Question1:

Please find the code attached below

Question 2:
Please find the code attached below

Question 3:
Please find the code attached below

Question 4:
We initialise random states: dealer’s one card, sum of player, usable ace or not
Then we take an action based on the policy that: stick if sum is 20/21 else hit

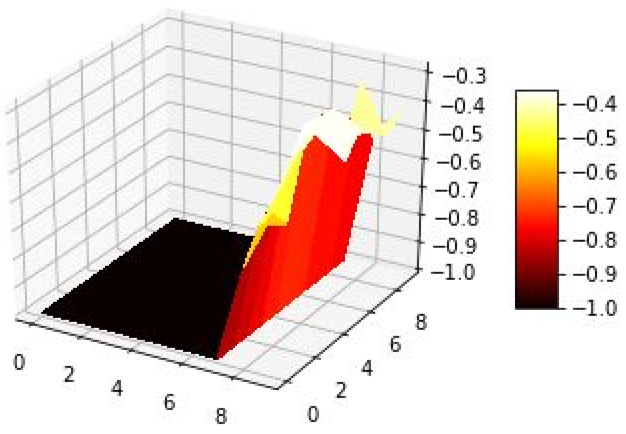
In the code, I assume that if we have a usable ace and if the sum exceed 21, then that usable ace is changed to “no usable ace” and the sum = sum - 10 (10 = 11 -1)

If player goes bust, straight away -1 given, dealer does not pays then.
The dealer only plays if player is not bust, if the dealer goes bust, then the player wins.

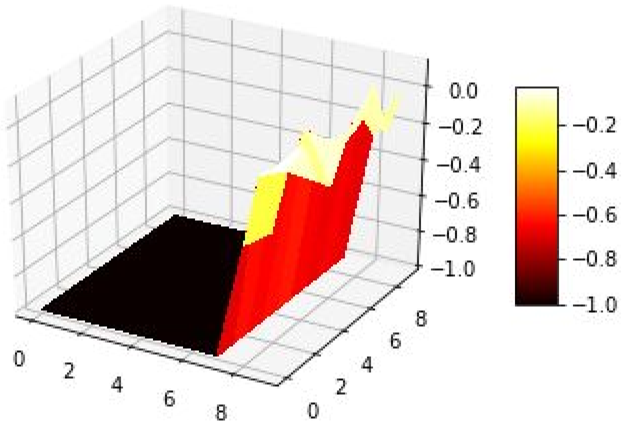
The dealer’s policy is as mentioned in the book.

Rest is also commented in the code

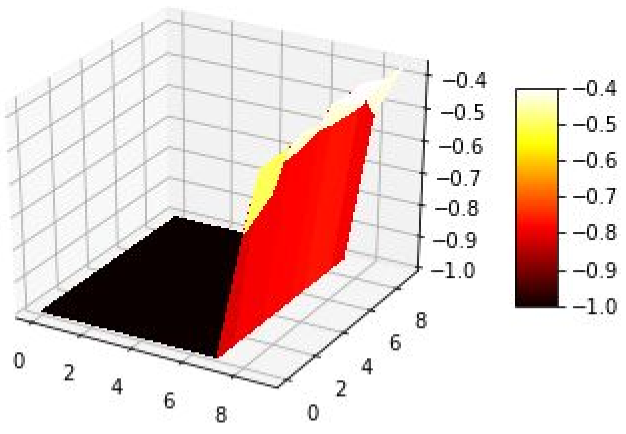
10,000 trials
Without Ace:



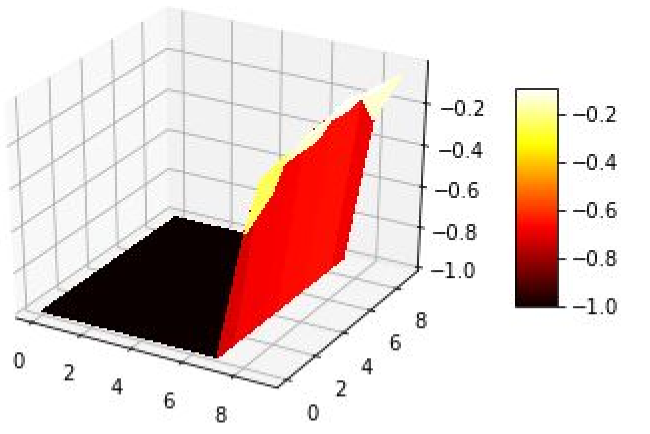
With Ace:



500,000 Trials
Without Ace:

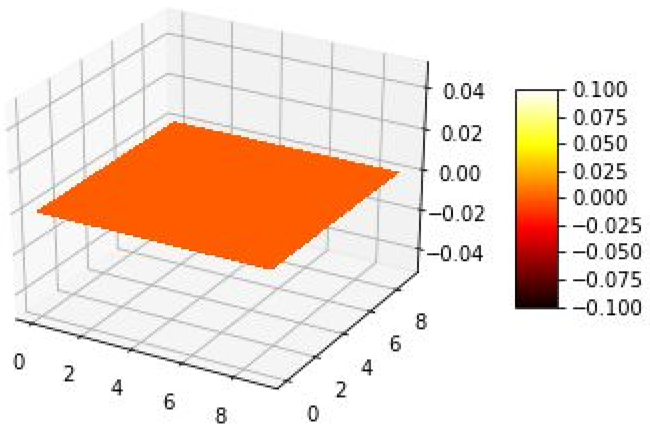


With Ace:

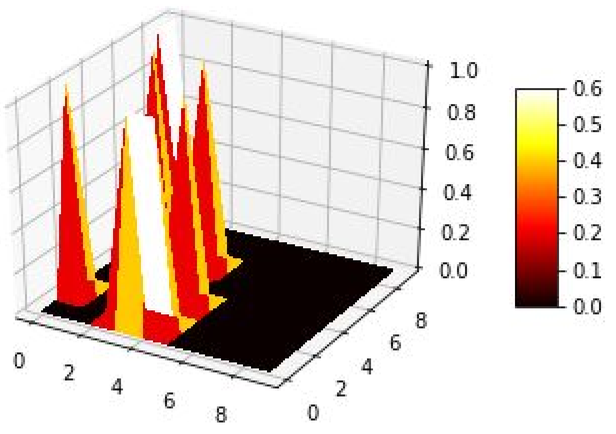


Policies:

Without Ace:

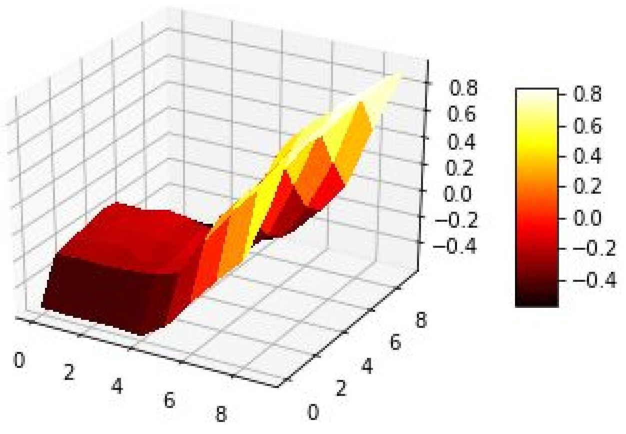


With Ace:

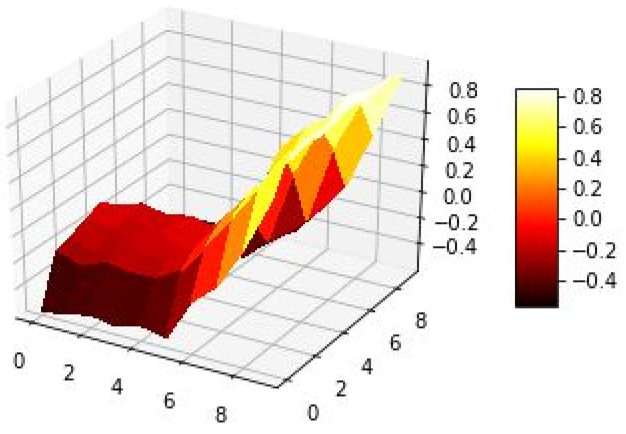


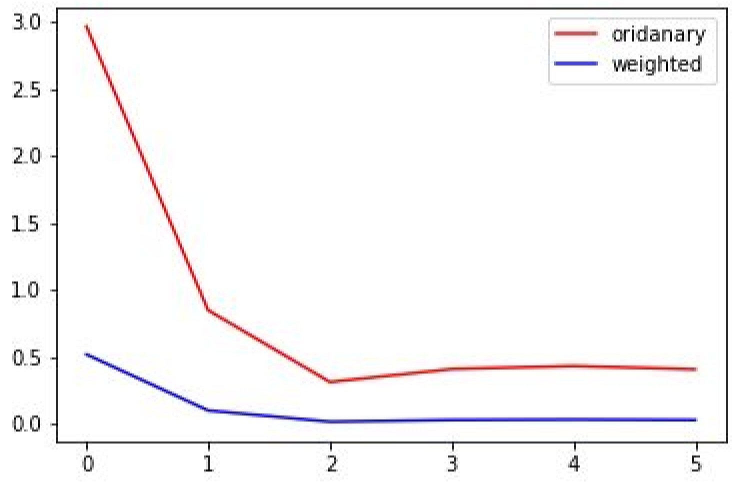
V STAR:

Without ace:



With Ace:





Question 5:

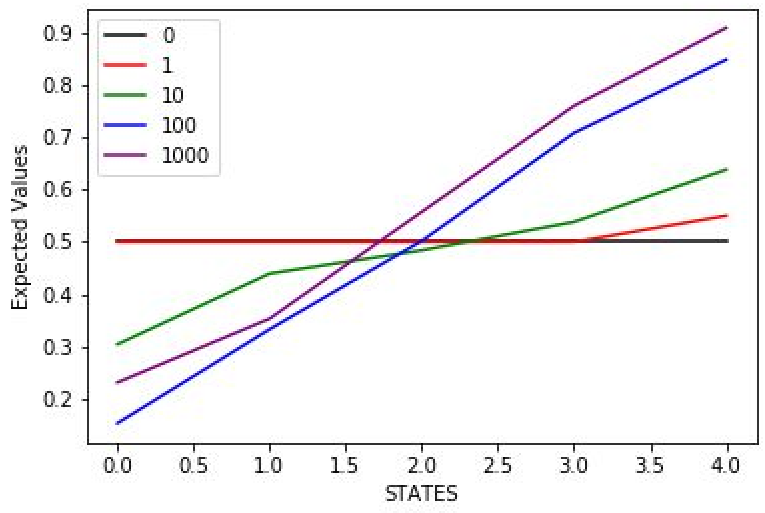
Whenever we change only a few states from the whole state set, the MC sequence will change and we'll have to learn everything again, which means we will need to run random walks again and again, even though the state which are unchanged will have learnt some estimate in the previous episodes (old states)
Let's suppose we have a set of states and we only change the first state, now in MC as the sequence has been updated, we will have to initiate random walks again. The already learn action values or state values of unchanged states goes to waste here. But if TD is used then we

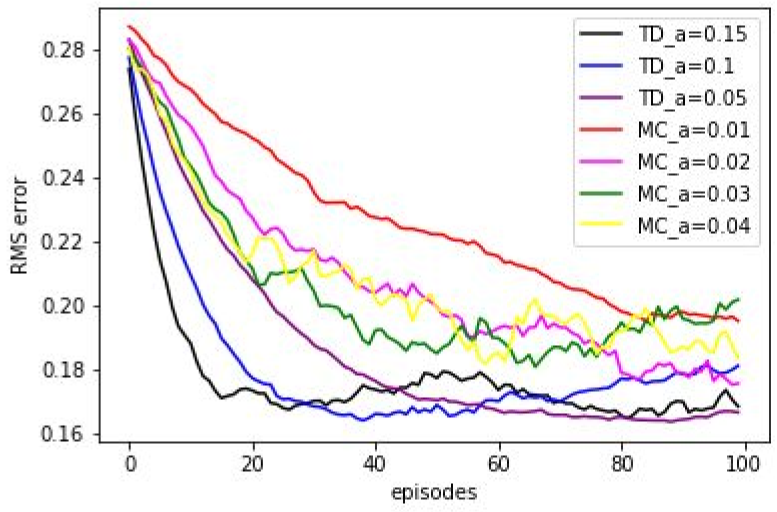
do not need to for all the states, and even if we do, the convergence will be faster. The changed state that is the first state will be affected by the estimate of the next state, as TD only looks one step ahead. The other state may also have some change but this change will not propagate to states near the termination. So only few updates will be made using TD versus MC.

Question 6:

1. The value of state A changes because the current value of V(A) = 0.5, if action is right then V(B) = 0.5 but if action is left then V(terminal state) = 0; as left and right have equal probability, V(A) can be reduced by 0.05, as the error in estimate = 0.5 and alpha = 0.1, so alpha*error = 0.05.
   Other states except E, have all neighbouring states with V = 0.5, so they remain the same.
2. No, even by using a wider range of alpha, the performance of TD outmatches that of MC. Also we can see that MC is not decreasing significantly with decrease in alpha, which means it'll saturate soon.
   There are values of alpha for which both of them perform better, as alpha is like a step size towards the expected value and the smaller this step size is the more chances are for the value to converge but it'll take more time too.

3. At high alpha the change in the value becomes higher, so similar to gradient descent, the new value is not the optimal value due to large step size. This makes the error oscillate about zero, as it keeps shifting above and below the zero value due to big step size. This usually occurs, tuning of alpha is something which is done heuristically and not analytically if alpha is a constant, so nothing wrong with the value function





Question 7:
Q learning is better than SARSA as can be seen

Question 8:

Assuming GREEDY policy in both SARSA and Q learning, as this is not specified, if only greedy in Q learning and not in SARSA, then of course both will be different as one is greedy while other is epsilon greedy

When we take greedy policy instead of epsilon greedy, then the SARSA and Q learning become similar, as now both calculate new state-action value using greedy policy in the discount factor. Earlier SARSA used epsilon greedy to find A_(t+1) while Q learning used argmax_{a}(Q(S_t, a)), Q_learning used greedy policy, so changing both to greedy will make them same.
They will become the same, actions will be selected from the same policy, states will be updated with the same weights.

Attachments:

We can update the Q values as follows: $\boxed{Q1}$

$Q(S_t, A_t) \leftarrow$ average (Returns(S_t, A_t))

$\Rightarrow$ if we take the returns as $G_1, G_2, \ldots$

then: $Q_n = \dfrac{~~Q_1~~ + G_1 + G_2 + \ldots G_n}{n}$

Now $Q_{n-1} = (G_1 + G_2 + \text{—} G_{n-1})/(n-1)$

$\Rightarrow Q_n = [(n-1)Q_{n-1} + G_n]/n$

$Q_n = Q_{n-1} + \frac{1}{n}\{G_n - Q_{n-1}\}$

$$\Rightarrow \left\{ Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \underset{\downarrow}{\frac{1}{n}} \left[\text{Returns}(S_t, A_t) - Q(S_t, A_t)\right] \right\} \checkmark\checkmark$$

$n(s,a) \rightarrow$ number of times the state action pair is visited where $s = S_t$, $a = A_t$

Initialize:

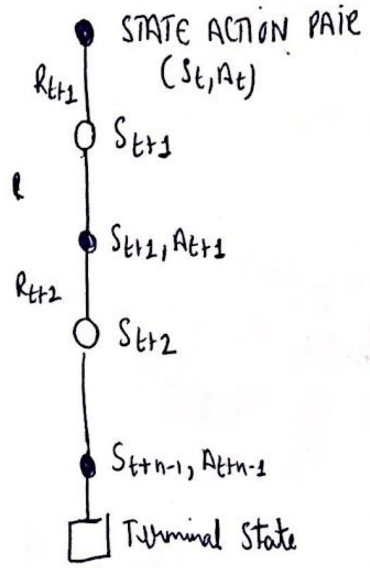
$\pi(s) \in A(s)$

$Q(s,a) \in R$

~~Ret~~ $n(s,a) = 0 \quad \forall\ s \in S,\ a \in A(s)$

```
Loop forever (for each episode):
Choose $S_0 \in S$, $A_0 \in A(S_0)$ randomly s.t probability > 0
Generate an episode from $S_0, A_0$ following $\pi$: $S_0, A_0, R_1 \ldots R_T$
 $G \leftarrow 0$
 Loop for each of episode : $t = T-1, T-2, \ldots 0$
 $G \leftarrow \gamma G + R_{t+1}$
 Unless pair $S_t, A_t$ in sequence: $S_0, A_0 \ldots$
  $n(S_t, A_t) \leftarrow n(S_t, A_t) + 1$
  $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{n}\{\text{Returns}(S_t, A_t) - Q(S_t, A_t)\}$
  $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$
```

Q2)



Q3) $$Q(s,a) = \frac{\sum_{t \in J(s,a)} \rho_{t:T(t)-1} G_t}{\sum_{t \in J(s,a)} \rho_{t:T(t)-1}}$$

where $s = S_t$ and $a = A_t$