Q0, Yes. Either anonymized or not is okay.

Q1,

(a) Input: policy π

Initialize: $V(s)=0$, for all $s \in S$.

Count(s) $\leftarrow$ 0 for all $s \in S$

Loop forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, S_1, \cdots S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \cdots 0$.

$G \leftarrow \gamma G + R_{t+1}$

Unless S_t appears in $S_0, S_1, \cdots S_{t-1}$.

$Count(S_t) \leftarrow Count(S_t) + 1$

$V(S_t) \leftarrow V(S_t) + \frac{1}{Count(S_t)}(G - V(S_t))$

(b)

Initialize: $\pi(s) \in A(s)$ arbitarily for $s \in S$

$Q(s,a)$ arbitarily for all $s \in S$, $a \in A(s)$

$Count(s,a) \leftarrow 0$ for all $s \in S$, $a \in A(s)$

Loop forever (for each episode):

Choose $S_0 \in S$, $A_0 \in A(S_0)$ randomly

Generate an episode from S_0, A_0 following π: $S_0, A_0, R_1, \cdots S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \cdots 0$

$G \leftarrow \gamma G + R_{t+1}$

Unless the pair (S_t, A_t) appears in $S_0, A_0 \cdots S_{t-1}, A_{t-1}$.

$Count(S_t, A_t) \leftarrow Count(S_t, A_t) + 1$

$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{Count(S_t, A_t)}(G - Q(S_t, A_t))$

Q2.

(a) No. Since every state in a single episode only appears once, every-visit mc and first-visit mc are equivalent in this case. Thus their result would be the same.

(b)

$S_0, a_0, r_1, S_1, a_1, r_2, S_2 \cdots S_9, a_9, r_{10}, S_T$

$G_{10} = 0,$

$G_9 = r_{10} + \gamma \cdot G_{10} = 1$

$G_8 = r_9 + \gamma \cdot G_9 = 2.$

$\vdots$

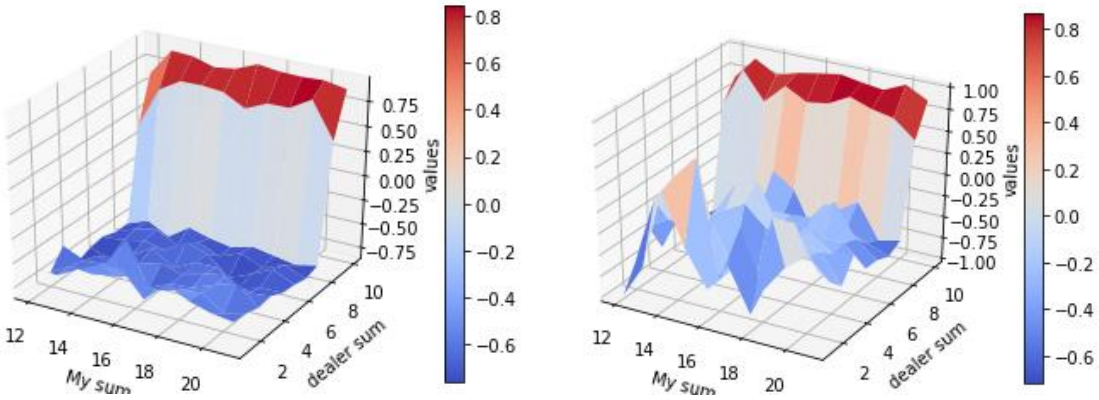
$G_1 = r_2 + \gamma G_2 = 10$

first-visit MC: $V(S) = 10$

every-visit MC: $V(S) = \frac{(1+\cdots 10)}{10} = 5.5$

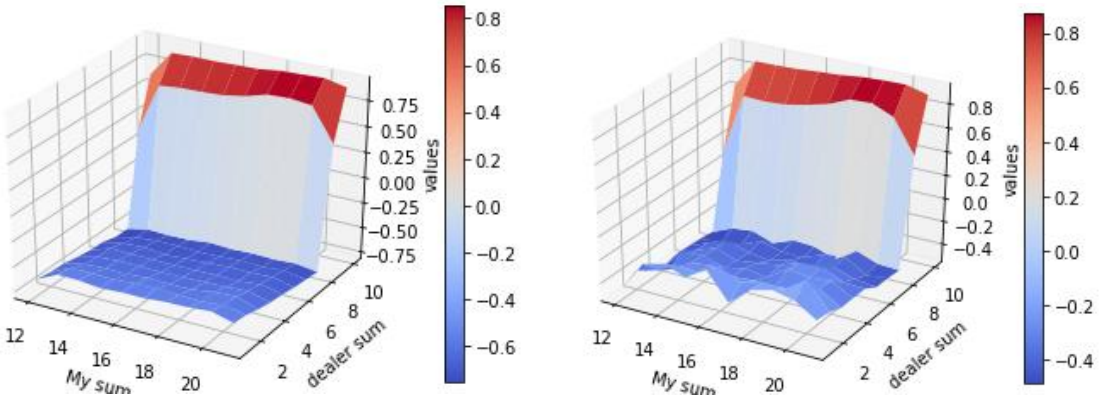
Q3

(a).



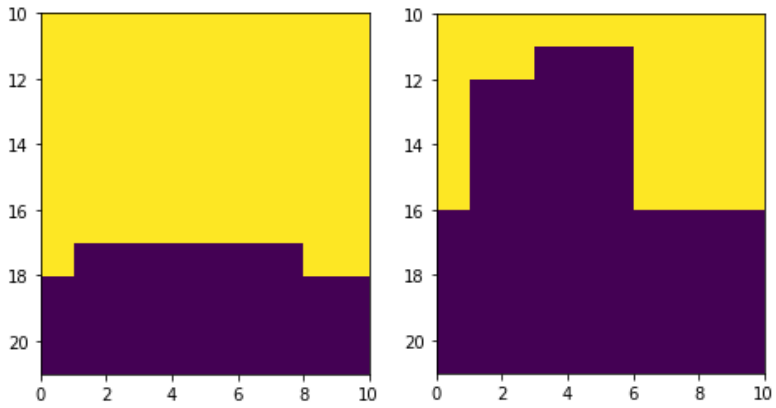
State value estimation after 10000 episodes

(left: no usable ace; right: usable ace)



State value estimation after 500000 episodes
(left: no usable ace; right: usable ace)

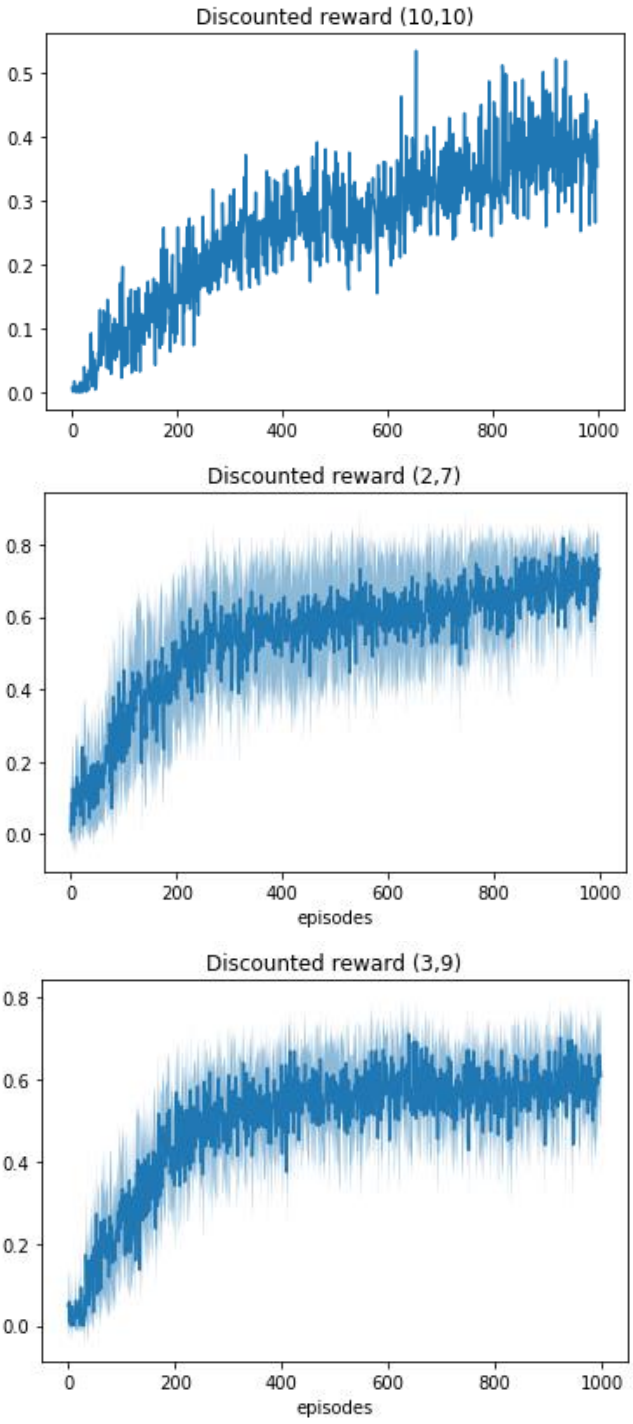
(b)



Policy after 3 million episodes for no usable ace(left) and usable ace(right)

Q4

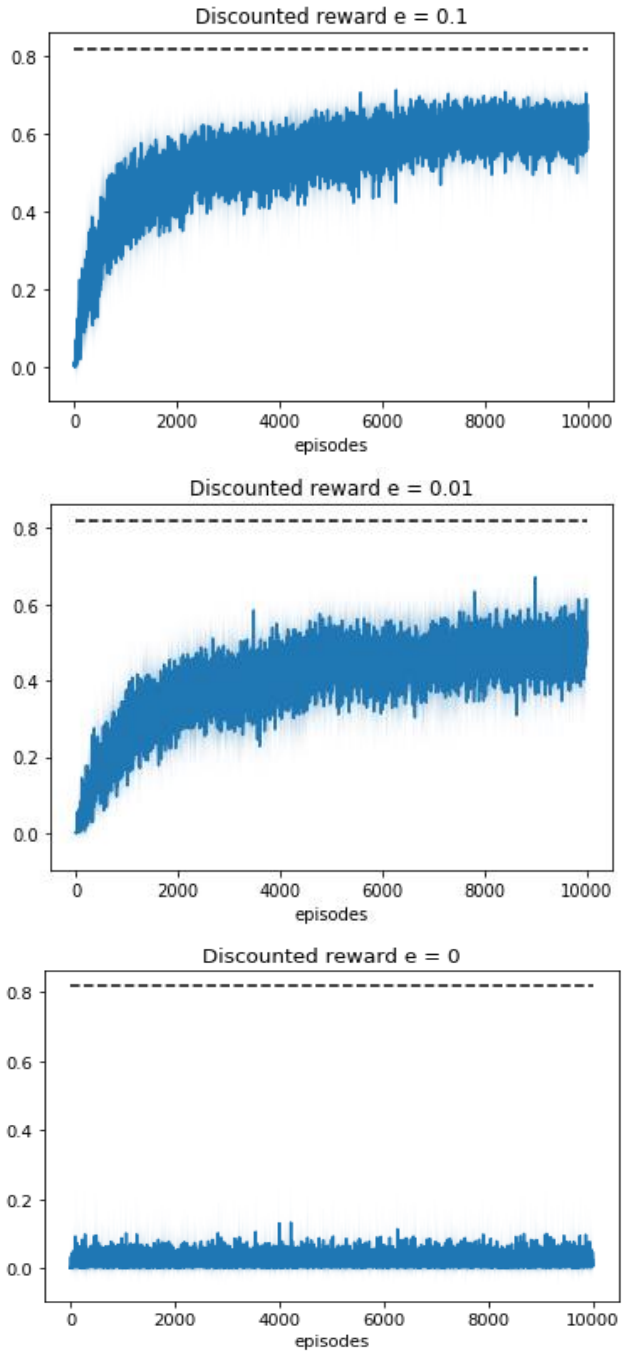
(a)



Learning curves (Discounted reward) for different goals

The rising curves suggest this algorithm is learning.

(b) best performance calculated as 0.99^20 = 0.818



Learning curves (Discounted reward) for different epsilons

(c) exploring start makes the probability of seeing every state larger than 0, thus even the policy is greedy we could still make improvement on the existing policy instead of stuck on the first states with non-negative(positive) value forever.

Q5

(a)

$$V_{n+1} = \frac{\sum_{k=1}^{n} W_k G_k}{\sum_{k=1}^{n} W_k}$$

$$= \frac{\sum_{k=1}^{n-1} W_k G_k + W_n G_n}{\sum_{k=1}^{n} W_k}$$

$$= \frac{\sum_{k=1}^{n-1} W_k G_k}{\sum_{k=1}^{n-1} W_k} \cdot \frac{\sum_{k=1}^{n-1} W_k}{\sum_{k=1}^{n} W_k} + \frac{W_n G_n}{\sum_{k=1}^{n} W_k}$$

$$= V_n \cdot \frac{(\sum_{k=1}^{n} W_k - W_n)}{\sum_{k=1}^{n} W_k} + \frac{W_n G_n}{\sum_{k=1}^{n} W_k}$$

$$= V_n - V_n \cdot \frac{W_n}{\sum_{k=1}^{n} W_k} + G_n \cdot \frac{W_n}{\sum_{k=1}^{n} W_k}.$$

$$= V_n + \frac{W_n}{\sum_{k=1}^{n} W_k} (G_n - V_n)$$

$\because C_n = \sum_{k=1}^{n} W_k$

$\therefore V_{n+1} = V_n + \frac{W_n}{C_n}(G_n - V_n) \qquad n \geq 1$

(b) It's correct, because π is greedy, when $A_t = \pi(S_t)$, $\pi(A_t|S_t)$ will equal to 1. and when $A_t \neq \pi(S_t)$, $\pi(A_t|S_t)$ will equal to 0, and then W will be 0 and nothing will be updated afterwards. Therefore, we should exit the for loop and thus this algorithm is correct.

Q6

(a)

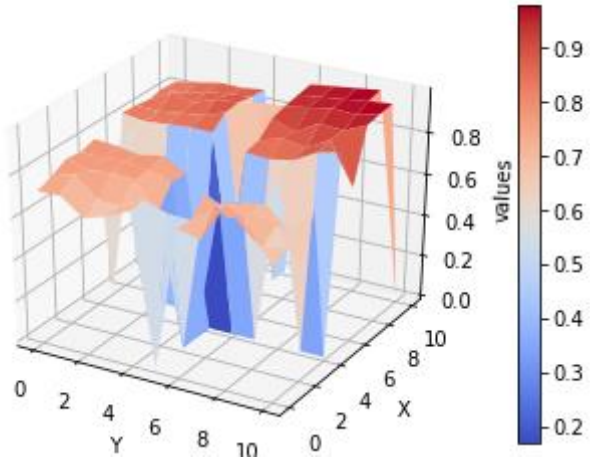
(b)

```
In [534]: gw.aMap
Out[534]:
[['r', 'd', 'l', 'r', 'd', 'l', 'u', 'u', 'l', 'd', 'l'],
 ['d', 'd', 'u', 'r', 'u', 'l', 'r', 'u', 'd', 'u', 'r'],
 ['r', 'r', 'd', 'r', 'r', 'l', 'u', 'r', 'u', 'r', 'u'],
 ['r', 'd', 'r', 'd', 'd', 'r', 'u', 'l', 'r', 'd', 'l'],
 ['u', 'r', 'l', 'l', 'l', 'd', 'r', 'l', 'u', 'r', 'd'],
 ['d', 'r', 'r', 'r', 'd', 'l', 'u', 'u', 'u', 'u', 'u'],
 ['r', 'r', 'u', 'r', 'r', 'd', 'd', 'l', 'r', 'r', 'r'],
 ['u', 'u', 'u', 'r', 'r', 'l', 'r', 'r', 'u', 'r', 'd'],
 ['d', 'u', 'l', 'u', 'u', 'u', 'u', 'r', 'r', 'r', 'r'],
 ['d', 'u', 'u', 'l', 'd', 'u', 'r', 'u', 'u', 'u', 'r'],
 ['l', 'l', 'u', 'l', 'd', 'u', 'u', 'u', 'u', 'u', 'r']]
```

Policy obtained from Monte-Carlo control (map has been flipped to fit in the matrix form; Break the ties randomly)

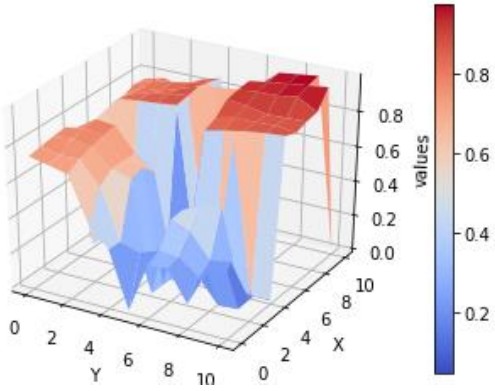
I find that the actions near the goal are most optimal, but in one of the rooms which doesn't contain goal and start point always has some actions that haven't been changed much from the initial policy (policy are initial randomly). I think it's because whenever the agent learns from a successful trajectory it tends to stick on this trajectory and thus the other potential trajectory goes through the other way is ignored, and I think the wrong part will be evaluated correctly if given more episodes.

(c) Since Q value is hard to visualize, I calculated state value and stored the Q value in separate txt files.



Off-policy MC prediction on V_π
(calculated on highest Q value for each state)

(d) Since Q value is hard to visualize, I calculated state value and stored the Q value in separate txt files.



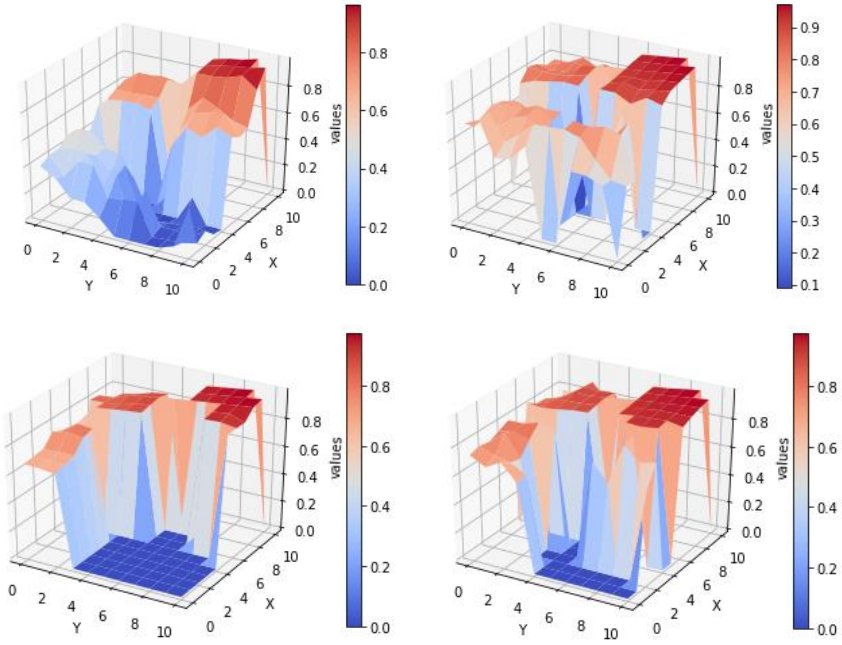
On-policy MC prediction on V_π
(calculated on highest Q value for each state)

(e)

I found that the off-policy estimation is sharper and closer to optimal value comparing to on-policy MC, and also has a better estimation of those states where the agent ignored in the on-policy MC control (the room that agent seldom/never goes in) while in on-policy MC many states are not updated under on-policy algorithm and thus have lower values.

I think off-policy has better estimation in all the 4 rooms because the behavior policy is soft greedy, and thus have the probability of reaching every valid state in the environment even the ones that seldom get updated, while in on-policy MC prediction the policy is greedy and deterministic, and thus we won't have much chance of reaching to those states.

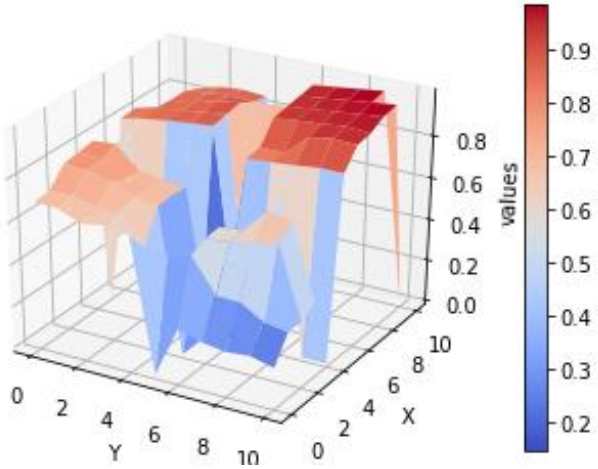
(f)



Comparison between on-policy (left) and off-policy (right)
(calculated on highest Q value for each state)

Same thing happens that off-policy estimation is sharper but closer to accurate state value especially on the room that the policy decides not to go into.

(g)



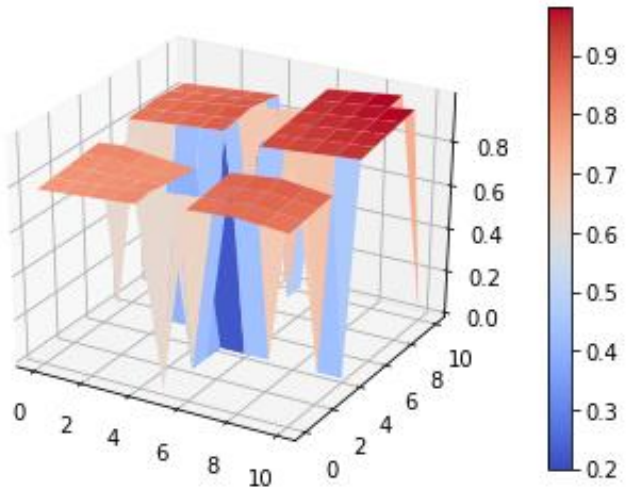
Dynamic programming prediction on V_π
(calculated on highest Q value for each state)

The state value estimation using DP is smoother than off-policy MC and more accurate than on-policy MC (especially on those states which are in the room that the policy never goes in). So off-policy is more accurate in reflecting true state values since it covers every state in the environment, while on-policy couldn't.

(h)

```
In [535]: pi_star
Out[535]:
[['u', 'u', 'u', 'u', 'l', 'd', 'l', 'd', 'd', 'l', 'r'],
 ['d', 'd', 'd', 'd', 'u', 'l', 'u', 'r', 'r', 'r', 'l'],
 ['l', 'l', 'l', 'l', 'l', 'r', 'l', 'l', 'r', 'l', 'l'],
 ['r', 'r', 'r', 'r', 'u', 'u', 'r', 'r', 'r', 'd', 'd'],
 ['u', 'r', 'd', 'l', 'l', 'd', 'u', 'r', 'd', 'd', 'u'],
 ['d', 'u', 'd', 'l', 'l', 'l', 'u', 'l', 'l', 'u', 'd'],
 ['u', 'l', 'l', 'u', 'r', 'r', 'l', 'r', 'l', 'd', 'd'],
 ['l', 'l', 'u', 'r', 'd', 'r', 'r', 'd', 'u', 'r', 'r'],
 ['l', 'd', 'u', 'r', 'r', 'r', 'd', 'd', 'l', 'd', 'u'],
 ['u', 'l', 'l', 'u', 'd', 'r', 'l', 'l', 'd', 'u', 'r'],
 ['l', 'd', 'l', 'd', 'd', 'l', 'd', 'r', 'u', 'd', 'l']]
```

Policy obtained from Monte-Carlo control (map has been flipped to fit in the matrix form; Break the ties randomly)



Dynamic programming prediction on V_*
(calculated on highest Q value for each state)

I think MC finds a good policy, because even it doesn't get the optimal state value it still gets the relative good/bad between different state, which is crucial in control phase and to find an optimal policy.