

## Question 1: Monte-Carlo Control in Easy21

- a. To implement Monte-Carlo control in Easy21, an iterative on-policy  $\epsilon$ -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<sup>1</sup>. 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  $\tau$  has been generated. For all unique state-action pairs appearing in  $\tau$ :  $\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) + \alpha[R(s, a) - \hat{Q}(s, a)]$  where  $R(s, a)$  is the return following the first visit to the state-action pair  $(s, a)$  (note that since all non-zero rewards occur at the end of the episode and  $\gamma = 1$ ,  $R(s, a) = r_{\text{episode}}$  for all  $(s, a)$  in  $\tau$ ). The parameter  $\alpha$  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  $\hat{Q}(s, \text{'hit'})$  is equal to  $\hat{Q}(s, \text{'stick'})$  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  $\epsilon$ -greedy policy that is GLIE. The parameter  $N_0$  is used to balance exploration vs exploitation for the number of episodes to be run (given time and computational constraints). As  $N_0$  increases, the exploration increases but so does the time required for convergence. For smaller values of  $N_0$ , convergence is quicker but the results likely inaccurate. The value of  $N_0$  will be determined by inspecting learning curves for different  $N_0$  (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  $N_0 = 10, 100$  and  $1000$ .

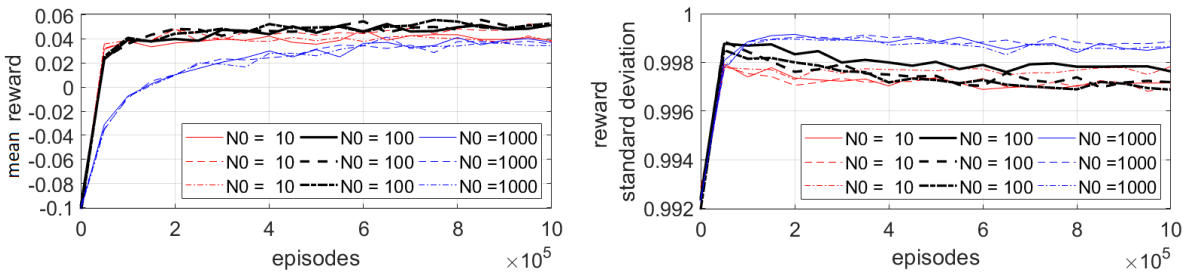


Figure 1: Learning Curves for  $N_0=\{10,100,1000\}$ .  $N_0=100$  offers the best overall performance.  $N_0=10$  converges faster but 2 out of 3 times converges to a worse policy than  $N_0=100$ . When  $N_0=1000$ , convergence has not occurred yet after  $10^6$  episodes.

<sup>1</sup>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).

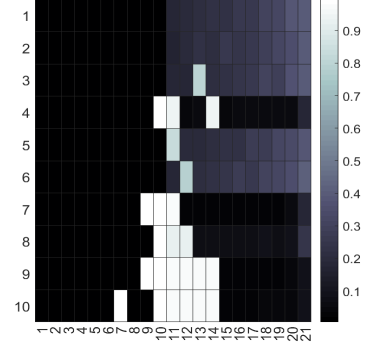
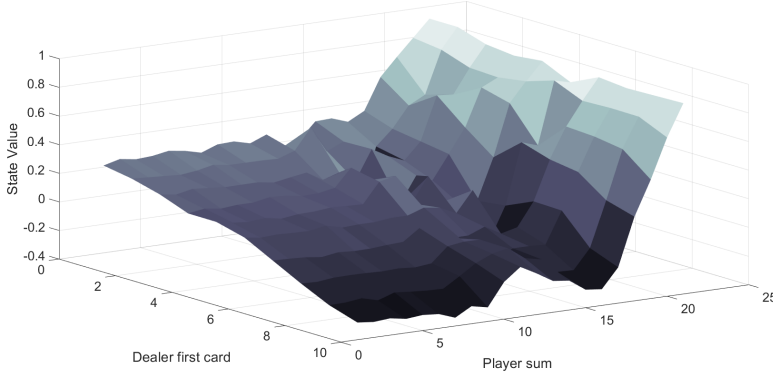
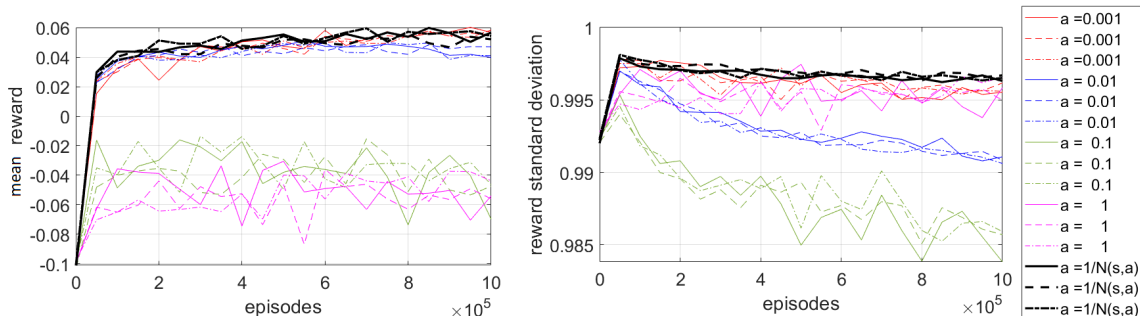
Figure 2: Optimal Value function estimation ( $N_0=100$ )

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  $\alpha$  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  $\epsilon$ -greedy policy that is GLIE.  $N_0=100$  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  $\alpha$ , plots were produced for different constant values of parameter  $\alpha=0.001, 0.01, 0.1, 1$  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  $\alpha$ , 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  $\alpha$  significantly under-perform. This suggests that while for constant  $\alpha$  SARSA does not converge to the optimal state-value function (Robbins-Monroe is not satisfied), when  $\alpha$  is sufficiently small the approximation can be very good.

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

- c. The estimated optimal value function for SARSA with  $N_0=100$  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).

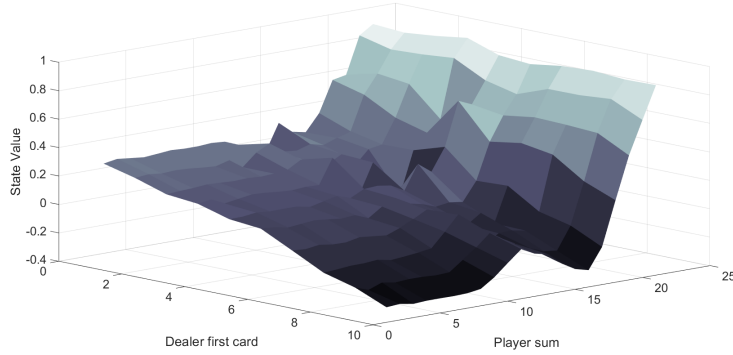
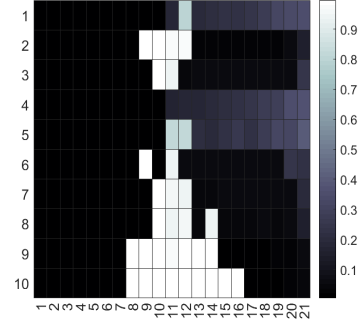
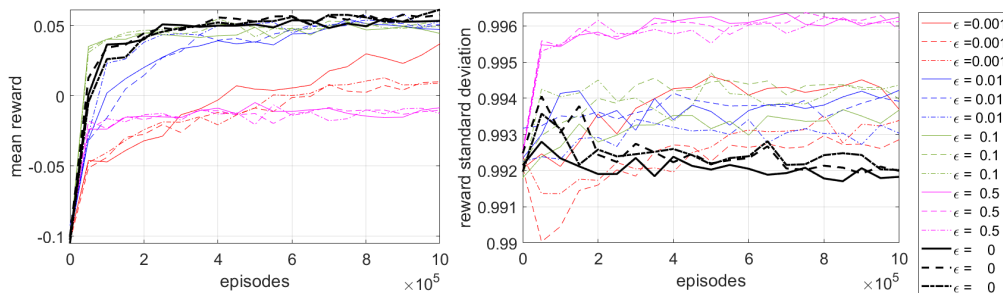
Figure 5: Optimal Value function estimation ( $N_0=100$ )

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  $\epsilon$ -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  $\epsilon$ -greedy policy that is GLIE.  $N_0=100$  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  $\epsilon$ -greediness, plots were produced for different constant values of parameter  $\epsilon=0.001, 0.01, 0.1, 1$  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  $\epsilon$  is equal to a very small constant value, not enough exploration will be performed resulting in poor learning ( $\epsilon = 0.001$ ). Conversely, when  $\epsilon$  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  $\epsilon$  within a range of appropriate values, the learning curves are close (but slightly worse) than those for the time-varying  $\epsilon$ . It is also important to note that any policy with constant  $\epsilon$  is not GLIE.

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

- 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  $N_0=100$  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.

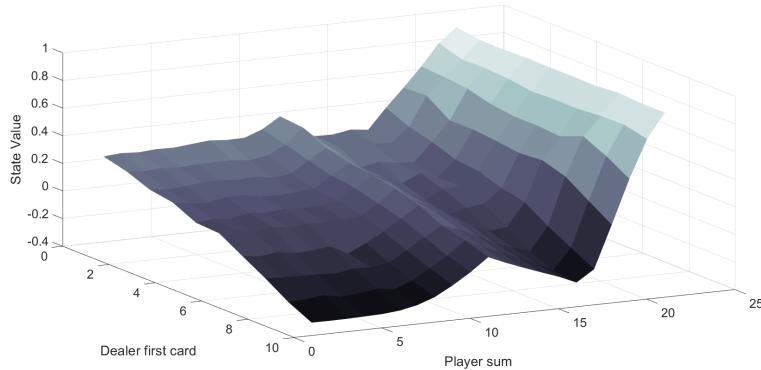


Figure 8: Optimal Value function

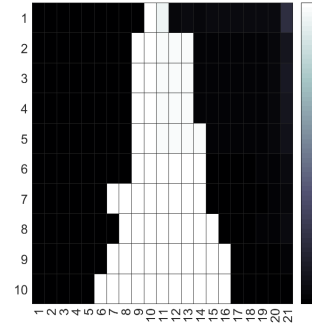


Figure 9: Optimal Policy for 'hit'

## 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  $N_0=100$ . Note that each algorithm has been run three times, producing nine plots.

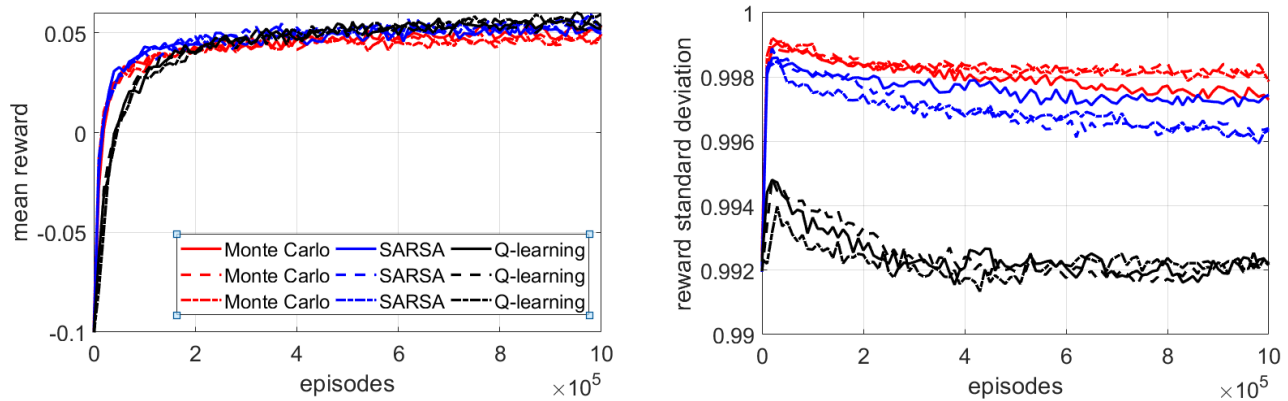


Figure 10: Learning Curves the three methods used

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

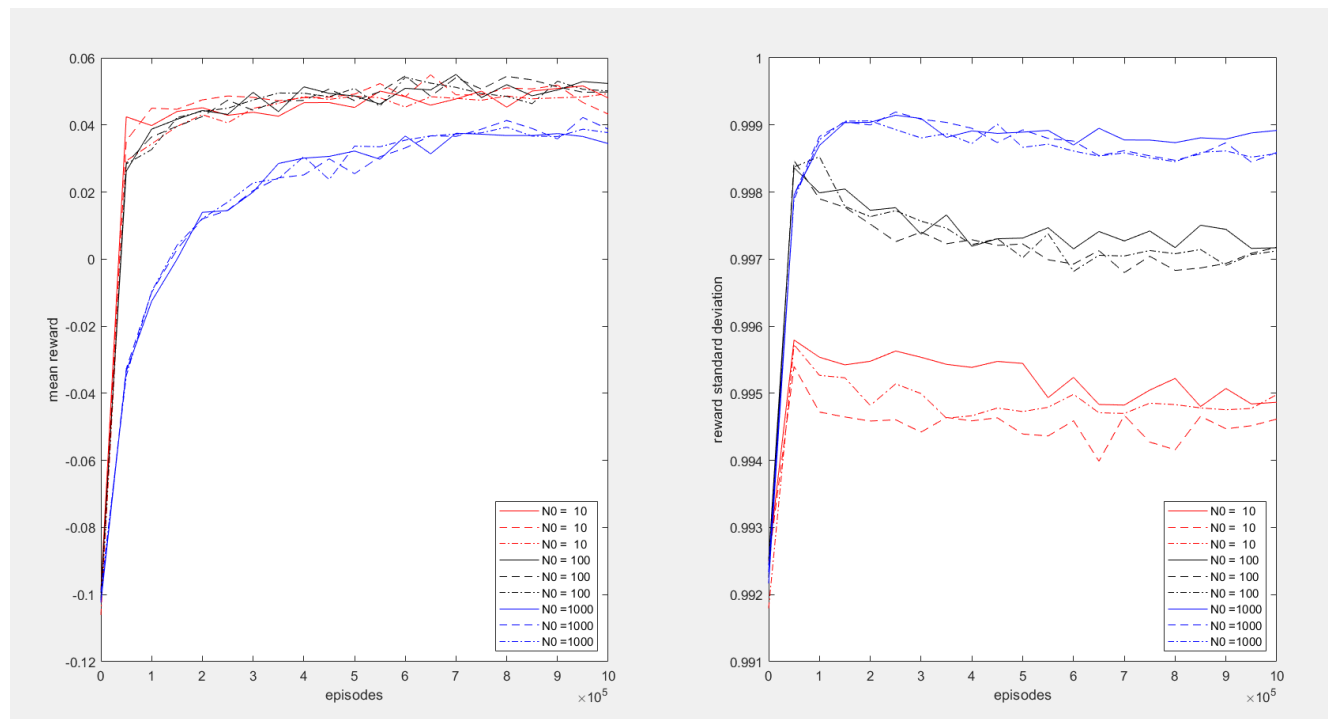


Figure 11: Learning Curves for SARSA with varying  $N_0$ .  $N_0=100$  offers best performance

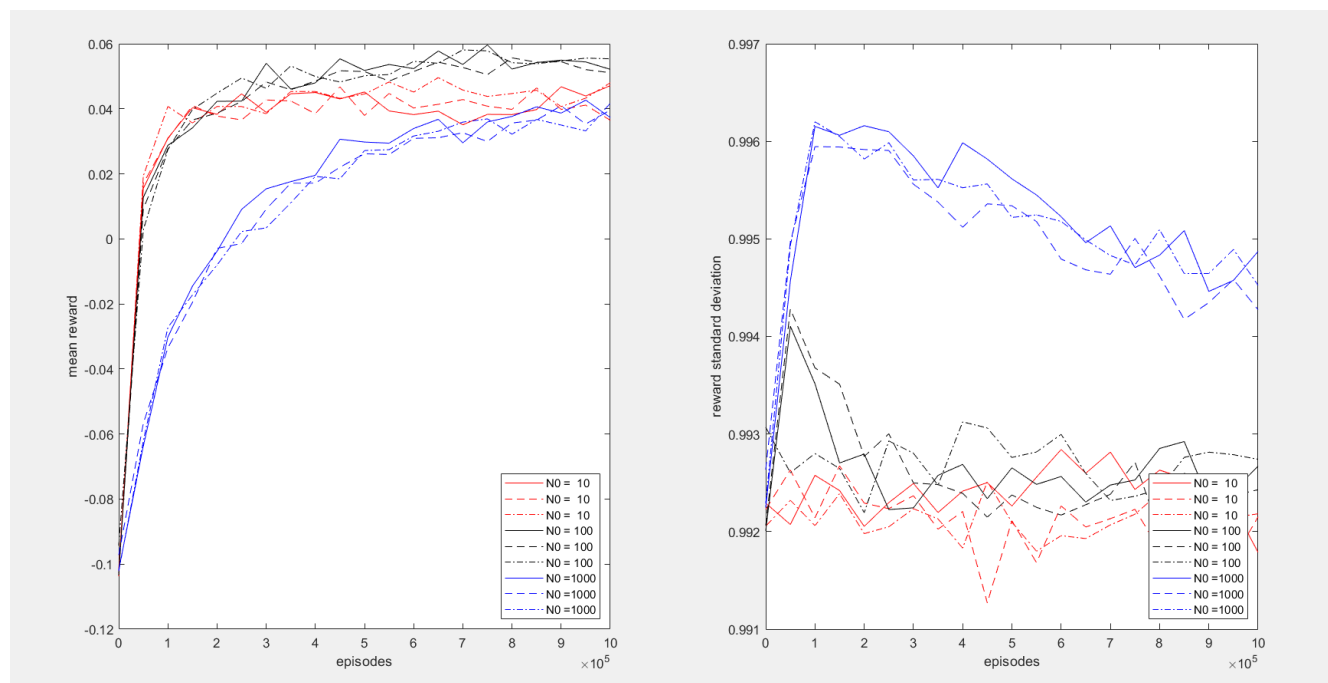


Figure 12: Learning Curves for Q-learning with varying  $N_0$ .  $N_0=100$  offers best performance

## 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.

```

1  %STEP FUNCTION
2  function [s_n , r , fin]=step(s , a)
3  ds=s(1);
4  ps=s(2);
5  fin=0;
6  if a==1
7      val=randi(10);
8      mult=2*(rand() > (1/3)) - 1; % 2/3 chance 1, 1/3 chance -1
9      ps=ps+mult*val;
10     if ps<1 || ps>21
11         r=-1;
12         fin=1;
13     else
14         r=0;
15     end
16 else
17     fin=1;
18     while ds<17 && ds>=1
19         val=randi(10);
20         mult=2*(rand() > (1/3)) - 1;
21         ds=ds+mult*val;
22     end
23     if ds<1 || ds>21 || ds<ps
24         r=1;
25     elseif ds==ps
26         r=0;
27     else
28         r=-1;
29     end
30 end
31 s_n=[ds ps];
32 end
33
34 %MONTE CARLO FUNCTION
35 function [R,R_test,Q,pol,count_state_action]=mc(n_test,n_epi,d_test,N0,
    ev,dshow)
36 ind_show=1;
37 ind_test_vec=[1 d_test:d_test:n_epi];
38 ind_test=1;
39 %
40 t0=tic;%timer
41 pol=0.5*ones(10,21,2); % pi('hit',s)

```

```

42 count_state_action=zeros(10,21,2);
43 Q=zeros(10,21,2);
44 R=zeros(n_test,length(ind_test_vec));
45 R_test=zeros(1,n_epi);
46 for epi=1:n_epi
47     fin=0;
48     s=randi(10,[1 2]);
49     trace=[];
50     while fin==0
51         temp=rand();
52         a=2-(temp<pol(s(1),s(2)));
53         [sn,r,fin]=step(s,a);
54         trace=[trace;[s a]];
55         s=sn;
56     end
57     R_test(epi)=r;
58     checked=zeros(10,21,2);
59     for ind=1:size(trace,1)
60         if ev || ~checked(trace(ind,1),trace(ind,2),trace(ind,3))
61             count_state_action(trace(ind,1),trace(ind,2),trace(ind,3))
62                 =count_state_action(trace(ind,1),trace(ind,2),trace(ind,3))+1;
63             Q_hat=Q(trace(ind,1),trace(ind,2),trace(ind,3));
64             Q_hat=Q_hat+(r-Q_hat)/count_state_action(trace(ind,1),
65                 trace(ind,2),trace(ind,3));
66             Q(trace(ind,1),trace(ind,2),trace(ind,3))=Q_hat;
67             checked(trace(ind,1),trace(ind,2),trace(ind,3))=1;
68         end
69     end
70     count_state=count_state_action(:,:,1)+count_state_action(:,:,2);
71     checked=zeros(10,21);
72     for ind=1:size(trace,1)
73         if ev || ~checked(trace(ind,1),trace(ind,2))
74             eps=N0/(N0+count_state(trace(ind,1),trace(ind,2)));
75             if Q(trace(ind,1),trace(ind,2),1)>Q(trace(ind,1),trace(ind,2),2)
76                 pol(trace(ind,1),trace(ind,2),1)=1-eps/2;
77                 pol(trace(ind,1),trace(ind,2),2)=eps/2;
78             end
79             if Q(trace(ind,1),trace(ind,2),1)<Q(trace(ind,1),trace(ind,2),2)
80                 pol(trace(ind,1),trace(ind,2),2)=1-eps/2;
81                 pol(trace(ind,1),trace(ind,2),1)=eps/2;
82             end
83             if Q(trace(ind,1),trace(ind,2),1)==Q(trace(ind,1),trace(ind,2),2)
84                 pol(trace(ind,1),trace(ind,2),1)=0.5;

```



```

83         pol(trace(ind,1),trace(ind,2),2)=0.5;
84     end
85     checked(trace(ind,1),trace(ind,2))=1;
86 end
87 end
88 if epi==ind_test_vec(ind_test)
89     for train_epi=1:n_test
90         fin=0;
91         s=randi(10,[1 2]);
92         trace=[];
93         while fin==0
94             temp=rand();
95             a=2-(temp<pol(s(1),s(2))); % is 1 for 'hit' 2 for '
                stick'
96             [sn,r,fin]=step(s,a);
97             s=sn;
98         end
99         R(train_epi,ind_test)=r;
100     end
101     ind_test=ind_test+1;
102 end
103 if (epi/n_epi)/dshow>ind_show
104     ind_show=ind_show+1;
105     [round(100*(epi/n_epi)) toc(t0)]
106     t0=tic;
107 end
108 end
109 end
110 %SARSA FUNCTION
111 function [R,R_train,Q,pol,count_state_action]=fsarsa(n_train,n_epi,
    d_test,N0,dshow,a_choice)
112 % counter stuff for visulaizing progress
113 ind_show=1;
114 ind_test_vec=[1 d_test:d_test:n_epi];
115 ind_test=1;
116 %
117 t0=tic;%timer
118 pol=0.5*ones(10,21,2); % pi('hit',s)
119 count_state_action=zeros(10,21,2);
120 Q=zeros(10,21,2); %dim 1 is 'hit dim 2 'stick'
121 R_train=zeros(1,n_epi);
122 R=zeros(n_train,length(ind_test_vec));
123 for epi=1:n_epi
124     fin=0;
125     s=randi(10,[1 2]);
126     temp=rand();
127     A=2-(temp<pol(s(1),s(2))); % is 1 for 'hit' 2 for 'stick'

```



```

128 while fin==0
129     [sn,r,fin]=step(s,A);
130     temp=rand();
131     count_state_action(s(1),s(2),A)=count_state_action(s(1),s(2),A
        )+1;
132     if a_choice==0
133         a=1/count_state_action(s(1),s(2),A);
134     else
135         a=a_choice;
136     end
137     if fin==0
138         An=2-(temp<pol(sn(1),sn(2))); % is 1 for 'hit' 2 for '
            stick '
139         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));
140     else
141         Q(s(1),s(2),A)=Q(s(1),s(2),A)+a*(r-Q(s(1),s(2),A));
142     end
143     eps=N0/(N0+count_state_action(s(1),s(2),1)+count_state_action(
        s(1),s(2),2));
144     if Q(s(1),s(2),1)>Q(s(1),s(2),2)
145         pol(s(1),s(2),1)=1-eps/2;
146         pol(s(1),s(2),2)=eps/2;
147     end
148     if Q(s(1),s(2),1)<Q(s(1),s(2),2)
149         pol(s(1),s(2),2)=1-eps/2;
150         pol(s(1),s(2),1)=eps/2;
151     end
152     if Q(s(1),s(2),1)==Q(s(1),s(2),2)
153         pol(s(1),s(2),1)=0.5;
154         pol(s(1),s(2),2)=0.5;
155     end
156     if fin==0
157         A=An;
158         s=sn;
159     end
160 end
161 R_train(eps)=r;
162 if eps==ind_test_vec(ind_test)
163     for train_eps=1:n_train
164         fin=0;
165         s=randi(10,[1 2]);
166         while fin==0
167             temp=rand();
168             a=2-(temp<pol(s(1),s(2))); % is 1 for 'hit' 2 for '
                stick '
169             [sn,r,fin]=step(s,a);

```

```

170         s=sn ;
171     end
172     R(train_epi , ind_test)=r ;
173 end
174 ind_test=ind_test+1;
175 end
176 %timer/counter for progress monitoring
177 if (epi/n_epi)/dshow>ind_show
178     ind_show=ind_show+1;
179     [round(100*(epi/n_epi)) toc(t0)]
180     t0=tic ;
181 end
182 end
183 end
184 %Q LEARN FUNCTION
185 function [R,R_train,Q,pol , count_state_action]=Q_learn (n_train , n_epi ,
    d_test , N0,dshow , eps_choice)
186 % counter stuff for visualizing progress
187 ind_show=1;
188 ind_test_vec=[1 d_test:d_test:n_epi];
189 ind_test=1;
190 %
191 t0=tic;%timer
192 pol=0.5*ones(10,21,2); % pi('hit',s)
193 count_state_action=zeros(10,21,2);
194 Q=zeros(10,21,2); %dim 1 is 'hit dim 2 'stick'
195 R_train=zeros(1,n_epi);
196 R=zeros(n_train , length(ind_test_vec));
197 for epi=1:n_epi
198     fin=0;
199     s=randi(10,[1 2]);
200     temp=rand();
201     A=2-(temp<pol(s(1),s(2))); % is 1 for 'hit' 2 for 'stick'
202     while fin==0
203         [sn,r,fin]=step(s,A);
204         temp=rand();
205         count_state_action(s(1),s(2),A)=count_state_action(s(1),s(2),A
            )+1;
206         a=1/count_state_action(s(1),s(2),A);
207         if fin==0
208             An=2-(temp<pol(sn(1),sn(2))); % is 1 for 'hit' 2 for '
                stick'
209             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));
210         else
211             Q(s(1),s(2),A)=Q(s(1),s(2),A)+a*(r-Q(s(1),s(2),A));
212         end

```

```

213     if eps_choice==0
214         eps=N0/(N0+count_state_action(s(1),s(2),1)+
                count_state_action(s(1),s(2),2));
215     else
216         eps=eps_choice;
217     end
218     if Q(s(1),s(2),1)>Q(s(1),s(2),2)
219         pol(s(1),s(2),1)=1-eps/2;
220         pol(s(1),s(2),2)=eps/2;
221     end
222     if Q(s(1),s(2),1)<Q(s(1),s(2),2)
223         pol(s(1),s(2),2)=1-eps/2;
224         pol(s(1),s(2),1)=eps/2;
225     end
226     if Q(s(1),s(2),1)==Q(s(1),s(2),2)
227         pol(s(1),s(2),1)=0.5;
228         pol(s(1),s(2),2)=0.5;
229     end
230     if fin==0
231         A=An;
232         s=sn;
233     end
234 end
235 R_train(eps)=r;
236 if epi==ind_test_vec(ind_test)
237     for train_epi=1:n_train
238         fin=0;
239         s=randi(10,[1 2]);
240         trace=[];
241         while fin==0
242             temp=rand();
243             a=2-(temp<pol(s(1),s(2))); % is 1 for 'hit' 2 for '
                stick'
244             [sn,r,fin]=step(s,a);
245             s=sn;
246         end
247         R(train_epi,ind_test)=r;
248     end
249     ind_test=ind_test+1;
250 end
251 %timer/counter for progress monitoring
252 if (epi/n_epi)/dshow>ind_show
253     ind_show=ind_show+1;
254     [round(100*(epi/n_epi)) toc(t0)]
255     t0=tic;
256 end
257 end

```

```

258 end
259
260 %PLOTING
261
262 clc;clear;close all
263 %%
264 n_test=16*10^4;
265 n_epi=10^6;
266 dshow=0.1;
267 d_test=5*10^4;
268 figure(1)
269 N0_vec=[10 100 1000];
270 c_vec=['r';'k';'b'];
271 for ind=1:length(N0_vec)
272     N0=N0_vec(ind);
273     [R,R_train,Q,pol,count_state_action]=mc(n_test,n_epi,d_test,N0,0,
        dshow);
274     subplot(1,2,1)
275     plot([1 d_test:d_test:n_epi],mean(R),'color',c_vec(ind))
276     hold on
277     subplot(1,2,2)
278     plot([1 d_test:d_test:n_epi],std(R),'color',c_vec(ind))
279     hold on
280     [R,R_train,Q,pol,count_state_action]=mc(n_test,n_epi,d_test,N0,0,
        dshow);
281     subplot(1,2,1)
282     plot([1 d_test:d_test:n_epi],mean(R),'color',c_vec(ind),'Linestyle',
        '—')
283     hold on
284     subplot(1,2,2)
285     plot([1 d_test:d_test:n_epi],std(R),'color',c_vec(ind),'Linestyle',
        '—')
286     hold on
287     [R,R_train,Q,pol,count_state_action]=mc(n_test,n_epi,d_test,N0,0,
        dshow);
288     subplot(1,2,1)
289     plot([1 d_test:d_test:n_epi],mean(R),'color',c_vec(ind),'Linestyle',
        '—.')
290     hold on
291     subplot(1,2,2)
292     plot([1 d_test:d_test:n_epi],std(R),'color',c_vec(ind),'Linestyle',
        '—.')
293     hold on
294 end
295 subplot(1,2,1)
296 legend(strcat('N0 = ',num2str([10 10 10 100 100 100 1000 1000 1000]')),
    , 'Location','southeast')

```

```

297 xlabel( 'episodes ' )
298 ylabel( 'mean reward ' )
299 subplot( 1,2,2 )
300 legend( strcat( 'N0 = ', num2str([10 10 10 100 100 100 1000 1000 1000]'))
        , 'Location', 'southeast' )
301 xlabel( 'episodes ' )
302 ylabel( 'reward standard deviation ' )
303 %%
304 n_test=1;
305 n_epi=10^6;
306 N0=100;
307 [R, R_train, Q, pol, count_state_action]=mc( n_test, n_epi, d_test, N0, 0, dshow
        );
308 [X,Y] = meshgrid( 1:10, 1:21 );
309 v=max(Q, [], 3);
310 figure( 2 )
311 colormap( 'bone' )
312 surf(X,Y,v, 'EdgeColor', 'none' );
313 xlabel( 'Dealer first card' )
314 ylabel( 'Player sum' )
315 zlabel( 'State Value' )
316 figure( 3 )
317 h=heatmap( pol(:, :, 1) )
318 h.Colormap=colormap( 'bone' );
319 %%
320 n_test=16*10^4;
321 n_epi=10^6;
322 dshow=0.1;
323 d_test=5*10^4;
324 figure( 4 )
325 N0_vec=[10 100 1000];
326 c_vec=['r'; 'k'; 'b'];
327 for ind=1:length( N0_vec )
328     N0=N0_vec( ind );
329     [R, R_train, Q, pol, count_state_action]=fsarsa( n_test, n_epi, d_test, N0
        , dshow, 0 );
330     subplot( 1,2,1 )
331     plot( [1 d_test:d_test:n_epi], mean(R), 'color', c_vec( ind ) )
332     hold on
333     subplot( 1,2,2 )
334     plot( [1 d_test:d_test:n_epi], std(R), 'color', c_vec( ind ) )
335     hold on
336     [R, R_train, Q, pol, count_state_action]=fsarsa( n_test, n_epi, d_test, N0
        , dshow, 0 );
337     subplot( 1,2,1 )
338     plot( [1 d_test:d_test:n_epi], mean(R), 'color', c_vec( ind ), 'LineStyle
        ', '—' )

```

```

339     hold on
340     subplot(1,2,2)
341     plot([1 d_test:d_test:n_epi],std(R),'color',c_vec(ind),'Linestyle'
342           ,'_-')
343     hold on
344     [R,R_train,Q,pol,count_state_action]=fsarsa(n_test,n_epi,d_test,N0
345           ,dshow,0);
346     subplot(1,2,1)
347     plot([1 d_test:d_test:n_epi],mean(R),'color',c_vec(ind),'Linestyle'
348           ,'_-')
349     hold on
350     subplot(1,2,2)
351     plot([1 d_test:d_test:n_epi],std(R),'color',c_vec(ind),'Linestyle'
352           ,'_-')
353     hold on
354     subplot(1,2,1)
355     legend(strcat('N0 = ',num2str([10 10 10 100 100 100 1000 1000 1000]'))
356           , 'Location', 'southeast')
357     xlabel('episodes')
358     ylabel('mean reward')
359     subplot(1,2,2)
360     legend(strcat('N0 = ',num2str([10 10 10 100 100 100 1000 1000 1000]'))
361           , 'Location', 'southeast')
362     xlabel('episodes')
363     ylabel('reward standard deviation')
364 %%
365 n_test=16*10^4;
366 n_epi=10^6;
367 dshow=0.1;
368 d_test=5*10^4;
369 figure(5)
370 a_vec=[0.001 0.01 0.1 1 0];
371 c_vec=['r';'b';'g';'m';'k'];
372 N0=100;
373 for ind=1:length(a_vec)
374     a=a_vec(ind);
375     [R,R_train,Q,pol,count_state_action]=fsarsa(n_test,n_epi,d_test,N0
376           ,dshow,a);
377     subplot(1,2,1)
378     plot([1 d_test:d_test:n_epi],mean(R),'color',c_vec(ind))
379     hold on
380     subplot(1,2,2)
381     plot([1 d_test:d_test:n_epi],std(R),'color',c_vec(ind))
382     hold on
383     [R,R_train,Q,pol,count_state_action]=fsarsa(n_test,n_epi,d_test,N0
384           ,dshow,a);

```

```

378     subplot(1,2,1)
379     plot([1 d_test:d_test:n_epi],mean(R),'color',c_vec(ind),'Linestyle'
380           ',','_'))
381     hold on
382     subplot(1,2,2)
383     plot([1 d_test:d_test:n_epi],std(R),'color',c_vec(ind),'Linestyle'
384           ',','_'))
385     hold on
386     [R,R_train,Q,pol,count_state_action]=fsarsa(n_test,n_epi,d_test,N0
387           ,dshow,a);
388     subplot(1,2,1)
389     plot([1 d_test:d_test:n_epi],mean(R),'color',c_vec(ind),'Linestyle'
390           ',','-'))
391     hold on
392     subplot(1,2,2)
393     plot([1 d_test:d_test:n_epi],std(R),'color',c_vec(ind),'Linestyle'
394           ',','-'))
395     hold on
396     legend(strcat('a = ',num2str([0.001 0.001 0.001 0.01 0.01 0.01 0.1 0.1
397           0.1 1 1 1 0 0 0])), 'Location','southeast')
398     xlabel('episodes')
399     ylabel('mean reward')
400     subplot(1,2,2)
401     legend(strcat('a = ',num2str([0.001 0.001 0.001 0.01 0.01 0.01 0.1 0.1
402           0.1 1 1 1 0 0 0])), 'Location','southeast')
403     xlabel('episodes')
404     ylabel('reward standard deviation')
405     %%
406     n_test=1;
407     n_epi=10^6;
408     N0=100;
409     [R,R_train,Q,pol,count_state_action]=fsarsa(n_test,n_epi,d_test,N0,
410           dshow,0);
411     [X,Y] = meshgrid(1:10,1:21);
412     v=max(Q,[],3);
413     figure(6)
414     colormap('bone')
415     surf(X,Y,v,'EdgeColor','none');
416     xlabel('Dealer first card')
417     ylabel('Player sum')
418     zlabel('State Value')
419     figure(7)
420     h=heatmap(pol(:,:,1))
421     h.Colormap=colormap('bone');
422     %%

```



```

417 n_test=16*10^4;
418 n_epi=10^6;
419 dshow=0.1;
420 d_test=5*10^4;
421 figure(8)
422 N0_vec=[10 100 1000];
423 c_vec=['r'; 'k'; 'b'];
424 for ind=1:length(N0_vec)
425     N0=N0_vec(ind);
426     [R,R_train,Q,pol,count_state_action]=Q_learn(n_test,n_epi,d_test,
        N0,dshow,0);
427     subplot(1,2,1)
428     plot([1 d_test:d_test:n_epi],mean(R),'color',c_vec(ind))
429     hold on
430     subplot(1,2,2)
431     plot([1 d_test:d_test:n_epi],std(R),'color',c_vec(ind))
432     hold on
433     [R,R_train,Q,pol,count_state_action]=Q_learn(n_test,n_epi,d_test,
        N0,dshow,0);
434     subplot(1,2,1)
435     plot([1 d_test:d_test:n_epi],mean(R),'color',c_vec(ind),'Linestyle'
        ',','_'))
436     hold on
437     subplot(1,2,2)
438     plot([1 d_test:d_test:n_epi],std(R),'color',c_vec(ind),'Linestyle'
        ',','_'))
439     hold on
440     [R,R_train,Q,pol,count_state_action]=Q_learn(n_test,n_epi,d_test,
        N0,dshow,0);
441     subplot(1,2,1)
442     plot([1 d_test:d_test:n_epi],mean(R),'color',c_vec(ind),'Linestyle'
        ',','-.'))
443     hold on
444     subplot(1,2,2)
445     plot([1 d_test:d_test:n_epi],std(R),'color',c_vec(ind),'Linestyle'
        ',','-.'))
446     hold on
447 end
448 subplot(1,2,1)
449 legend(strcat('N0 = ',num2str([10 10 10 100 100 100 1000 1000 1000])),
        ',','Location','southeast')
450 xlabel('episodes')
451 ylabel('mean reward')
452 subplot(1,2,2)
453 legend(strcat('N0 = ',num2str([10 10 10 100 100 100 1000 1000 1000])),
        ',','Location','southeast')
454 xlabel('episodes')

```

```

455 ylabel('reward standard deviation')
456 %%
457 n_test=16*10^4;
458 n_epi=10^6;
459 dshow=0.1;
460 d_test=5*10^4;
461 figure(9); close; figure(9)
462 e_vec=[0.001 0.01 0.1 0.5 0];
463 c_vec=['r'; 'b'; 'g'; 'm'; 'k'];
464 N0=100;
465 for ind=1:length(e_vec)
466     e=e_vec(ind);
467     [R,R_train,Q,pol,count_state_action]=Q_learn(n_test,n_epi,d_test,
468         N0,dshow,e);
469     subplot(1,2,1)
470     plot([1 d_test:d_test:n_epi],mean(R),'color',c_vec(ind))
471     hold on
472     subplot(1,2,2)
473     plot([1 d_test:d_test:n_epi],std(R),'color',c_vec(ind))
474     hold on
475     [R,R_train,Q,pol,count_state_action]=Q_learn(n_test,n_epi,d_test,
476         N0,dshow,e);
477     subplot(1,2,1)
478     plot([1 d_test:d_test:n_epi],mean(R),'color',c_vec(ind),'Linestyle',
479         '—')
480     hold on
481     subplot(1,2,2)
482     plot([1 d_test:d_test:n_epi],std(R),'color',c_vec(ind),'Linestyle',
483         '—')
484     hold on
485     [R,R_train,Q,pol,count_state_action]=Q_learn(n_test,n_epi,d_test,
486         N0,dshow,e);
487     subplot(1,2,1)
488     plot([1 d_test:d_test:n_epi],mean(R),'color',c_vec(ind),'Linestyle',
489         '—')
490     hold on
491     subplot(1,2,2)
492     plot([1 d_test:d_test:n_epi],std(R),'color',c_vec(ind),'Linestyle',
493         '—')
494     hold on
495 end
496 subplot(1,2,1)
497 legend(strcat('\epsilon = ',num2str([0.001 0.001 0.001 0.01 0.01 0.01
498     0.1 0.1 0.1 0.5 0.5 0.5 0 0 0])), 'Location','southeast')
499 xlabel('episodes')
500 ylabel('mean reward')
501 subplot(1,2,2)

```

```

494 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')
495 xlabel('episodes')
496 ylabel('reward standard deviation')
497 %%
498 n_test=1;
499 n_epi=10^6;
500 N0=100;
501 [R, R_train, Q, pol, count_state_action]=Q_learn(n_test, n_epi, d_test, N0,
      dshow, 0);
502 [X, Y] = meshgrid(1:10, 1:21);
503 v=max(Q, [], 3);
504 figure(10)
505 colormap('bone')
506 surf(X, Y, v, 'EdgeColor', 'none');
507 xlabel('Dealer first card')
508 ylabel('Player sum')
509 zlabel('State Value')
510 figure(11)
511 h=heatmap(pol(:, :, 1))
512 h.Colormap=colormap('bone');
513 %%
514 n_test=25*10^4;
515 n_epi=10^6;
516 dshow=0.95;
517 d_test=10^4;
518 figure(12); close; figure(12)
519 N0=100;
520 %MC-RED
521 [R, R_train, Q, pol, count_state_action]=mc(n_test, n_epi, d_test, N0, 0, dshow
      );
522 subplot(1, 2, 1)
523 plot([1 d_test:d_test:n_epi], mean(R), 'color', 'r', 'Linestyle', '-')
524 hold on
525 subplot(1, 2, 2)
526 plot([1 d_test:d_test:n_epi], std(R), 'color', 'r', 'Linestyle', '-')
527 hold on
528 [R, R_train, Q, pol, count_state_action]=mc(n_test, n_epi, d_test, N0, 0, dshow
      );
529 subplot(1, 2, 1)
530 plot([1 d_test:d_test:n_epi], mean(R), 'color', 'r', 'Linestyle', '-')
531 hold on
532 subplot(1, 2, 2)
533 plot([1 d_test:d_test:n_epi], std(R), 'color', 'r', 'Linestyle', '-')
534 hold on
535 [R, R_train, Q, pol, count_state_action]=mc(n_test, n_epi, d_test, N0, 0, dshow
      );

```

```

536 subplot(1,2,1)
537 plot([1 d_test:d_test:n_epi],mean(R), 'color','r', 'Linestyle','-.')
538 hold on
539 subplot(1,2,2)
540 plot([1 d_test:d_test:n_epi],std(R), 'color','r', 'Linestyle','-.')
541 hold on
542 %FSARSA-BLUE
543 [R,R_train,Q,pol,count_state_action]=fsarsa(n_test,n_epi,d_test,N0,
    dshow,0);
544 subplot(1,2,1)
545 plot([1 d_test:d_test:n_epi],mean(R), 'color','b', 'Linestyle','-.')
546 hold on
547 subplot(1,2,2)
548 plot([1 d_test:d_test:n_epi],std(R), 'color','b', 'Linestyle','-.')
549 hold on
550 [R,R_train,Q,pol,count_state_action]=fsarsa(n_test,n_epi,d_test,N0,
    dshow,0);
551 subplot(1,2,1)
552 plot([1 d_test:d_test:n_epi],mean(R), 'color','b', 'Linestyle','—')
553 hold on
554 subplot(1,2,2)
555 plot([1 d_test:d_test:n_epi],std(R), 'color','b', 'Linestyle','—')
556 hold on
557 [R,R_train,Q,pol,count_state_action]=fsarsa(n_test,n_epi,d_test,N0,
    dshow,0);
558 subplot(1,2,1)
559 plot([1 d_test:d_test:n_epi],mean(R), 'color','b', 'Linestyle','-.')
560 hold on
561 subplot(1,2,2)
562 plot([1 d_test:d_test:n_epi],std(R), 'color','b', 'Linestyle','-.')
563 hold on
564 %Q-BLACK
565 [R,R_train,Q,pol,count_state_action]=Q_learn(n_test,n_epi,d_test,N0,
    dshow,0);
566 subplot(1,2,1)
567 plot([1 d_test:d_test:n_epi],mean(R), 'color','k', 'Linestyle','-.')
568 hold on
569 subplot(1,2,2)
570 plot([1 d_test:d_test:n_epi],std(R), 'color','k', 'Linestyle','-.')
571 hold on
572 [R,R_train,Q,pol,count_state_action]=Q_learn(n_test,n_epi,d_test,N0,
    dshow,0);
573 subplot(1,2,1)
574 plot([1 d_test:d_test:n_epi],mean(R), 'color','k', 'Linestyle','—')
575 hold on
576 subplot(1,2,2)
577 plot([1 d_test:d_test:n_epi],std(R), 'color','k', 'Linestyle','—')

```

```
578 hold on
579 [R, R_train, Q, pol, count_state_action] = Q_learn(n_test, n_epi, d_test, N0,
    dshow, 0);
580 subplot(1, 2, 1)
581 plot([1 d_test:d_test:n_epi], mean(R), 'color', 'k', 'Linestyle', '-.')
582 hold on
583 subplot(1, 2, 2)
584 plot([1 d_test:d_test:n_epi], std(R), 'color', 'k', 'Linestyle', '-.')
585 hold on
586 subplot(1, 2, 1)
587 xlabel('episodes')
588 ylabel('mean reward')
589 subplot(1, 2, 2)
590 legend([{'Monte Carlo'}, {'Monte Carlo'}, {'Monte Carlo'}, {'SARSA'}, {'SARSA'}, {'SARSA'}, {'Q-learning'}, {'Q-learning'}, {'Q-learning'}])
591 xlabel('episodes')
592 ylabel('reward standard deviation')
```