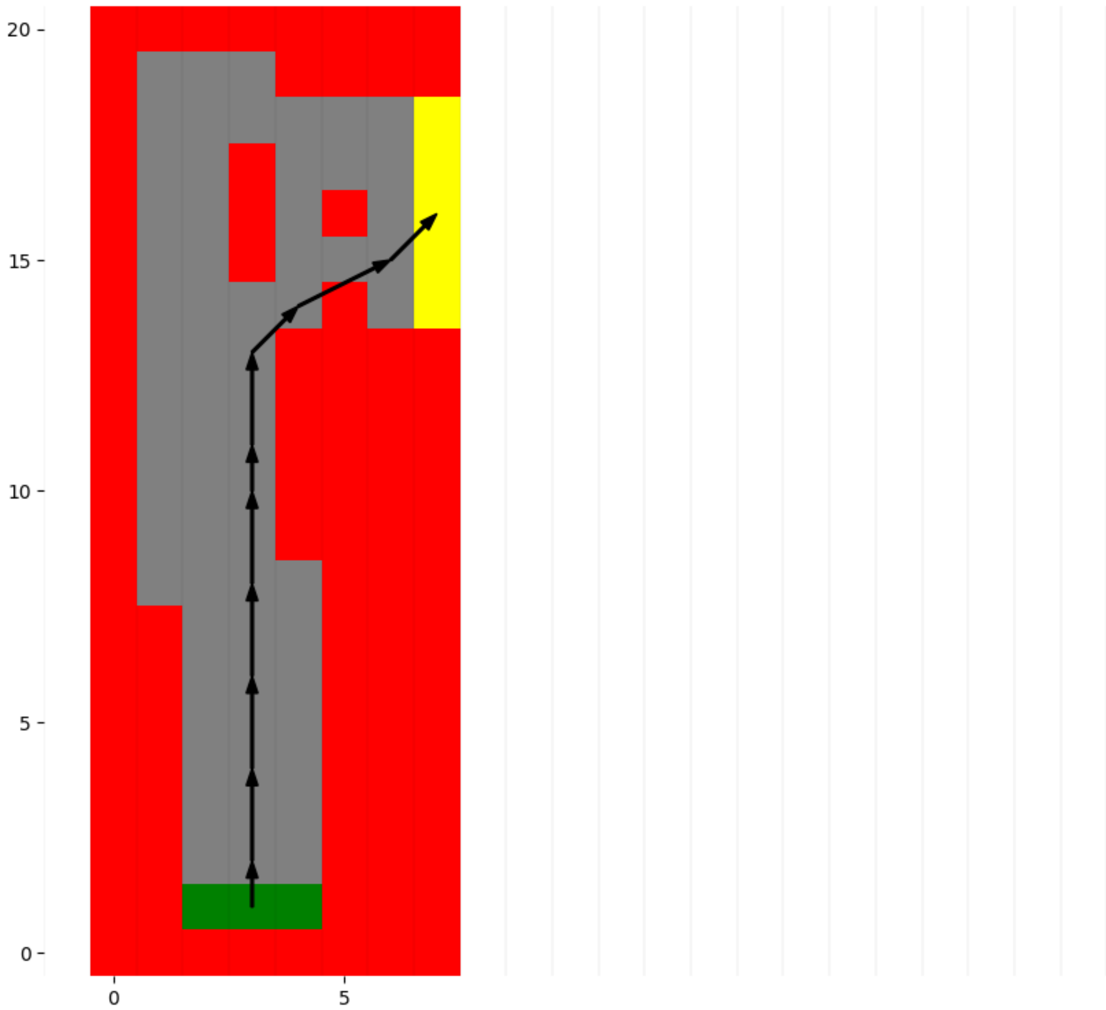
- Then set policy value of all action for each state as 1 if that action in argmax of state

On policy plots:-

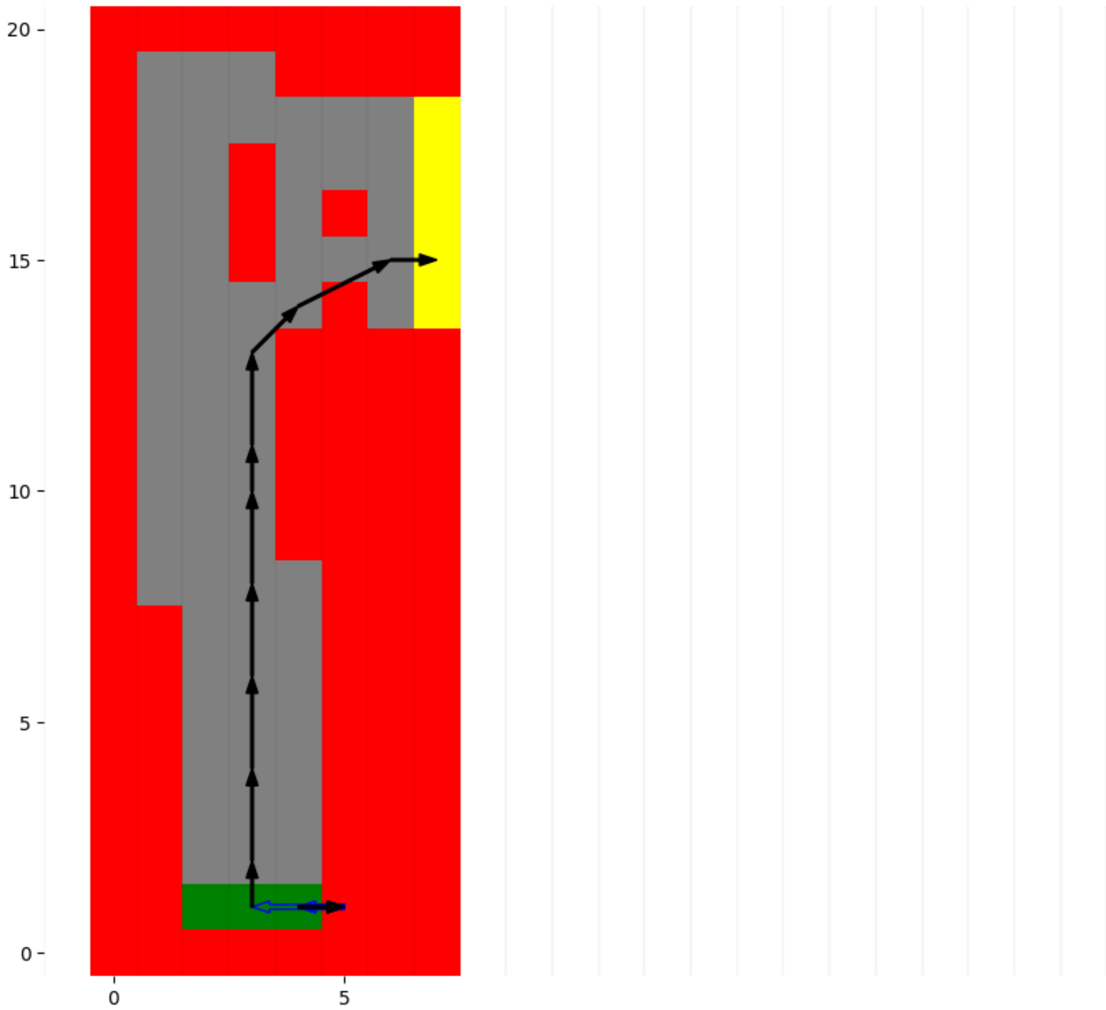
- start state 1



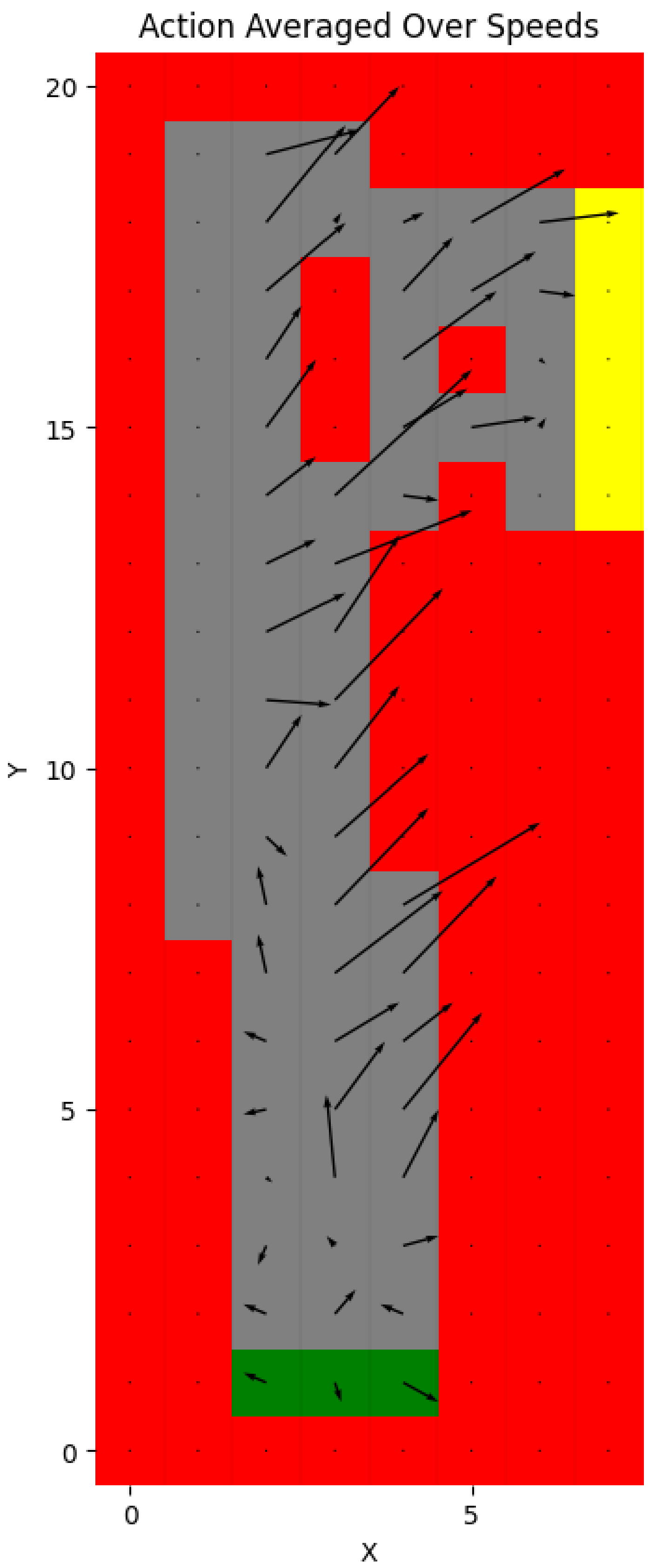
- start state 2



- start state 3



- Action using policy for in each state



Off Policy Algorithm Used

Off-policy MC control, for estimating $\pi \approx \pi_*$

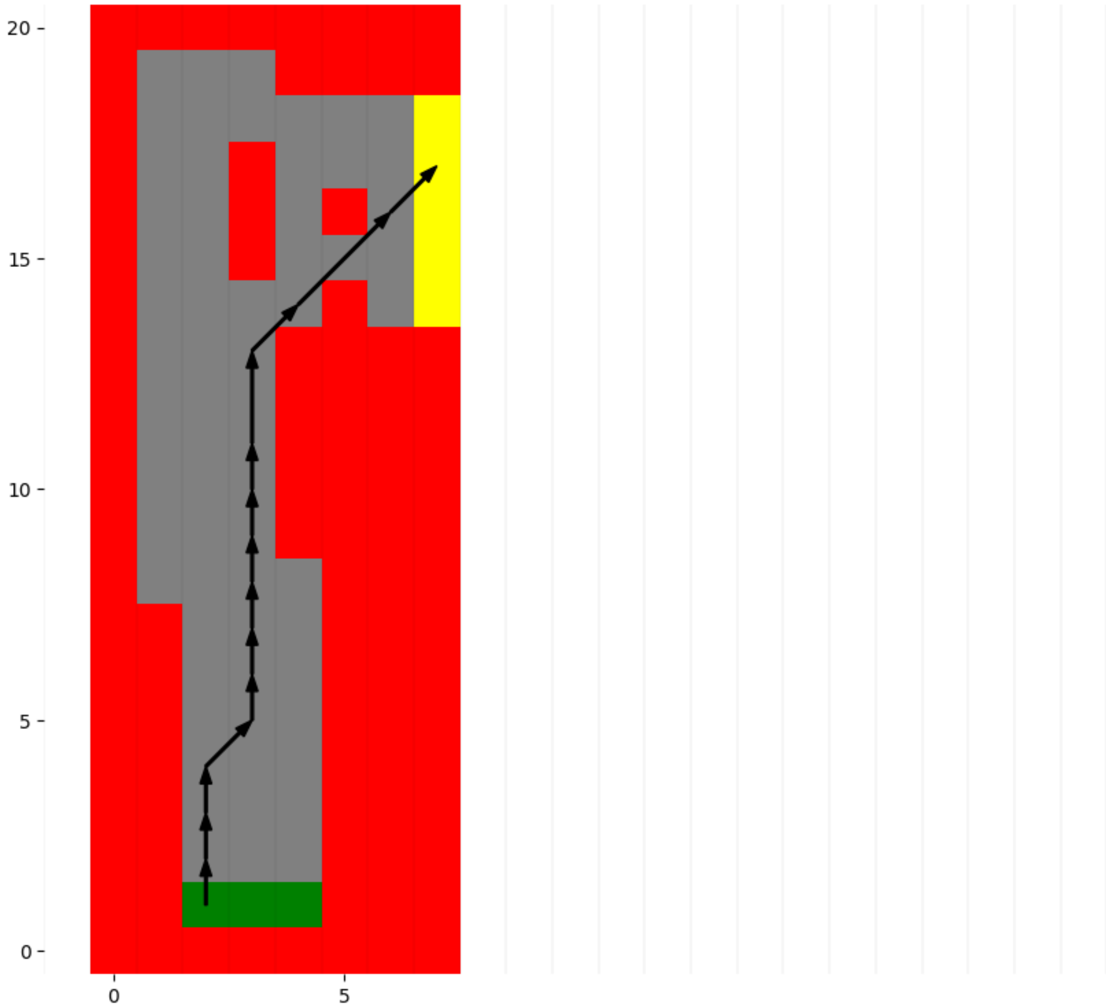
Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
- $Q(s,a) \leftarrow$ arbitrary
- $C(s,a) \leftarrow 0$
- $\pi(s) \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken consistently)

Repeat forever:
- $b \leftarrow$ any soft policy
- Generate an episode using b:
  - $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T, S_T$
- $G \leftarrow 0$
- $W \leftarrow 1$
- For $t = T-1, T-2, \ldots$ down to 0:
  - $G \leftarrow \gamma G + R_{t+1}$
  - $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
  - $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
  - $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken consistently)
  - If $A_t \neq \pi(S_t)$ then exit For loop
  - $W \leftarrow W \frac{1}{b(A_t|S_t)}$

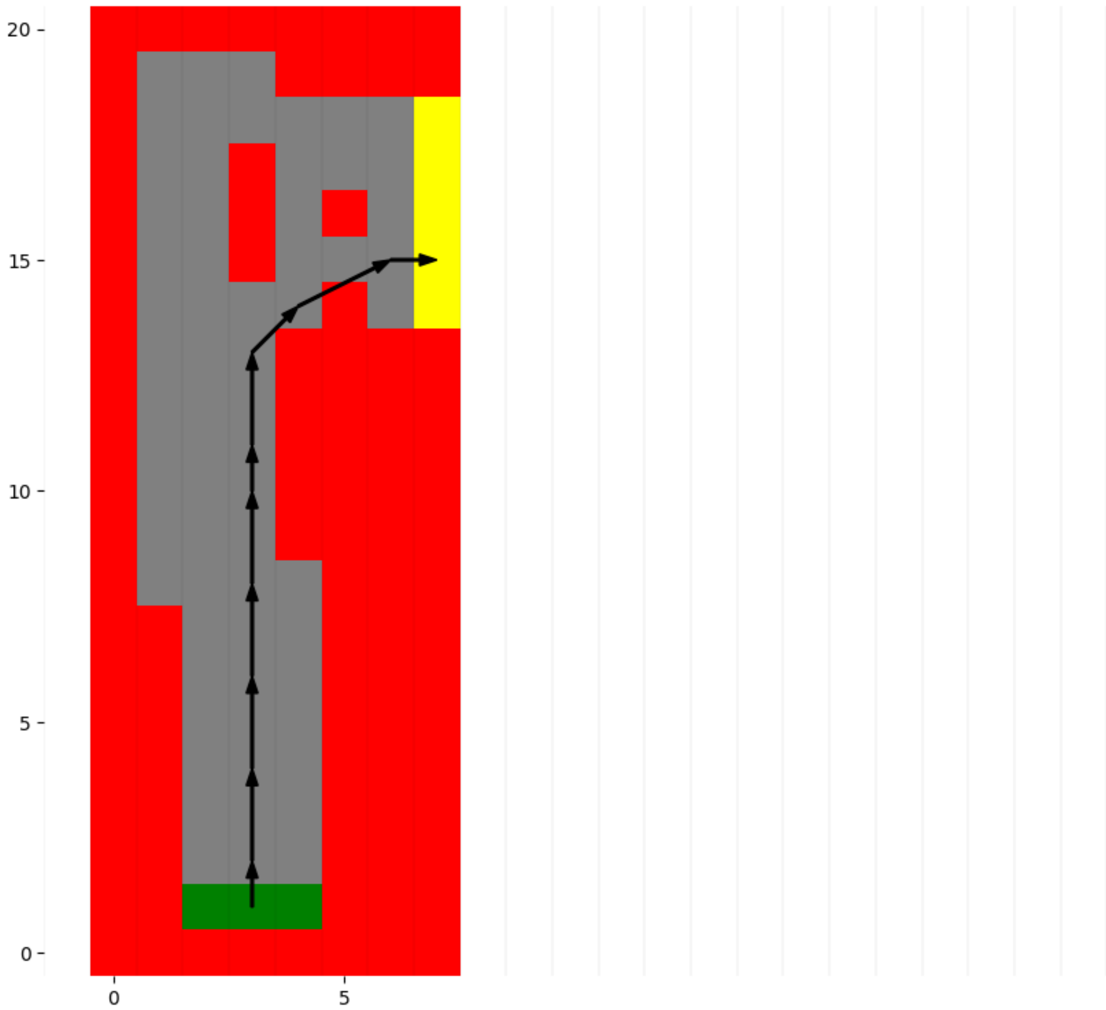
- I initialized the q as np.ones() of dimension (number of states, number of actions)
- I initialized the pi as np.ones()/number_of_actions of dimension (number of states, number of actions)
- C = zeros for dimension (number of states,number of action)
- soft_policy = result of on policy
- num_iterations = 1,00,000
- gamma = 0.1
- for each iteration
  - w = 1
  - states,returns,actions = run episode
  - for each timestep
    - c(state,action) = 1 + w
    - q(state,action) += gamma* (g - q[state,action])/(W/c[state,action])
    - set policy[state, greedyAction] as 1 rest as 0. Here greedy action is np.argmax(q[state])
    - if action != greedy action, move to next episode
    - else W = W/(soft_policy[state,action]

Off Policy Plots

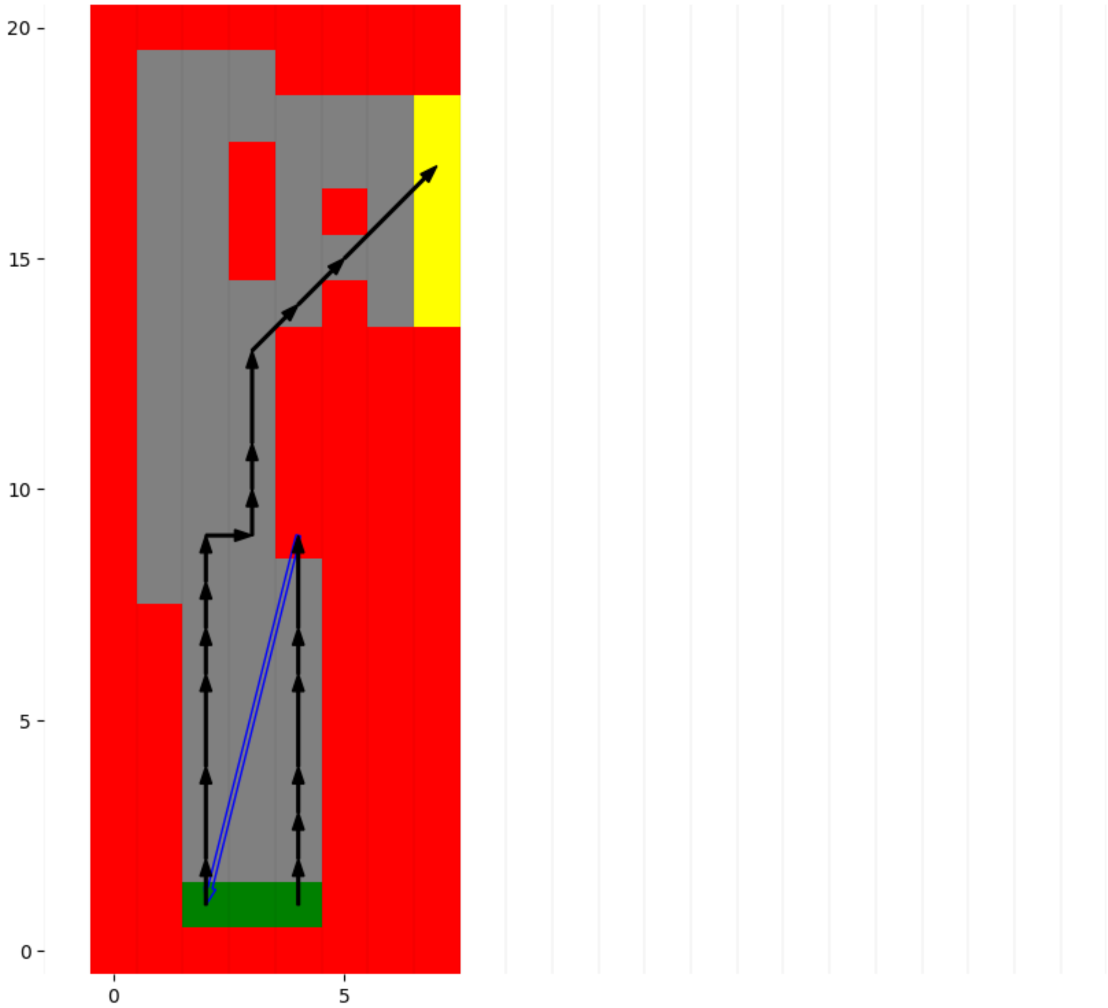
- starting state 1



- starting state 2



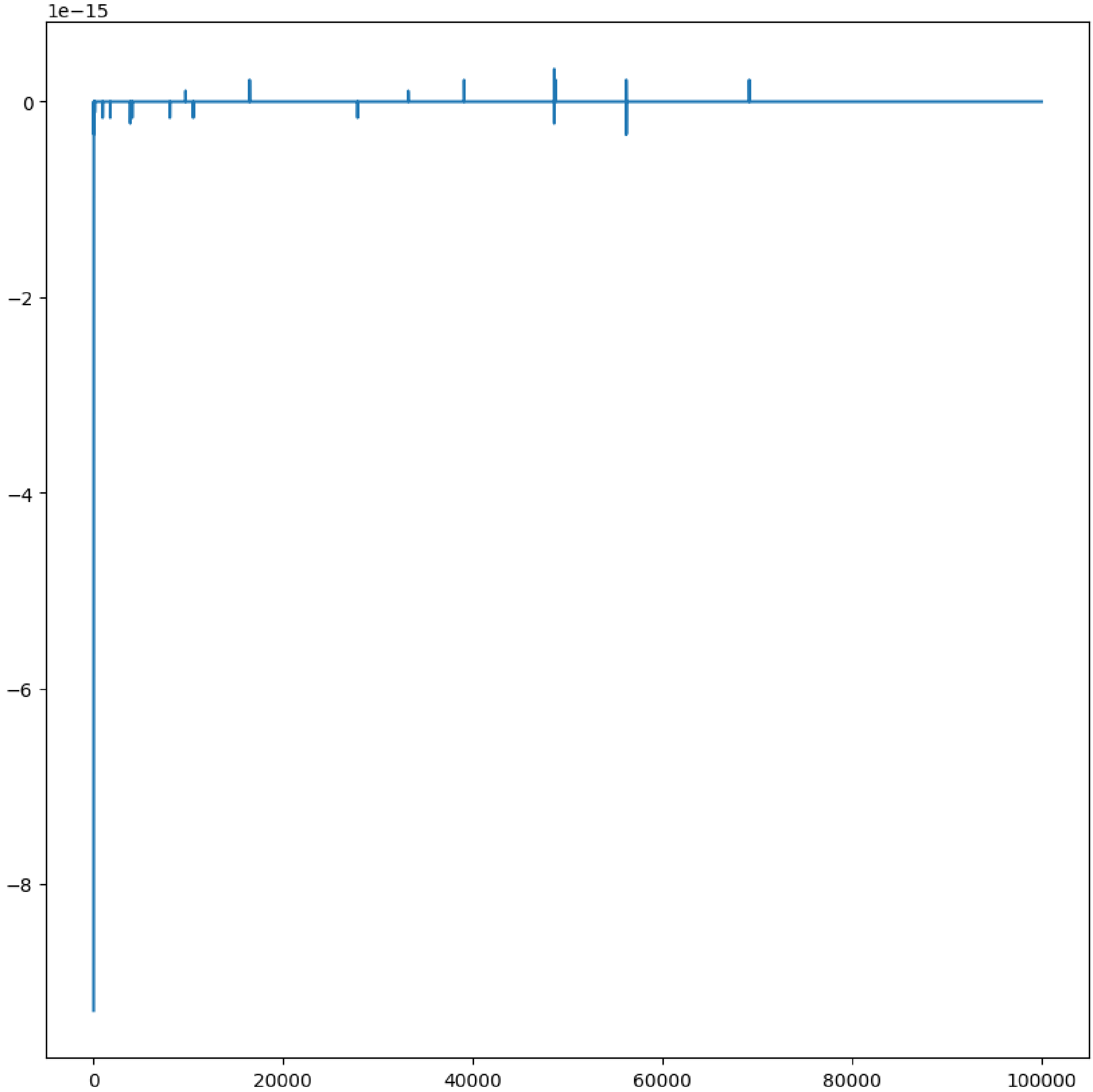
- starting state 3



Convergence Test

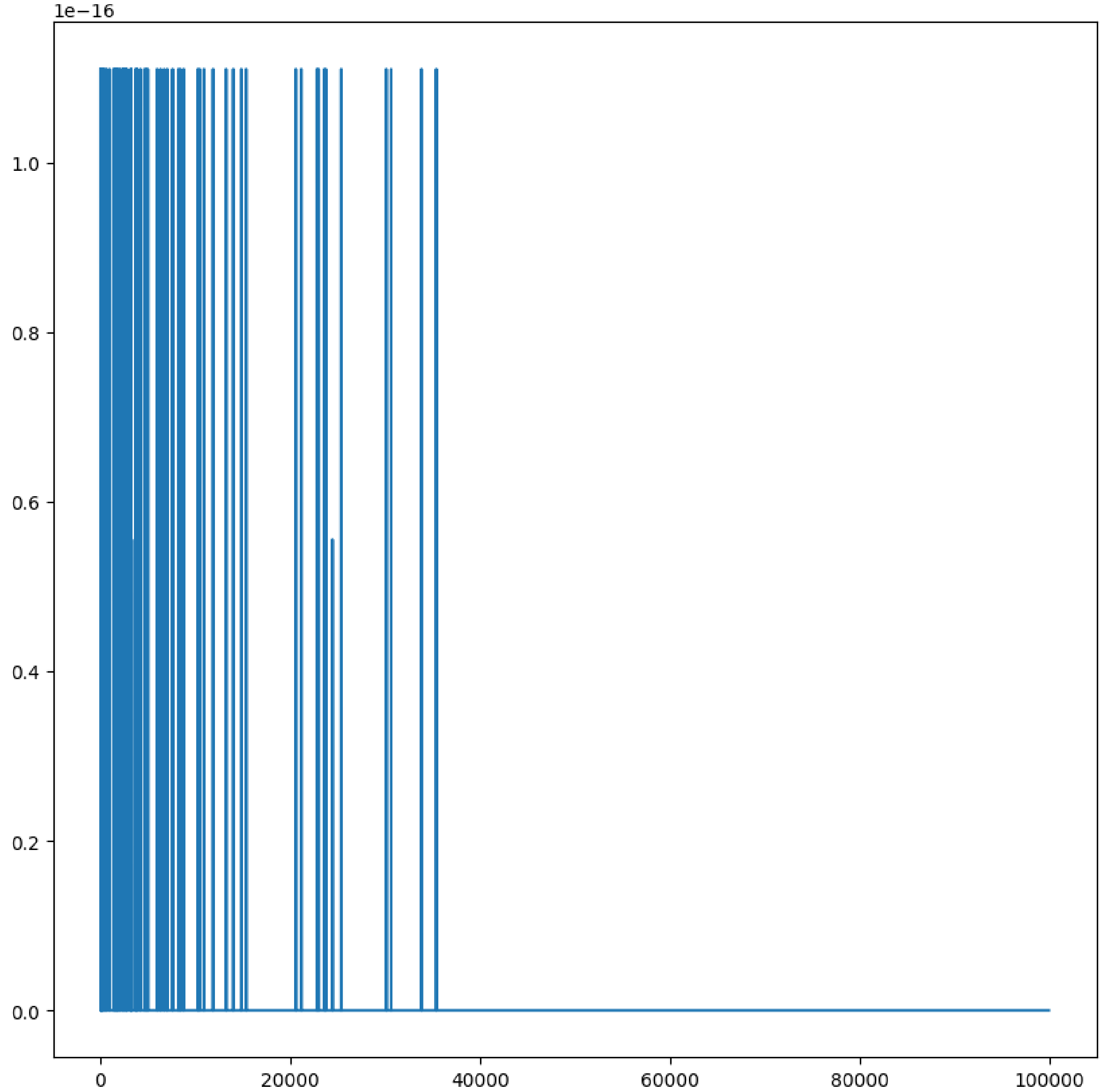
I am checking the difference in policy before the iteration and after the iteration and then plotting the graph between episode number and difference.

For On Policy Method:-



As we can see difference is getting very close to 0 when episode count increase, so we can say on policy is converging

For Off-policy



As number of iterations increase by 40000, difference become close to 0. So we can say, off policy is also converging.

References

- https://github.com/BY571/Medium_Code_Examples/blob/master/Gridworld/Monte%20Carlo%20Methods%20Examples.ipynb
- https://medium.com/analytics-vidhya/monte-carlo-methods-in-reinforcement-learning-part-1-on-policy-methods-1f004d59686a
- https://github.com/vojtamolda/reinforcement-learning-an-introduction
- https://towardsdatascience.com/solving-racetrack-in-reinforcement-learning-using-monte-carlo-control-bdee2aa4f04e