Question 1: Monte-Carlo Control in Easy21

a. To implement Monte-Carlo control in Easy21, an iterative on-policy ϵ-greedy first-visit MC control algorithm is used. This iterative algorithm was chosen over a batch approach as it has lower time complexity and memory requirements, both of which are important as the number of episodes used will be very large and the process will run on a system with moderate computing power. The first-visit method is preferred over every-visit as it has been shown that for a large enough number of episodes it will result in lower average MSE in approximating the optimal value function[1]. Finally, this algorithm does not require exploring starts, making it applicable to scenarios where starting from any state is not feasible.
The State-Action Value function is updated after each new trace τ has been generated. For all unique state-action pairs appearing in τ: $\hat{Q}(s,a) \leftarrow \hat{Q}(s,a) + \alpha[R(s,a) - \hat{Q}(s,a)]$ where R(s,a) is the return following the first visit to the state-action pair (s, a) (note that since all non-zeros rewards occur at the end of the episode and $\gamma = 1$, R(s,a)=r_{episode} for all (s, a) in τ). The parameter α is chosen to be equal to the reciprocal of the cumulative number of first visits to the state-action pair ($\alpha = 1/N(s,a)$, which satisfies the Robbins-Monroe conditions). Initializing $\hat{Q}(s,a)$ as zero for all state-action pairs and setting $\alpha = 1/N(s,a)$ makes the update rule an online average of the returns following the first visit to the state-action pair. The policy is improved by setting $\hat{\pi}(s,a) = 1 - \epsilon/2$ if a maximizes $\hat{Q}(s,a)$, $\hat{\pi}(s,a) = 0.5$ if $\hat{Q}$(s,'hit') is equal to $\hat{Q}$(s,'stick') and $\hat{\pi}(s,a) = \epsilon/2$ otherwise. Here, $\epsilon = \frac{N_0}{N_0 + \sum_{a \in \mathcal{A}} N(s,a)}$ which results in an ϵ-greedy policy that is GLIE. The parameter N0 is used to balance exploration vs exploitation for the number of episodes to be run (given time and computational constraints). As N0 increases, the exploration increases but so does the time required for convergence. For smaller values of N0, convergence is quicker but the results likely inaccurate. The value of N0 will be determined by inspecting learning curves for different N0 (see Q1b).

b. The algorithm is run for 10^6 episodes, as it produces good results without being overly time consuming. Training stops every few episodes and N simulations of the game are run using the current policy. The mean and standard deviation of the rewards are then calculated. Since the standard deviation is approximately 1 and the mean in the order of 0.01, N must be in the order of 10^4 for a learning curve with reasonable SNR (the standard deviation of the mean decreases approximately with $1/\sqrt{N}$). Hence, due to limitations in computational power, testing is performed every $5 \cdot 10^4$ episodes. Empirically it was found that for N=$16 \cdot 10^4$, the plot becomes sufficiently clear. Nine learning curves are plotted, three separate instances for each of N0 = 10, 100 and 1000.

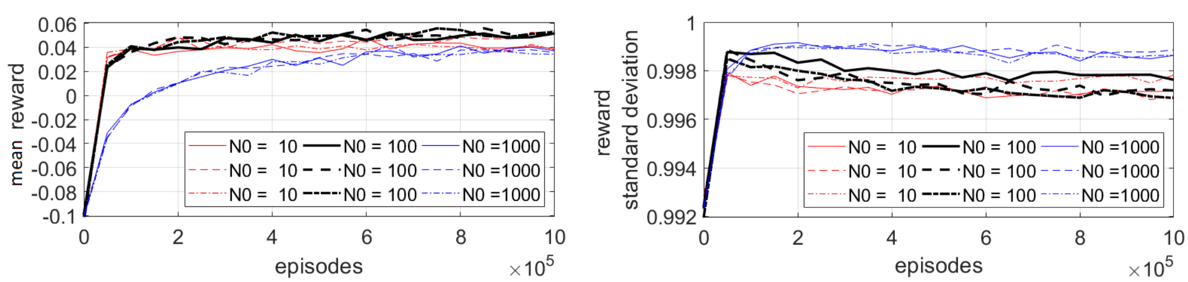

Figure 1: Learning Curves for N0={10,100,1000}. N0=100 offers the best overall performance. N0=10 converges faster but 2 out of 3 times converges to a worse policy than N0=100. When N0=1000, convergence has not occurred yet after 10^6 episodes.

[1]Singh, S.P. & Sutton, R.S. Mach Learn (1996) 22: 123. https://doi.org/10.1007/BF00114726

c. The estimate for the optimal value function after 10^6 episodes is shown in the figure below. It is notable that the value of states where the player sum is close to 21 have value of approximately 1. Additionally, as the value of the dealer card increases, the value of the states decreases. While the specific trend is covered by noise and hard to establish, it seems that there is a local maximum when player sum is equal to 11. This can be explained by noting that when the player sum is 11, the player cannot go bust in the next draw. This is further supported by the estimated optimal policy (also shown below).

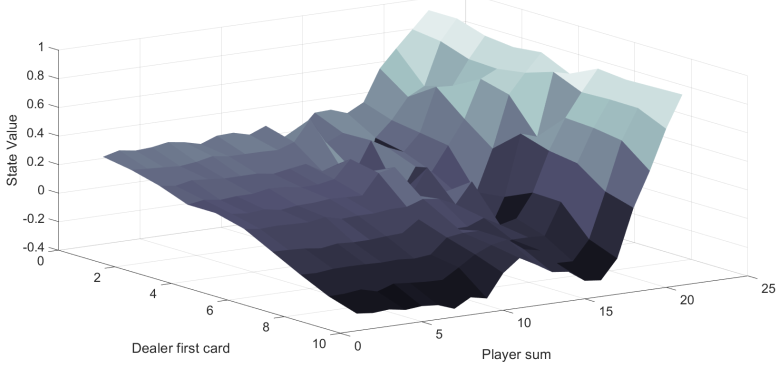

Figure 2: Optimal Value function estimation (N0=100)

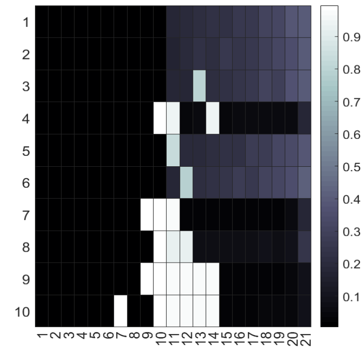

Figure 3: Optimal Policy for 'hit'

Question 2: TD Learning in Easy21

a. In this section, the SARSA on-policy learning TD control algorithm is used. This method offers lower variance than MC control. The State-Action Value function is updated online using the rule: $\hat{Q}(s,a) \leftarrow \hat{Q}(s,a) + \alpha[R(s,a) + \gamma\hat{Q}(s',a') - \hat{Q}(s,a)]$, where $(s',\ a')$ represent the next state-action pair. Again, $\alpha = 1/N(s,a)$ which satisfies Robbins-Monroe. It is expected that this decreasing implementation of α should perform better than constant step sizes. $\epsilon = \frac{N_0}{N_0+\sum_{a\in\mathcal{A}} N(s,a)}$, which results in an ϵ-greedy policy that is GLIE. N0=100 balances accuracy vs convergence speed (chosen empirically by plotting learning curves, Appendix A).

b. Learning curves are plotted using the same testing parameters as in Question 1b. To investigate the effects of α, plots were produced for different constant values of parameter α=0.001, 0.01, 0.1, 1 as well as $\alpha = 1/N(s,a)$ (three separate instances were run for each parameter value resulting in 15 plots). The plots below suggest that for small values of α, the process takes longer to converge and is relatively accurate ($\alpha = 0.001$ has very similar performance to $\alpha = 1/N(s,a)$), whereas higher values of α significantly under-perform. This suggests that while for constant α SARSA does not converge to the optimal state-value function (Robbins-Monroe is not satisfied), when α is sufficiently small the approximation can be very good.

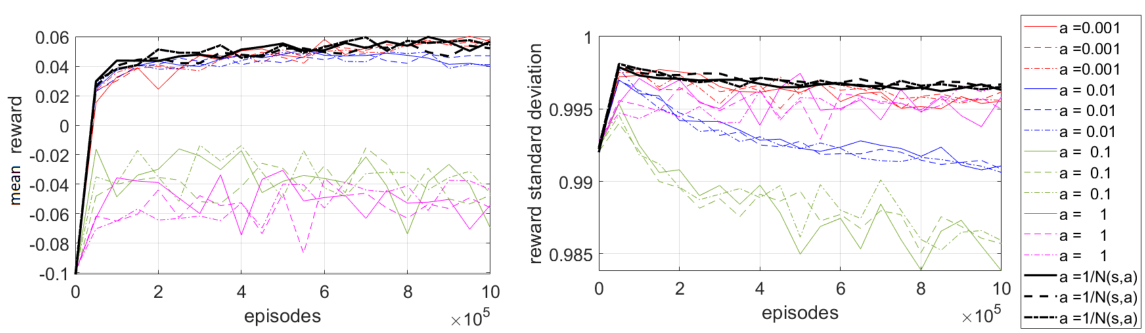

Figure 4: Learning Curves for a={0.001, 0.01, 0.1, 1, 1/N(s,a)}

c. The estimated optimal value function for SARSA with N0=100 after 10^6 episodes is plotted. The plot's characteristics are identical to those in MC control (Question 1c) apart from a slight increase in smoothness (can be attributed to the lower variance of SARSA vs MC).

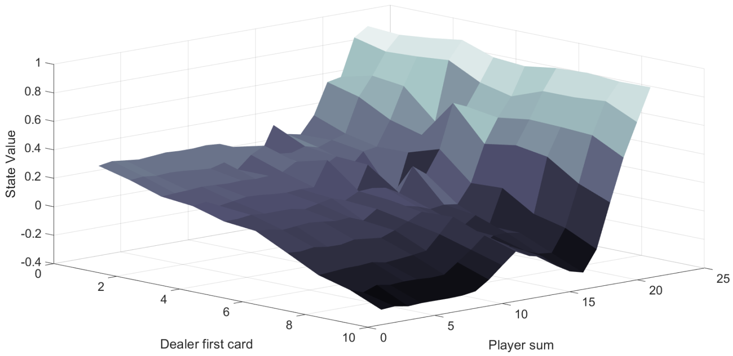

Figure 5: Optimal Value function estimation (N0=100)

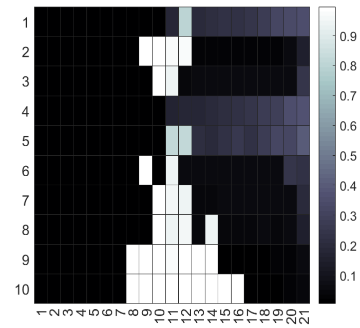

Figure 6: Optimal Policy for 'hit'

Question 3: Q-Learning in Easy21

a. In this section, Q-Learning, which is an Off-Policy TD Control algorithm, is implemented. Q-learning aims to improve two policies: the target policy whose value function is the target of the learning process, and the behaviour policy controlling the agent's actions. To achieve this the following update rule is used: $\hat{Q}(s,a) \leftarrow \hat{Q}(s,a) + \alpha[R(s,a) + \gamma \max_a \hat{Q}(s',a) - \hat{Q}(s,a)]$. Taking $\max_a \hat{Q}(s',a)$, means that the target policy is greedy w.r.t. $\hat{Q}(s,a)$ while, as in both previous methods, the behaviour policy is ϵ-greedy w.r.t. $\hat{Q}(s,a)$. $\alpha = 1/N(s,a)$ as in previous sections and satisfies Robbins-Monroe. $\epsilon = \frac{N_0}{N_0 + \sum_{a\in\mathcal{A}} N(s,a)}$, which results in an ϵ-greedy policy that is GLIE. N0=100 balances accuracy vs convergence speed (chosen empirically by plotting learning curves, Appendix A).

b. The learning curves are plotted using the same testing parameters as in Questions 1b, 2b. To investigate the effect of different strategies for handling ϵ-greediness, plots were produced for different constant values of parameter ϵ=0.001, 0.01, 0.1, 1 as well as $\epsilon = \frac{100}{100 + \sum_{a\in\mathcal{A}} N(s,a)}$ (three separate instances were run for each parameter value resulting in 15 plots). The plots in the figure below suggest that when ϵ is equal to a very small constant value, not enough exploration will be performed resulting in poor learning ($\epsilon = 0.001$). Conversely, when ϵ is constant and high, while exploration is achieved, the optimal policy estimate is far enough from convergence that the results are poor ($\epsilon = 0.5$). Finally, for ϵ within a range of appropriate values, the learning curves are close (but slightly worse) than those for the time-varying ϵ. It is also important to note that any policy with constant ϵ is not GLIE.

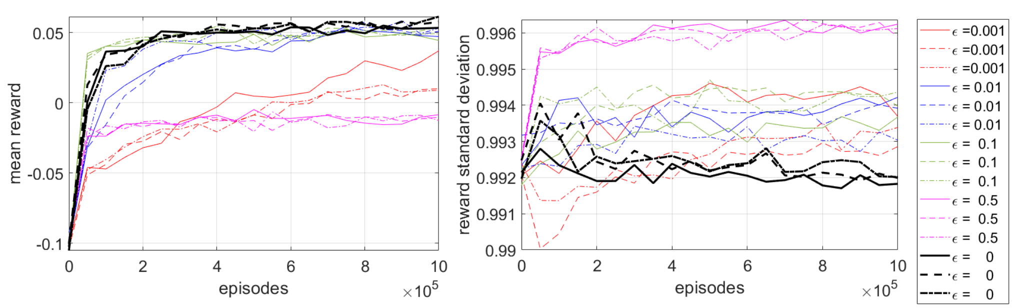

Figure 7: Learning Curves for ϵ={0.001, 0.01, 0.1, 0.5, $\frac{100}{100 + \sum_{a\in\mathcal{A}} N(s,a)}$}

c. The estimated optimal value function for Q-learning with $\epsilon = \frac{N_0}{N_0 + \sum_{a \in \mathcal{A}} N(s,a)}$ and N0=100 after 10^6 episodes is plotted. While the plot has the same shape as those in MC control and SARSA (Question 1c, 2c), it is by far the smoothest. This implies that the variance is significantly lower than in both other methods.

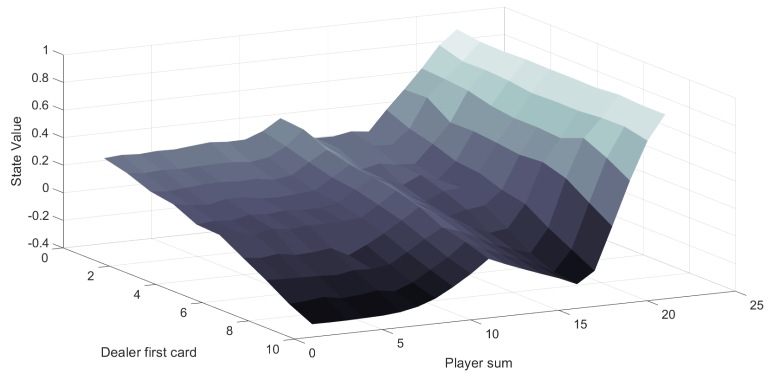

Figure 8: Optimal Value function

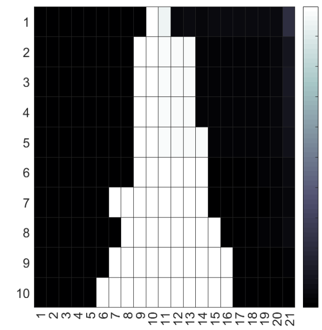

Figure 9: Optimal Policy for 'hit'

Question 4: Compare the algorithms

To compare the algorithms, learning curves are computed for all three methods. The number of testing simulations has been increased to $25{\cdot}10^4$ and testing occurs every 10^4 episodes. All algorithms use time varying $\alpha = 1/N(s,a)$ and $\epsilon = \frac{N_0}{N_0 + \sum_{a \in \mathcal{A}} N(s,a)}$ with N0=100. Note that each algorithm has been run three times, producing nine plots.

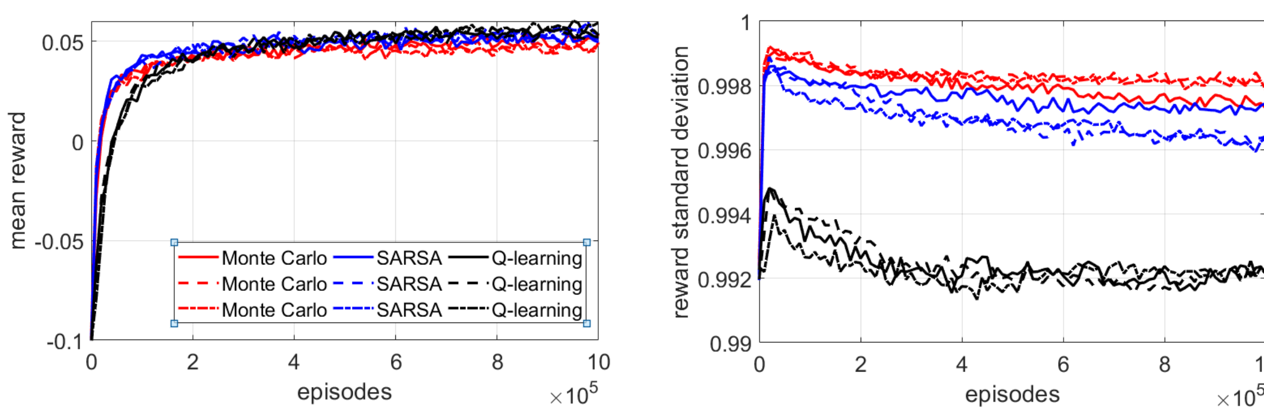

Figure 10: Learning Curves the three methods used

From the above figures, it can be seen that after 10^6 episodes, the mean reward for Q-learning is larger than that for SARSA which is larger than that for Monte Carlo. Q-learning producing better results than SARSA can be explained by noting that Q-learning converges closer to the optimal policy, while SARSA converges to a safer policy. Since the penalties aren't large, the reward from following a riskier but more accurate policy will be higher than the reward from following a safe policy. Monte Carlo has high variance and because it is implemented iteratively, it also accumulates error. Thus, it falls below SARSA. It is expected that batch MC would perform better but would be very computationally intensive. The standard deviation of rewards is linked to the variance of the estimator of the optimal value function, thus MC which has the highest estimator variance will also have the highest standard deviation of rewards. Finally, it is worth noting that SARSA and Q-learning exploit the Markov property whereas MC does not, which could explain why the former perform better.

Appendix A

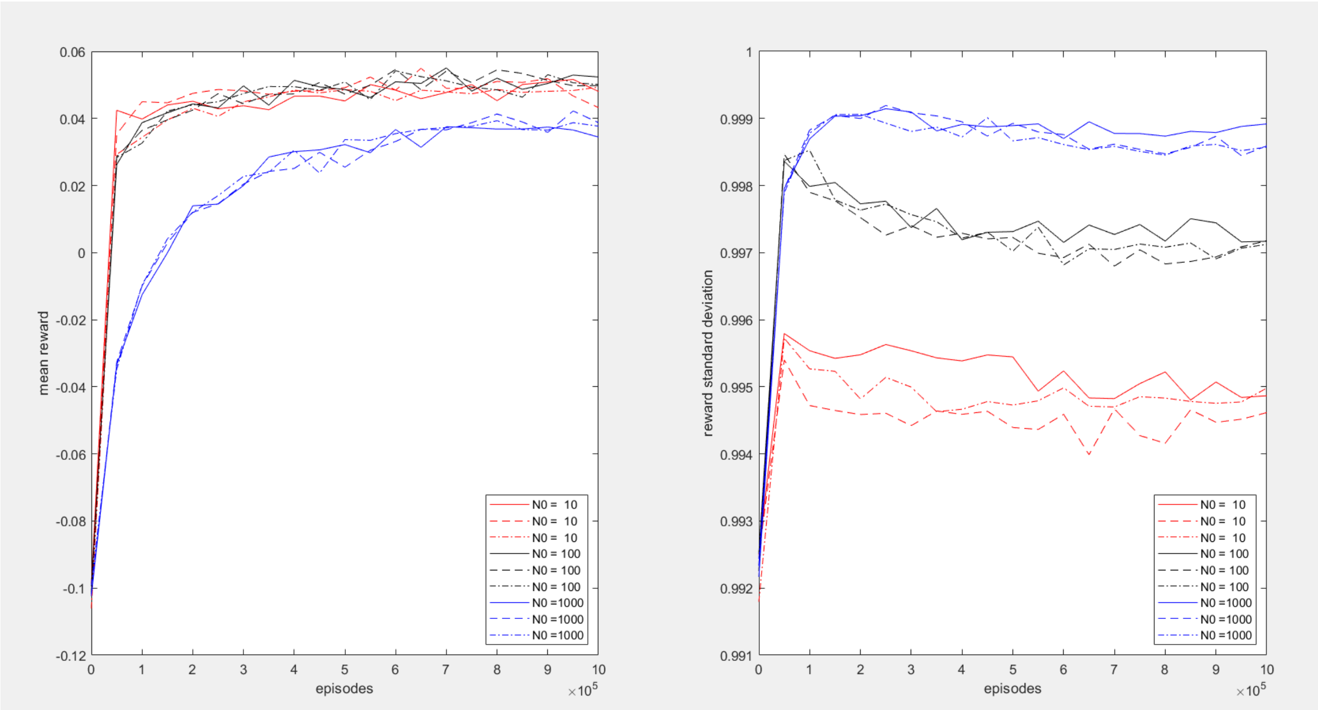

Figure 11: Learning Curves for SARSA with varying N0. N0=100 offers best performance

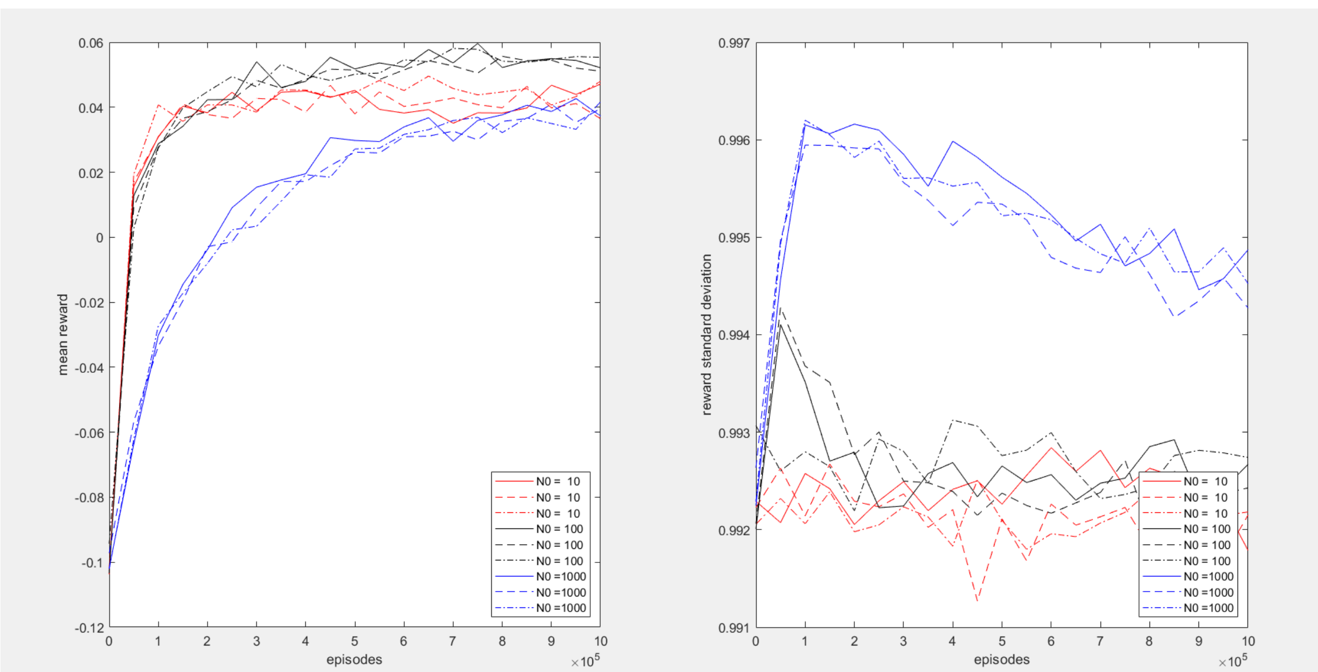

Figure 12: Learning Curves for Q-learning with varying N0. N0=100 offers best performance

Appendix B: MATLAB Code

Note that while the algorithms have been coded quite efficiently, the plotting section is not implemented elegantly and some cosmetic editing of the plots (line-widths, colours etc) has been done through the MATLAB GUI and not through coding.

```
%STEP FUNCTION
function [s_n, r, fin]=step(s, a)
ds=s(1);
ps=s(2);
fin=0;
if a==1
    val=randi(10);
    mult=2*(rand()>(1/3))-1; % 2/3 chance 1, 1/3 chance -1
    ps=ps+mult*val;
    if ps<1 || ps>21
        r=-1;
        fin=1;
    else
        r=0;
    end
else
    fin=1;
    while ds<17 && ds>=1
        val=randi(10);
        mult=2*(rand()>(1/3))-1;
        ds=ds+mult*val;
    end
    if ds<1||ds>21||ds<ps
        r=1;
    elseif ds==ps
        r=0;
    else
        r=-1;
    end
end
s_n=[ds ps];
end

%MONTE CARLO fUNCTION
function [R,R_test,Q,pol,count_state_action]=mc(n_test,n_epi,d_test,N0,
    ev,dshow)
ind_show=1;
ind_test_vec=[1 d_test:d_test:n_epi];
ind_test=1;
%
t0=tic;%timer
pol=0.5*ones(10,21,2); % pi('hit',s)
```

```
count_state_action=zeros(10,21,2);
Q=zeros(10,21,2);
R=zeros(n_test,length(ind_test_vec));
R_test=zeros(1,n_epi);
for epi=1:n_epi
    fin=0;
    s=randi(10,[1 2]);
    trace=[];
    while fin==0
        temp=rand();
        a=2-(temp<pol(s(1),s(2)));
        [sn,r,fin]=step(s,a);
        trace=[trace;[s a]];
        s=sn;
    end
    R_test(epi)=r;
    checked=zeros(10,21,2);
    for ind=1:size(trace,1)
        if ev || ~checked(trace(ind,1),trace(ind,2),trace(ind,3))
            count_state_action(trace(ind,1),trace(ind,2),trace(ind,3))=count_state_action(trace(ind,1),trace(ind,2),trace(ind,3))+1;
            Q_hat=Q(trace(ind,1),trace(ind,2),trace(ind,3));
            Q_hat=Q_hat+(r-Q_hat)/count_state_action(trace(ind,1),trace(ind,2),trace(ind,3));
            Q(trace(ind,1),trace(ind,2),trace(ind,3))=Q_hat;
            checked(trace(ind,1),trace(ind,2),trace(ind,3))=1;
        end
    end
    count_state=count_state_action(:,:,1)+count_state_action(:,:,2);
    checked=zeros(10,21);
    for ind=1:size(trace,1)
        if ev || ~checked(trace(ind,1),trace(ind,2))
            eps=N0/(N0+count_state(trace(ind,1),trace(ind,2)));
            if Q(trace(ind,1),trace(ind,2),1)>Q(trace(ind,1),trace(ind,2),2)
                pol(trace(ind,1),trace(ind,2),1)=1-eps/2;
                pol(trace(ind,1),trace(ind,2),2)=eps/2;
            end
            if Q(trace(ind,1),trace(ind,2),1)<Q(trace(ind,1),trace(ind,2),2)
                pol(trace(ind,1),trace(ind,2),2)=1-eps/2;
                pol(trace(ind,1),trace(ind,2),1)=eps/2;
            end
            if Q(trace(ind,1),trace(ind,2),1)==Q(trace(ind,1),trace(ind,2),2)
                pol(trace(ind,1),trace(ind,2),1)=0.5;
```

```
                    pol(trace(ind,1),trace(ind,2),2)=0.5;
                end
                checked(trace(ind,1),trace(ind,2))=1;
            end
        end
        if epi==ind_test_vec(ind_test)
            for train_epi=1:n_test
                fin=0;
                s=randi(10,[1 2]);
                trace=[];
                while fin==0
                    temp=rand();
                    a=2-(temp<pol(s(1),s(2))); % is 1 for 'hit' 2 for 'stick'
                    [sn,r,fin]=step(s,a);
                    s=sn;
                end
                R(train_epi,ind_test)=r;
            end
            ind_test=ind_test+1;
        end
        if (epi/n_epi)/dshow>ind_show
            ind_show=ind_show+1;
            [round(100*(epi/n_epi)) toc(t0)]
            t0=tic;
        end
end
end
%SARSA FUNCTION
function [R,R_train,Q,pol,count_state_action]=fsarsa(n_train,n_epi,d_test,N0,dshow,a_choice)
% counter stuff for visulaizing progress
ind_show=1;
ind_test_vec=[1 d_test:d_test:n_epi];
ind_test=1;
%
t0=tic;%timer
pol=0.5*ones(10,21,2); % pi('hit',s)
count_state_action=zeros(10,21,2);
Q=zeros(10,21,2);     %dim 1 is 'hit dim 2 'stick'
R_train=zeros(1,n_epi);
R=zeros(n_train,length(ind_test_vec));
for epi=1:n_epi
    fin=0;
    s=randi(10,[1 2]);
    temp=rand();
    A=2-(temp<pol(s(1),s(2))); % is 1 for 'hit' 2 for 'stick'
```

```
    while fin==0
        [sn,r,fin]=step(s,A);
        temp=rand();
        count_state_action(s(1),s(2),A)=count_state_action(s(1),s(2),A
            )+1;
        if a_choice==0
            a=1/count_state_action(s(1),s(2),A);
        else
            a=a_choice;
        end
        if fin==0
            An=2-(temp<pol(sn(1),sn(2))); % is 1 for 'hit' 2 for '
                stick'
            Q(s(1),s(2),A)=Q(s(1),s(2),A)+a*(r+Q(sn(1),sn(2),An)-Q(s
                (1),s(2),A));
        else
            Q(s(1),s(2),A)=Q(s(1),s(2),A)+a*(r-Q(s(1),s(2),A));
        end
        eps=N0/(N0+count_state_action(s(1),s(2),1)+count_state_action(
            s(1),s(2),2));
        if Q(s(1),s(2),1)>Q(s(1),s(2),2)
            pol(s(1),s(2),1)=1-eps/2;
            pol(s(1),s(2),2)=eps/2;
        end
        if Q(s(1),s(2),1)<Q(s(1),s(2),2)
            pol(s(1),s(2),2)=1-eps/2;
            pol(s(1),s(2),1)=eps/2;
        end
        if Q(s(1),s(2),1)==Q(s(1),s(2),2)
            pol(s(1),s(2),1)=0.5;
            pol(s(1),s(2),2)=0.5;
        end
        if fin==0
            A=An;
            s=sn;
        end
    end
    R_train(epi)=r;
    if epi==ind_test_vec(ind_test)
        for train_epi=1:n_train
            fin=0;
            s=randi(10,[1 2]);
            while fin==0
                temp=rand();
                a=2-(temp<pol(s(1),s(2))); % is 1 for 'hit' 2 for '
                    stick'
                [sn,r,fin]=step(s,a);
```

```
                    s=sn;
                end
                R(train_epi,ind_test)=r;
            end
            ind_test=ind_test+1;
        end
        %timer/counter for progress monitoring
        if (epi/n_epi)/dshow>ind_show
            ind_show=ind_show+1;
            [round(100*(epi/n_epi)) toc(t0)]
            t0=tic;
        end
end
end
%Q LEARN FUNCTION
function [R,R_train,Q,pol,count_state_action]=Q_learn(n_train,n_epi,
    d_test,N0,dshow,eps_choice)
% counter stuff for visulaizing progress
ind_show=1;
ind_test_vec=[1 d_test:d_test:n_epi];
ind_test=1;
%
t0=tic;%timer
pol=0.5*ones(10,21,2); % pi('hit',s)
count_state_action=zeros(10,21,2);
Q=zeros(10,21,2);     %dim 1 is 'hit dim 2 'stick'
R_train=zeros(1,n_epi);
R=zeros(n_train,length(ind_test_vec));
for epi=1:n_epi
    fin=0;
    s=randi(10,[1 2]);
    temp=rand();
    A=2-(temp<pol(s(1),s(2))); % is 1 for 'hit' 2 for 'stick'
    while fin==0
        [sn,r,fin]=step(s,A);
        temp=rand();
        count_state_action(s(1),s(2),A)=count_state_action(s(1),s(2),A
            )+1;
        a=1/count_state_action(s(1),s(2),A);
        if fin==0
            An=2-(temp<pol(sn(1),sn(2))); % is 1 for 'hit' 2 for '
                stick'
            Q(s(1),s(2),A)=Q(s(1),s(2),A)+a*(r+max(Q(sn(1),sn(2),1),Q(
                sn(1),sn(2),2))-Q(s(1),s(2),A));
        else
            Q(s(1),s(2),A)=Q(s(1),s(2),A)+a*(r-Q(s(1),s(2),A));
        end
```

```
            if eps_choice==0
                eps=N0/(N0+count_state_action(s(1),s(2),1)+count_state_action(s(1),s(2),2));
            else
                eps=eps_choice;
            end
            if Q(s(1),s(2),1)>Q(s(1),s(2),2)
                pol(s(1),s(2),1)=1-eps/2;
                pol(s(1),s(2),2)=eps/2;
            end
            if Q(s(1),s(2),1)<Q(s(1),s(2),2)
                pol(s(1),s(2),2)=1-eps/2;
                pol(s(1),s(2),1)=eps/2;
            end
            if Q(s(1),s(2),1)==Q(s(1),s(2),2)
                pol(s(1),s(2),1)=0.5;
                pol(s(1),s(2),2)=0.5;
            end
            if fin==0
                A=An;
                s=sn;
            end
        end
        R_train(epi)=r;
        if epi==ind_test_vec(ind_test)
            for train_epi=1:n_train
                fin=0;
                s=randi(10,[1 2]);
                trace=[];
                while fin==0
                    temp=rand();
                    a=2-(temp<pol(s(1),s(2))); % is 1 for 'hit' 2 for 'stick'
                    [sn,r,fin]=step(s,a);
                    s=sn;
                end
                R(train_epi,ind_test)=r;
            end
            ind_test=ind_test+1;
        end
        %timer/counter for progress monitoring
        if (epi/n_epi)/dshow>ind_show
            ind_show=ind_show+1;
            [round(100*(epi/n_epi)) toc(t0)]
            t0=tic;
        end
end
```

```
end

%PLOTTING

clc;clear;close all
%%
n_test=16*10^4;
n_epi=10^6;
dshow=0.1;
d_test=5*10^4;
figure(1)
N0_vec=[10 100 1000];
c_vec=['r';'k';'b'];
for ind=1:length(N0_vec)
    N0=N0_vec(ind);
    [R,R_train,Q,pol,count_state_action]=mc(n_test,n_epi,d_test,N0,0,dshow);
    subplot(1,2,1)
    plot([1 d_test:d_test:n_epi],mean(R),'color',c_vec(ind))
    hold on
    subplot(1,2,2)
    plot([1 d_test:d_test:n_epi],std(R),'color',c_vec(ind))
    hold on
    [R,R_train,Q,pol,count_state_action]=mc(n_test,n_epi,d_test,N0,0,dshow);
    subplot(1,2,1)
    plot([1 d_test:d_test:n_epi],mean(R),'color',c_vec(ind),'Linestyle','--')
    hold on
    subplot(1,2,2)
    plot([1 d_test:d_test:n_epi],std(R),'color',c_vec(ind),'Linestyle','--')
    hold on
    [R,R_train,Q,pol,count_state_action]=mc(n_test,n_epi,d_test,N0,0,dshow);
    subplot(1,2,1)
    plot([1 d_test:d_test:n_epi],mean(R),'color',c_vec(ind),'Linestyle','-.')
    hold on
    subplot(1,2,2)
    plot([1 d_test:d_test:n_epi],std(R),'color',c_vec(ind),'Linestyle','-.')
    hold on
end
subplot(1,2,1)
legend(strcat('N0 = ',num2str([10 10 10 100 100 100 1000 1000 1000]')),'Location','southeast')
```

```
xlabel('episodes')
ylabel('mean reward')
subplot(1,2,2)
legend(strcat('N0 = ',num2str([10 10 10 100 100 100 1000 1000 1000]'))
    ,'Location','southeast')
xlabel('episodes')
ylabel('reward standard deviation')
%%
n_test=1;
n_epi=10^6;
N0=100;
[R,R_train,Q,pol,count_state_action]=mc(n_test,n_epi,d_test,N0,0,dshow
    );
[X,Y] = meshgrid(1:10,1:21);
v=max(Q,[],3);
figure(2)
colormap('bone')
surf(X,Y,v','EdgeColor','none');
xlabel('Dealer first card')
ylabel('Player sum')
zlabel('State Value')
figure(3)
h=heatmap(pol(:,:,1))
h.Colormap=colormap('bone');
%%
n_test=16*10^4;
n_epi=10^6;
dshow=0.1;
d_test=5*10^4;
figure(4)
N0_vec=[10 100 1000];
c_vec=['r';'k';'b'];
for ind=1:length(N0_vec)
    N0=N0_vec(ind);
    [R,R_train,Q,pol,count_state_action]=fsarsa(n_test,n_epi,d_test,N0
        ,dshow,0);
    subplot(1,2,1)
    plot([1 d_test:d_test:n_epi],mean(R),'color',c_vec(ind))
    hold on
    subplot(1,2,2)
    plot([1 d_test:d_test:n_epi],std(R),'color',c_vec(ind))
    hold on
    [R,R_train,Q,pol,count_state_action]=fsarsa(n_test,n_epi,d_test,N0
        ,dshow,0);
    subplot(1,2,1)
    plot([1 d_test:d_test:n_epi],mean(R),'color',c_vec(ind),'Linestyle
        ','--')
```

```
    hold on
    subplot(1,2,2)
    plot([1 d_test:d_test:n_epi],std(R),'color',c_vec(ind),'Linestyle','--')
    hold on
    [R,R_train,Q,pol,count_state_action]=fsarsa(n_test,n_epi,d_test,N0,dshow,0);
    subplot(1,2,1)
    plot([1 d_test:d_test:n_epi],mean(R),'color',c_vec(ind),'Linestyle','-.')
    hold on
    subplot(1,2,2)
    plot([1 d_test:d_test:n_epi],std(R),'color',c_vec(ind),'Linestyle','-.')
    hold on
end
subplot(1,2,1)
legend(strcat('N0 = ',num2str([10 10 10 100 100 100 1000 1000 1000]')),'Location','southeast')
xlabel('episodes')
ylabel('mean reward')
subplot(1,2,2)
legend(strcat('N0 = ',num2str([10 10 10 100 100 100 1000 1000 1000]')),'Location','southeast')
xlabel('episodes')
ylabel('reward standard deviation')
%%
n_test=16*10^4;
n_epi=10^6;
dshow=0.1;
d_test=5*10^4;
figure(5)
a_vec=[0.001 0.01 0.1 1 0];
c_vec=['r';'b';'g';'m';'k'];
N0=100;
for ind=1:length(a_vec)
    a=a_vec(ind);
    [R,R_train,Q,pol,count_state_action]=fsarsa(n_test,n_epi,d_test,N0,dshow,a);
    subplot(1,2,1)
    plot([1 d_test:d_test:n_epi],mean(R),'color',c_vec(ind))
    hold on
    subplot(1,2,2)
    plot([1 d_test:d_test:n_epi],std(R),'color',c_vec(ind))
    hold on
    [R,R_train,Q,pol,count_state_action]=fsarsa(n_test,n_epi,d_test,N0,dshow,a);
```

```
    subplot(1,2,1)
    plot([1 d_test:d_test:n_epi],mean(R),'color',c_vec(ind),'Linestyle','--')
    hold on
    subplot(1,2,2)
    plot([1 d_test:d_test:n_epi],std(R),'color',c_vec(ind),'Linestyle','--')
    hold on
    [R,R_train,Q,pol,count_state_action]=fsarsa(n_test,n_epi,d_test,N0,dshow,a);
    subplot(1,2,1)
    plot([1 d_test:d_test:n_epi],mean(R),'color',c_vec(ind),'Linestyle','-.')
    hold on
    subplot(1,2,2)
    plot([1 d_test:d_test:n_epi],std(R),'color',c_vec(ind),'Linestyle','-.')
    hold on
end
subplot(1,2,1)
legend(strcat('a = ',num2str([0.001 0.001 0.001 0.01 0.01 0.01 0.1 0.1 0.1 1 1 1 0 0 0]')),'Location','southeast')
xlabel('episodes')
ylabel('mean reward')
subplot(1,2,2)
legend(strcat('a = ',num2str([0.001 0.001 0.001 0.01 0.01 0.01 0.1 0.1 0.1 1 1 1 0 0 0]')),'Location','southeast')
xlabel('episodes')
ylabel('reward standard deviation')
%%
n_test=1;
n_epi=10^6;
N0=100;
[R,R_train,Q,pol,count_state_action]=fsarsa(n_test,n_epi,d_test,N0,dshow,0);
[X,Y] = meshgrid(1:10,1:21);
v=max(Q,[],3);
figure(6)
colormap('bone')
surf(X,Y,v','EdgeColor','none');
xlabel('Dealer first card')
ylabel('Player sum')
zlabel('State Value')
figure(7)
h=heatmap(pol(:,:,1))
h.Colormap=colormap('bone');
%%
```

```
n_test=16*10^4;
n_epi=10^6;
dshow=0.1;
d_test=5*10^4;
figure(8)
N0_vec=[10 100 1000];
c_vec=['r';'k';'b'];
for ind=1:length(N0_vec)
    N0=N0_vec(ind);
    [R,R_train,Q,pol,count_state_action]=Q_learn(n_test,n_epi,d_test,
        N0,dshow,0);
    subplot(1,2,1)
    plot([1 d_test:d_test:n_epi],mean(R),'color',c_vec(ind))
    hold on
    subplot(1,2,2)
    plot([1 d_test:d_test:n_epi],std(R),'color',c_vec(ind))
    hold on
    [R,R_train,Q,pol,count_state_action]=Q_learn(n_test,n_epi,d_test,
        N0,dshow,0);
    subplot(1,2,1)
    plot([1 d_test:d_test:n_epi],mean(R),'color',c_vec(ind),'Linestyle
        ','--')
    hold on
    subplot(1,2,2)
    plot([1 d_test:d_test:n_epi],std(R),'color',c_vec(ind),'Linestyle'
        ,'--')
    hold on
    [R,R_train,Q,pol,count_state_action]=Q_learn(n_test,n_epi,d_test,
        N0,dshow,0);
    subplot(1,2,1)
    plot([1 d_test:d_test:n_epi],mean(R),'color',c_vec(ind),'Linestyle
        ','-.')
    hold on
    subplot(1,2,2)
    plot([1 d_test:d_test:n_epi],std(R),'color',c_vec(ind),'Linestyle'
        ,'-.')
    hold on
end
subplot(1,2,1)
legend(strcat('N0 = ',num2str([10 10 10 100 100 100 1000 1000 1000]'))
    ,'Location','southeast')
xlabel('episodes')
ylabel('mean reward')
subplot(1,2,2)
legend(strcat('N0 = ',num2str([10 10 10 100 100 100 1000 1000 1000]'))
    ,'Location','southeast')
xlabel('episodes')
```

```
ylabel('reward standard deviation')
%%
n_test=16*10^4;
n_epi=10^6;
dshow=0.1;
d_test=5*10^4;
figure(9);close;figure(9)
e_vec=[0.001 0.01 0.1 0.5 0];
c_vec=['r';'b';'g';'m';'k'];
N0=100;
for ind=1:length(e_vec)
    e=e_vec(ind);
    [R,R_train,Q,pol,count_state_action]=Q_learn(n_test,n_epi,d_test,N0,dshow,e);
    subplot(1,2,1)
    plot([1 d_test:d_test:n_epi],mean(R),'color',c_vec(ind))
    hold on
    subplot(1,2,2)
    plot([1 d_test:d_test:n_epi],std(R),'color',c_vec(ind))
    hold on
    [R,R_train,Q,pol,count_state_action]=Q_learn(n_test,n_epi,d_test,N0,dshow,e);
    subplot(1,2,1)
    plot([1 d_test:d_test:n_epi],mean(R),'color',c_vec(ind),'Linestyle','--')
    hold on
    subplot(1,2,2)
    plot([1 d_test:d_test:n_epi],std(R),'color',c_vec(ind),'Linestyle','--')
    hold on
    [R,R_train,Q,pol,count_state_action]=Q_learn(n_test,n_epi,d_test,N0,dshow,e);
    subplot(1,2,1)
    plot([1 d_test:d_test:n_epi],mean(R),'color',c_vec(ind),'Linestyle','-.')
    hold on
    subplot(1,2,2)
    plot([1 d_test:d_test:n_epi],std(R),'color',c_vec(ind),'Linestyle','-.')
    hold on
end
subplot(1,2,1)
legend(strcat('\epsilon = ',num2str([0.001 0.001 0.001 0.01 0.01 0.01 0.1 0.1 0.1 0.5 0.5 0.5 0 0 0]')),'Location','southeast')
xlabel('episodes')
ylabel('mean reward')
subplot(1,2,2)
```

```
legend(strcat('\epsilon = ',num2str([0.001 0.001 0.001 0.01 0.01 0.01 0.1 0.1 0.1 0.5 0.5 0.5 0 0 0]')),'Location','southeast')
xlabel('episodes')
ylabel('reward standard deviation')
%%
n_test=1;
n_epi=10^6;
N0=100;
[R,R_train,Q,pol,count_state_action]=Q_learn(n_test,n_epi,d_test,N0,dshow,0);
[X,Y] = meshgrid(1:10,1:21);
v=max(Q,[],3);
figure(10)
colormap('bone')
surf(X,Y,v','EdgeColor','none');
xlabel('Dealer first card')
ylabel('Player sum')
zlabel('State Value')
figure(11)
h=heatmap(pol(:,:,1))
h.Colormap=colormap('bone');
%%
n_test=25*10^4;
n_epi=10^6;
dshow=0.95;
d_test=10^4;
figure(12);close;figure(12)
N0=100;
%MC-RED
[R,R_train,Q,pol,count_state_action]=mc(n_test,n_epi,d_test,N0,0,dshow);
subplot(1,2,1)
plot([1 d_test:d_test:n_epi],mean(R),'color','r','Linestyle','-')
hold on
subplot(1,2,2)
plot([1 d_test:d_test:n_epi],std(R),'color','r','Linestyle','-')
hold on
[R,R_train,Q,pol,count_state_action]=mc(n_test,n_epi,d_test,N0,0,dshow);
subplot(1,2,1)
plot([1 d_test:d_test:n_epi],mean(R),'color','r','Linestyle','--')
hold on
subplot(1,2,2)
plot([1 d_test:d_test:n_epi],std(R),'color','r','Linestyle','--')
hold on
[R,R_train,Q,pol,count_state_action]=mc(n_test,n_epi,d_test,N0,0,dshow);
```

```
subplot(1,2,1)
plot([1 d_test:d_test:n_epi],mean(R),'color','r','Linestyle','-.')
hold on
subplot(1,2,2)
plot([1 d_test:d_test:n_epi],std(R),'color','r','Linestyle','-.')
hold on
%FSARSA-BLUE
[R,R_train,Q,pol,count_state_action]=fsarsa(n_test,n_epi,d_test,N0,dshow,0);
subplot(1,2,1)
plot([1 d_test:d_test:n_epi],mean(R),'color','b','Linestyle','-')
hold on
subplot(1,2,2)
plot([1 d_test:d_test:n_epi],std(R),'color','b','Linestyle','-')
hold on
[R,R_train,Q,pol,count_state_action]=fsarsa(n_test,n_epi,d_test,N0,dshow,0);
subplot(1,2,1)
plot([1 d_test:d_test:n_epi],mean(R),'color','b','Linestyle','--')
hold on
subplot(1,2,2)
plot([1 d_test:d_test:n_epi],std(R),'color','b','Linestyle','--')
hold on
[R,R_train,Q,pol,count_state_action]=fsarsa(n_test,n_epi,d_test,N0,dshow,0);
subplot(1,2,1)
plot([1 d_test:d_test:n_epi],mean(R),'color','b','Linestyle','-.')
hold on
subplot(1,2,2)
plot([1 d_test:d_test:n_epi],std(R),'color','b','Linestyle','-.')
hold on
%Q-BLACK
[R,R_train,Q,pol,count_state_action]=Q_learn(n_test,n_epi,d_test,N0,dshow,0);
subplot(1,2,1)
plot([1 d_test:d_test:n_epi],mean(R),'color','k','Linestyle','-')
hold on
subplot(1,2,2)
plot([1 d_test:d_test:n_epi],std(R),'color','k','Linestyle','-')
hold on
[R,R_train,Q,pol,count_state_action]=Q_learn(n_test,n_epi,d_test,N0,dshow,0);
subplot(1,2,1)
plot([1 d_test:d_test:n_epi],mean(R),'color','k','Linestyle','--')
hold on
subplot(1,2,2)
plot([1 d_test:d_test:n_epi],std(R),'color','k','Linestyle','--')
```

```
hold on
[R,R_train,Q,pol,count_state_action]=Q_learn(n_test,n_epi,d_test,N0,
    dshow,0);
subplot(1,2,1)
plot([1 d_test:d_test:n_epi],mean(R),'color','k','Linestyle','-.')
hold on
subplot(1,2,2)
plot([1 d_test:d_test:n_epi],std(R),'color','k','Linestyle','-.')
hold on
subplot(1,2,1)
xlabel('episodes')
ylabel('mean reward')
subplot(1,2,2)
legend([{'Monte Carlo'},{'Monte Carlo'},{'Monte Carlo'},{'SARSA'},{'
    SARSA'},{'SARSA'},{'Q-learning'},{'Q-learning'},{'Q-learning'}])
xlabel('episodes')
ylabel('reward standard deviation')
```