Q0. Yes. Either anonymized or not is okay.

Q1.

(a).

No, because Dyna-Q will update every transition that it ever seen in the planning phase at each step, but multistep bootstrapping could only update 'n' states along the trajectory. There will be an improvement if we compare the performance of n-step method to one step method, but it still won't be as good as Dyna-Q.

(b)

It might perform better, since usually the optimal n-step method will be somewhere between one step method and Monte-Carlo method, and use appropriate number n will definitely help to update more efficiently and make the model converge faster. However, there isn't a way to find the best n directly, and n-step method is also more computation-consuming.

Q2.

(a)

Because Dyna-Q+ involves more exploration than Dyna-Q, since they use same epsilon value but Dyna-Q+ has extra exploration bonus to further increase the action value of those actions based on how long they haven't been chosen.

In the first phase of both blocking and shortcut experiments, the one focuses more on exploration will raise the curve earlier, so Dyna-Q+ outperform Dyna-Q in the first phase. In the second phase of blocking experiment, Dyna-Q+ finds the new path faster than Dyna-Q also because of the more exploration it has. In the second phase of shortcut experiment, Dyna-Q+ will encourage the agent to try the actions that haven't been chosen for longer times, and thus it's able to find the shorter path appears after 3000 steps, but Dyna-Q doesn't have the bonus to encourage more exploration and it will stick to the current solution forever.

(b)

Since Dyna-Q+ focuses more on exploration, in the first phase of the experiment the cumulative reward curve of it will rise up earlier than Dyna-Q, however since exploration will result in deviation from optimal policy and thus the more exploration the worse performance when it converges. As an result, the performance of Dyna-Q will gradually outperform Dyna-Q+ if given longer time, and that's why the difference narrowed.

Q3.

(a)

Dyna-Q+:

Initialize ***Q(s,a)*** and ***Model(s,a)*** and ***T(s,a) = 0*** for a ***s*** and ***a***

Loop forever:

S = current state

A = ***epsilon-greedy(S,Q)***

Take action ***A***: observe resultant reward, ***R***, and state, ***S***

T(S, A) = 0

Q(S,A) = ***Q(S,A)*** + alpha * (***R*** + gamma* max_a***Q(S',a)*** – ***Q(S,A)***)

Model(S,A) = R, S' (assuming deterministic environment)

Loop repeat ***n*** times:

S = random previously observed state

A = random action that could be taken at state ***S***

If action A is previously observed:

R, ***S'*** = ***Model(S,A)***

Else:

R = 0, ***S'*** = ***S***

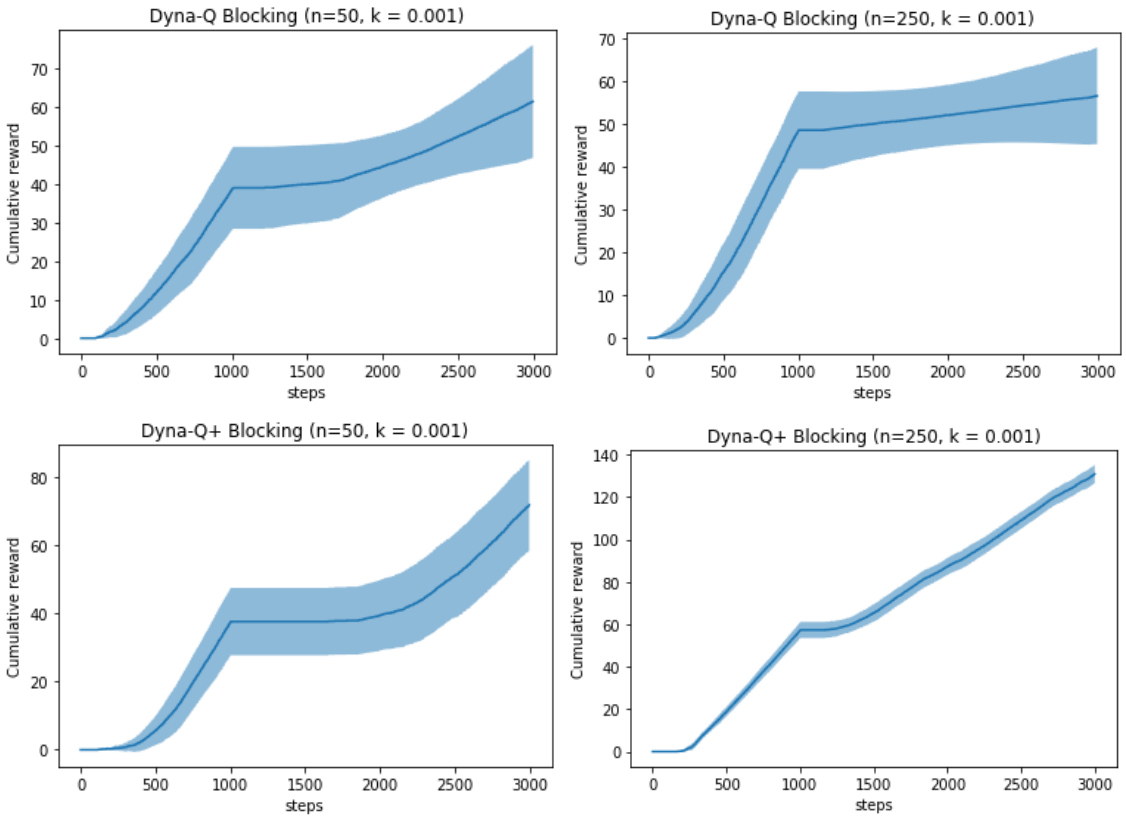
Bonus = tau * sqrt(***T(S, A)***)

Q(S,A) = ***Q(S,A)*** + alpha * (***R*** + Bonus + gamma* max_a***Q(S',a)*** – ***Q(S,A)***)

T(S,A) += 1 for all state ***S*** and action ***A***

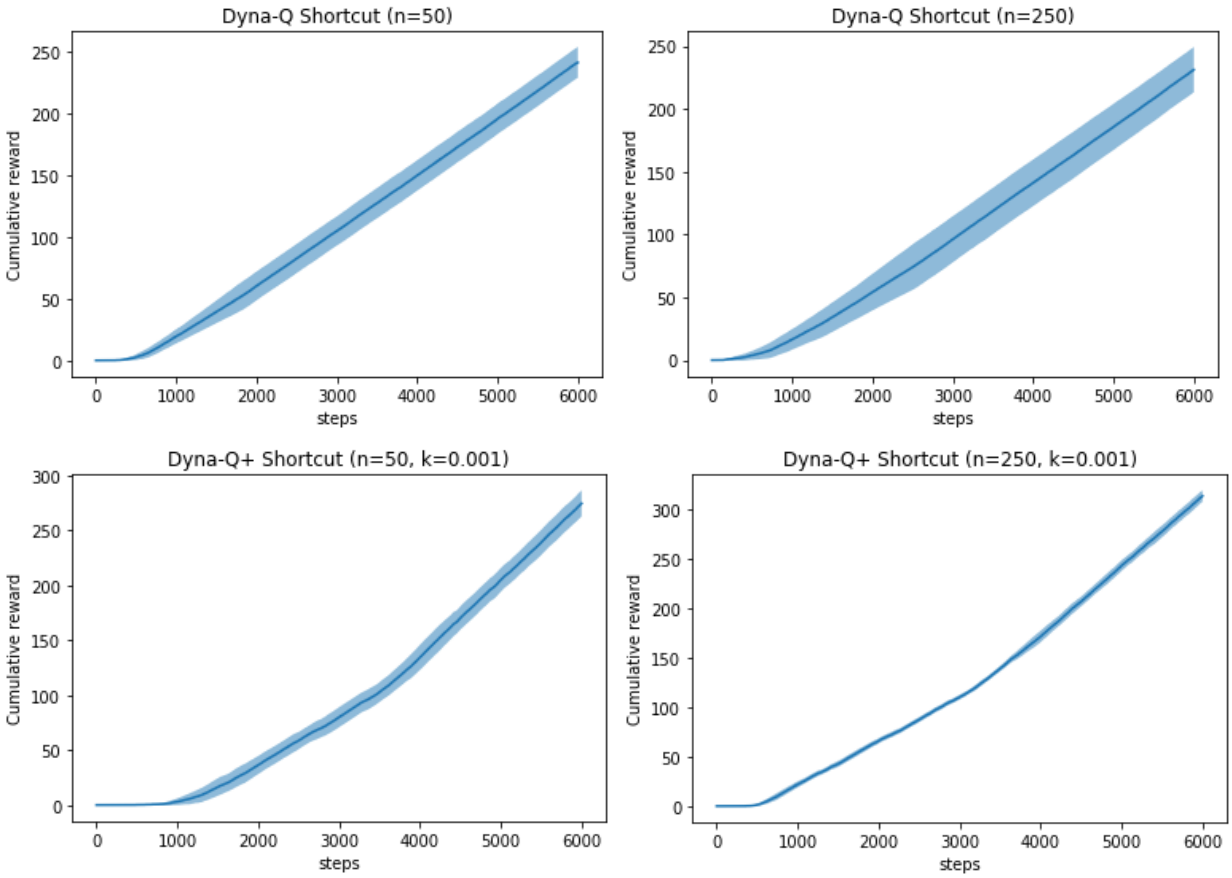
(b)

Blocking Experiment (typo: k is not involved in Dyna-Q):



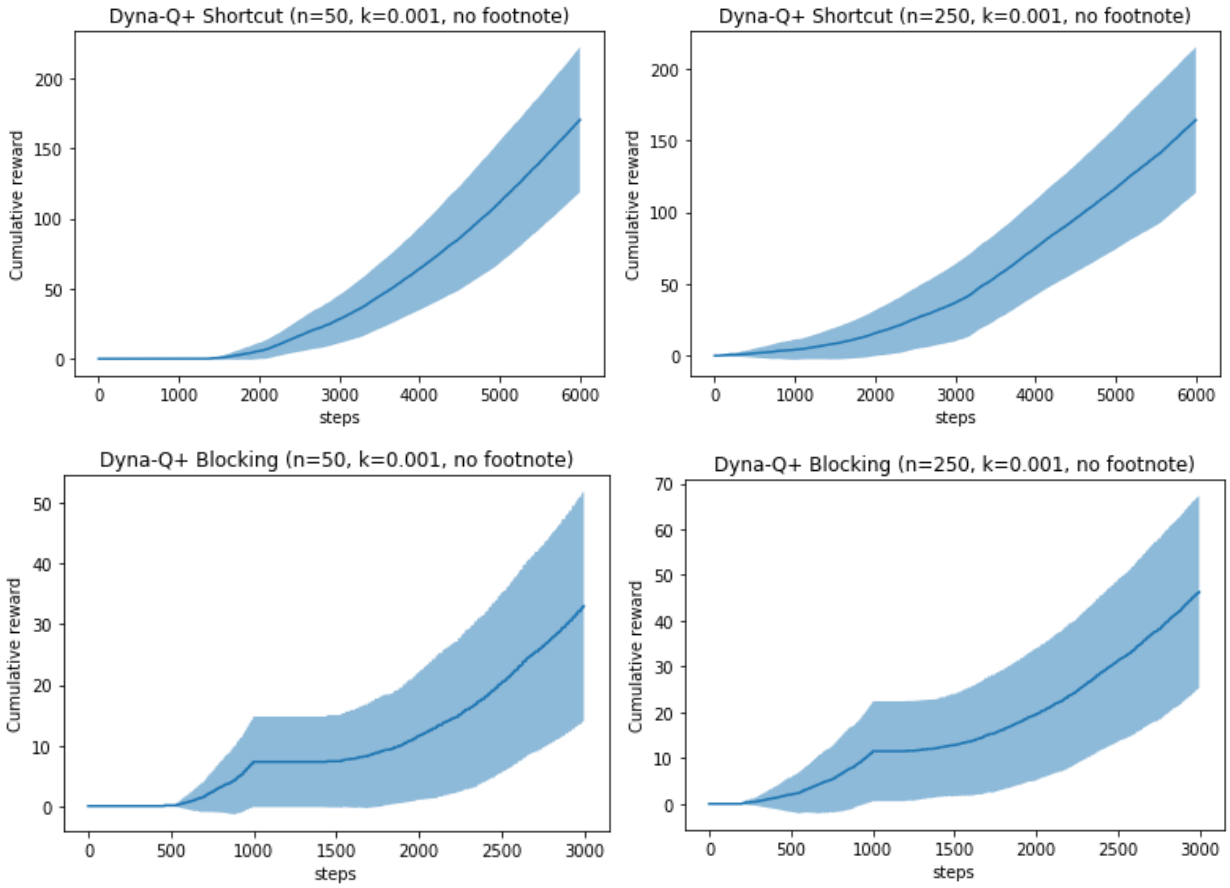
Dyna-Q+ generally performs better in both cases. It finds the new path much faster and thus achieves higher cumulative reward.

Shortcut experiment



Dyna-Q could always find the shortcut no matter what n it uses, and Dyna-Q never find the shortcut, as suggested in the book.

(c)




```
[[0.6  0.58 0.56 0.53 0.52 0.51 0.   1.02 1.02]
 [0.63 0.6  0.57 0.54 0.52 0.51 0.54 1.02 1.02]
 [0.66 0.   0.   0.   0.   0.   0.   0.   1.02]
 [0.69 0.72 0.75 0.79 0.82 0.86 0.9  0.93 1.02]
 [0.72 0.75 0.77 0.8  0.84 0.88 0.92 0.97 1.01]
 [0.71 0.77 0.8  0.84 0.87 0.91 0.96 1.   0.  ]]
```

Max Q value of each state at the end of blocking experiment

The footnote does matter. After removing the footnote suggestion, the learning curves rise very slowly, and the variance is huge. Though the result shows that it does find the shortcut, it isn't easy to assert it just by looking at the learning curve because it converges too slow.

After removing the suggestion from footnote, those actions that haven't been observed will not be update during planning phase, and it will only start to appear in planning phase after the agent actually take that action in real environment, and it's only depended on the exporation nature of epsilon, which make it learns slowly when meet a new environment.

Q4.

(a)

In both methods the extra bonus will encourage the agent to chose those actions that haven't been chosen for long time, but UCB is used for k-arm bandit problem and it doesn't have transitions between different states, and it only need to pick the action once to know how good is the action now. In Dyna-Q+ the bonus has actually been used in action value computation. If in the planning phase the update time n is large enough, the action value will be infinity (given the learning target ***R*** + Bonus + gamma* $\boldsymbol{max_a Q(S',a)}$).

UCB:

Pros: less space complexity, since we only need to record the time that every state-action pair last appears

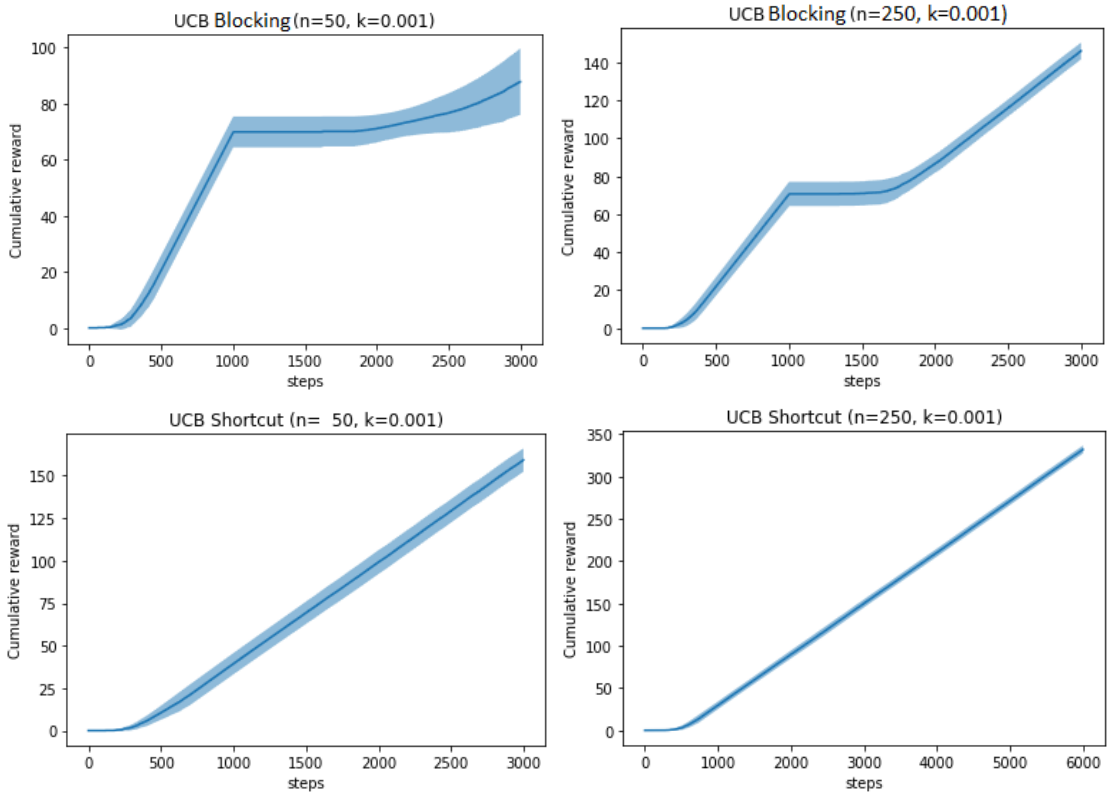
Cons: cannot handle state transition well, since it doesn't update action value based on the next state it reaches to.

Dyna-Q+:

Pros: Bootstrap update based on following state, could handle state transition better than UCB.

Cons: Extra bonus reward makes all action value keep increasing, which makes the action value deviate from the true value and not realistic.

(b)



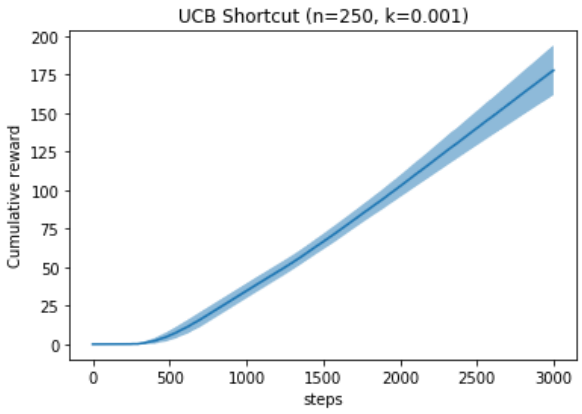
It does match my prediction. In blocking experiment the behavior is similar to Dyna-Q method, and its cumulative reward is less than Dyna-Q+'s, but higher than Dyna-Q's.

If we only use the bonus reward in action selection phase, then there won't be update on action value based on bootstrapping from S' in planning phase. Therefore, even if the agent chose an action that rarely be chosen, the next time when the agent meet this state, it probably not chose that action again, since the term **tau * sqrt(*T(S, A)*)** has been reset to 0 and thus has lower **reward + bonus** value than other actions in this state. As a result, if the new shortcut deviates a lot form the optimal path the agent found before 3000 step, it will take very long time for the agent to try the shortcut once. That's why the UCB doesn't seem to find the shortcut at all, and I believe it's able to find the shortcut if given longer running time.

(c)

Since the reason of not finding the shortcut is the path towards shortcut deviates too much from the current optimal path, I guess that if the new shortcut that appears at 1000 step deviate less will help UCB to find it, and it actually does.

I modified the shortcut environment by shorten the total step to 3000 and make the shortcut appears at 1000 step. The learning curve does become steeper after the shortcut appears. It isn't very obvious so we can check at which step does the agent complete an episode, and the result suggest that it on takes 10 step on average to reach the goal, which means it has already found the shortcut, because there isn't another way to reach the goal within 16 steps.




```
2862
2872
2882
2892
2902
2912
2922
2933
2943
2953
2963
2973
2983
2993
```

It shows that when the new optimal policy doesn't deviate too much, UCB will be able to capture the change and find the new optimal policy, and it will outperform Dyna-Q+ since it does not use soft-greedy policy, even if there's an exploration bonus for UCB. However if the new optimal policy deviates too much from the previous one then UCB will take a really long time to find it and thus be outperformed by Dyna-Q+.

Q5.

(a)

For **stochastic** environment, we can store transitions instead of S', R for a certain state-action pair, and during planning phase we will use the collected transitions to do the random update.

Dyna-Q+(stochastic):

Initialize ***Q(s,a)*** and time function ***T(s,a) = 0*** for a ***s*** and ***a***, and transition buffer ***Transition = []***

Loop forever:

S = current state

A = ***epsilon-greedy(S,Q)***

Take action ***A***: observe resultant reward, ***R***, and state, ***S'***

T(S, A) = 0

Q(S,A) = ***Q(S,A)*** + alpha * (***R*** + gamma* max_a***Q(S',a)*** – ***Q(S,A)***)

Transition.append(tuple(S, A, R, S'))

Loop repeat ***n*** times:

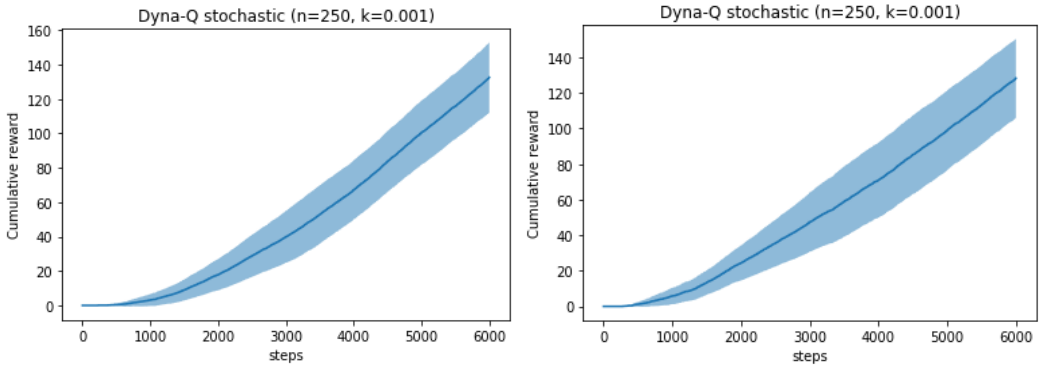
Random choose a transition tuple from ***Transition*** buffer

S, A, R, S' = ***tuple(S, A, R, S')***

Bonus = tau * sqrt(***T(S, A)***)

Q(S,A) = ***Q(S,A)*** + alpha * (***R*** + Bonus + gamma* max_a***Q(S',a)*** – ***Q(S,A)***)

T(S,A) += 1 for all state ***S*** and action ***A***



Left: Dyna-Q with stochastic environment, Right: Dyna-Q with stochastic & changing environment

I use the shortcut experiment to test the stochastic Dyna-Q's ability to handle stochastic and stochastic&changing environment. It appears that it performs good in stochastic environment while it doesn't find the shortcut appears at 3000 steps.

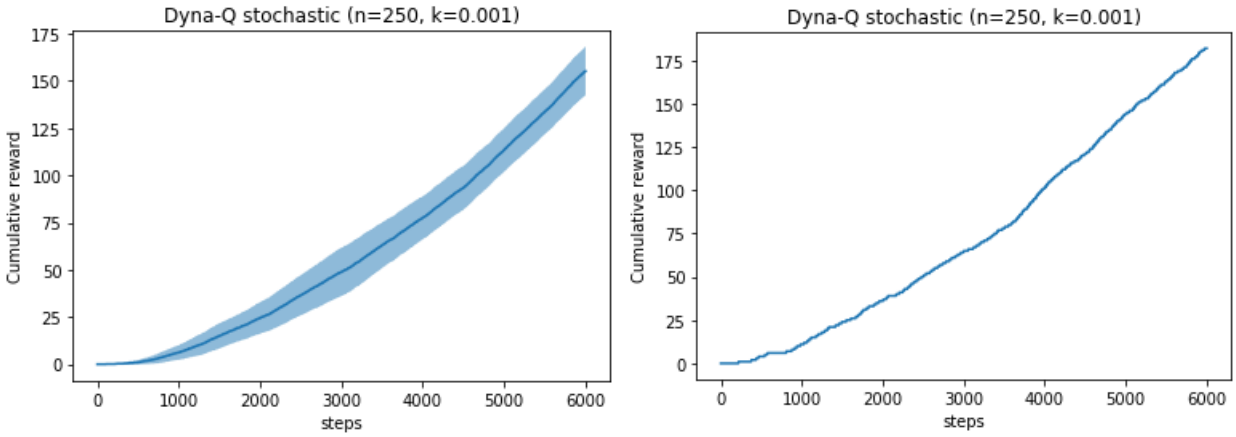
In order to also handle changing environment, we should let the agent have some sort of forgetting mechanism. In this case, I let the agent only record the latest 800 transitions, and any transition before that will be forgotten. Therefore it will keep catching up with the latest environment change it perceives.

Dyna-Q+(stochastic):

Initialize ***Q(s,a)*** and time function ***T(s,a) = 0*** for a ***s*** and ***a***, and transition buffer ***Transition = []***

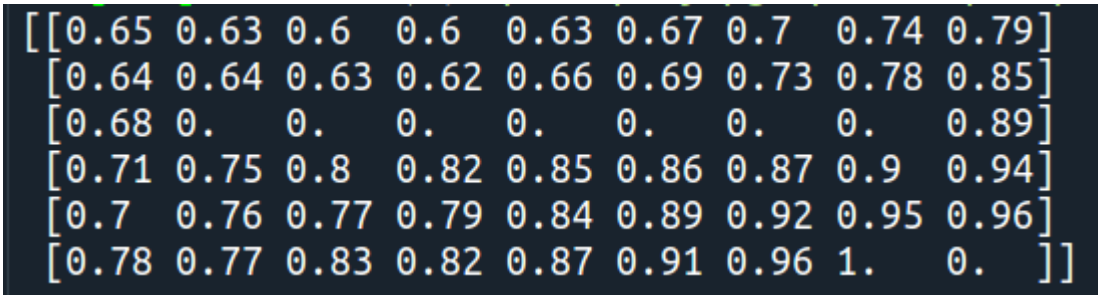
Loop forever:

> ***S*** = current state
>
> ***A*** = ***epsilon-greedy(S,Q)***
>
> Take action ***A***: observe resultant reward, ***R***, and state, ***S'***
>
> ***T(S, A)*** = 0
>
> ***Q(S,A)*** = ***Q(S,A)*** + alpha * (***R*** + gamma* max_a***Q(S',a)*** – ***Q(S,A)***)
>
> ***Transition.append(tuple(S, A, R, S'))***
>
> If transition is longer than 800: drop the first transition tuple from the ***Transition*** buffer
>
> Loop repeat ***n*** times:
>
> > Random choose a transition tuple from ***Transition*** buffer
> >
> > ***S, A, R, S'*** = ***tuple(S, A, R, S')***
> >
> > Bonus = tau * sqrt(***T(S, A)***)
> >
> > ***Q(S,A)*** = ***Q(S,A)*** + alpha * (***R*** + Bonus + gamma* max_a***Q(S',a)*** – ***Q(S,A)***)
>
> ***T(S,A)*** += 1 for all state ***S*** and action ***A***



Left: 10 trials average, right: single run

We can see from the curves that it gradually rises up after 3000 step. It's a symbol that it has found the shortcut. There isn't a clear turning point because the variance is fairly high due to the stochastic environment. We can look at the Q value map to further prove it.



We can see that the value at [2, 8] has been updated to a reasonable number, and there is a path along the start point to the goal (the map has been flipped). It proves that the new Dyna method is able to handle both stochastic environment and changing environment.