```
% Implements a soft policy exploration Monte Carlo estimation algorithm
% to compute the optimal action-value function for the racetrack example.
%
% Mods to try:
% 0) Implement recursive averaging update of Q
%
%    Q_{k+1} <- Q_k + (1/(k+1)) (r_{k+1} - Q_{k})
%
% Result:
%
% 1) Add a fixed step size learning algorithm,
%
%    Q_{k+1} <- Q_k + \alpha (r_{k+1} - Q_{k})
%
%  which should be better for problems where the action value function may
%  change over time which is the case when we are performing value iteration.
%
% Result:
%
% 2) Set Q initially very large to encourage exploration.  A value ~ +5
% should be large enough.
%
% Results:
%
% With geometric update (involving alpha):
%
% With recursive update:
%
% Written by:
% --
% John L. Weatherwax                2007-12-07
%
% email: wax@alum.mit.edu
%
% Please send comments and especially bug reports to the
% above email address.
%
%-----

close all;
clc;

% the fixed step size learning parameter:
alpha = 0.1;

N_EPISODES=10;     % a numerical approximation of +Inf
N_EPISODES=100;
N_EPISODES=2e3;    % 17 seconds
N_EPISODES=5e5;    % 1 hour
N_EPISODES=1.2e7;  % <- should take 24 hours ...

% sample_discrete.m
addpath( genpath( '../../../FullBNT-1.0.4/KPMstats/' ) );
addpath( genpath( '/home/wax/Programming/Matlab/MathPath' ) );
try, BayesianNetworks; catch, end

rand('seed',0); randn('seed',0);

% generate the race track and initialize some sizes:
```

```
RT = mk_rt(1); [maxNPii,maxNPjj] = size(RT);

% the dimensions of the velocity state:
maxNVii = 6; maxNVjj = 6;

% the dimensions of the possible actions:
maxNAii = 3; maxNAjj = 3;

% the maximal state/action dimenions:
%
% a state consists of [pii,pjj,vii,vjj] with
% pii \in maxNPii, pjj \in maxNPjj, vii \in 0:5, vjj \in 0:5
maxNStates  = prod([maxNPii,maxNPjj,maxNVii,maxNVjj]);   %  ~ 9216 states!
maxNActions = prod([maxNAii,maxNAjj]);

% storage for the objects we will calculate:
Q = zeros(maxNStates,maxNActions);         % the initial action-value function
%Q = +5*ones(maxNStates,maxNActions);      % the initial action-value function taken to
encourage exploration
%Q = the_valid_spots;

firstSARewSum = zeros(maxNStates,maxNActions);
firstSARewCnt = zeros(maxNStates,maxNActions);

%timePerPlay   = zeros(1,N_EPISODES);

% enumerate the possible starting locations:
posStarts = find(RT(end,:)); nPosStarts = length(posStarts);

% initialize our policy:
%pol_pi        = zeros(maxNStates,maxNActions);         % the storage for our initial
policy
pol_pi = init_unif_policy(RT,
maxNStates,maxNActions,maxNPii,maxNPjj,maxNVii,maxNVjj,maxNAii,maxNAjj);

tic
for ei=1:N_EPISODES,

  % (A) generate an episode following the policy pol_pi:
  %
  [stateseen,act_taken,rew] = gen_rt_episode(ei,pol_pi,
RT,posStarts,nPosStarts,maxNStates,maxNActions,maxNPii,maxNPjj,maxNVii,maxNVjj,maxNAii,ma
xNAjj);

  % (B) estimate the action value function "Q" via monte carlo methods:
  %
  [Q,firstSARewCnt,firstSARewSum] = mcEstQ(stateseen,act_taken,rew,
firstSARewCnt,firstSARewSum,Q, maxNPii,maxNPjj,maxNVii,maxNVjj);

  % (C) update our policy:
  %
  [pol_pi] = rt_pol_mod(stateseen,Q, pol_pi,
maxNPii,maxNPjj,maxNVii,maxNVjj,maxNAii,maxNAjj);

end % end number of episods loop
toc

%fprintf('timePerPlay = %f\n',mean(timePerPlay));
```

```
% plot the learned action-value function Q^{*}
% ... skipped for now


% plot the learned state-value function V^{*} (greedy from Q) as a function of position
ONLY:
%
% This means that we average out
% -- the action variables in Q
% -- the velocity state variables (vxx,vyy)
%
% We assume that 0.0 are variables that have NOT been updated and are INACCESABLE states
...
%
% to just look at the POSITION part of the state value function
%
Q( find(Q(:)==0.0) ) = NaN;                              % <- replace zeros with NaN's
V = nanmean( Q, 2 );                                     % <- average out the action
variables
V( find(isnan(V(:))) ) = 0.0;                            % <- replace back with zeros
V = reshape( V, [maxNPii,maxNPjj,maxNVii,maxNVjj] );  % <- do the same for the velocities
...
V( find(V(:)==0.0) ) = NaN;
V = nanmean( V, 4 );
V = nanmean( V, 3 );
V( find(isnan(V(:))) ) = 0.0;

figure; imagesc( V ); colorbar;
xlabel( 'jj location' ); ylabel( 'ii location' );
drawnow;
saveas( gcf, sprintf('avg_state_value_fn_%d',N_EPISODES), 'png' );

return;
```