```
function [theta,avg_ts_per_episode] = mnt_car_learn(nEpisodes,
epsilon,gamma,alpha,lambda,ACC_ET, DO_PLOTS)
% MNT_CAR_LEARN - Learns the linear parameters for the mountain car example
% using linear, gradient-decent SARSA(\lambda) with binary features and an \epsilon-
greedy policy
%
% Inputs:
%   epsilon:   the probabiity of an exploratory move
%   gamma:     the discount factor
%   alpha:     the learning parameter
%   lambda:    the blending parameter
%   ACC_ET:    the type of elagability trace to use (accumulating=1 or replacing=0)
%   nEpisodes: the number of learning episodes
%
% Outputs:
%   theta:                the linear coefficients
%   avg_ts_per_episode:   the average number of timesteps per episdes
% Written by:
% --
% John L. Weatherwax                2008-02-19
%
% email: wax@alum.mit.edu
%
% Please send comments and especially bug reports to the
% above email address.
%
%-----

CLEAR_OTHER_TRACES = 1;

% bounds on the position and velocity:
pos_bnds = [  -1.2, +0.6  ]; mcar_goal_position = 0.5;
vel_bnds = [ -0.07, +0.07 ];

% We'll do "nTilings" tilings with "np" and "nv" tiles in each dimension:
nTilings = 10; np = 8; nv = 8;

% Compute the individual tile widths:
dp  = diff( [ pos_bnds(1), mcar_goal_position ] )/np; dv = diff( vel_bnds )/nv;

% The total number of binary features (add 2 for safty in hashing):
nFeatures = nTilings*(np+2)*(nv+2);

% The total number of actions:
nActions = 3; % -1,0,+1

% the total hashing memory size:
memory_size = nActions*nFeatures;

% the maximum number of timesteps we will take during any episode:
%maxNTimesteps = 1e2;
maxNTimesteps = 1e3; % <- seem to require a VERY long time to finish the experiments in
do_mnt_car_Exps.m
%maxNTimesteps = 1e4;
%maxNTimesteps = 1e5;  % <- official number of max total timesteps ...

% The parameters representing our linear function:
theta = zeros(memory_size,1);
```

```
num_ts = zeros(1,nEpisodes);
% do an episode:
for ei=1:nEpisodes,
  if( mod(ei,500)==0 ) fprintf('working on episode %10d...\n',ei); end
  %st = [0.0,-0.5]; % our initial state
  %st = [0.0, 0.0]; % our initial state
  st(1) = unifrnd(pos_bnds(1),mcar_goal_position); % a random initial state
  st(2) = unifrnd(vel_bnds(1),vel_bnds(2));

  et = zeros(memory_size,1); % initialize our eligibility traces

  % compute Q in the state st, an epsilon greedy action selection, and the feature list:
  [Q,action,F] = ret_q_in_st(st,theta,nActions,dp,dv,nTilings,memory_size, epsilon);
  fl_at        = F(action,:);

  for tsi=1:maxNTimesteps,
    et = gamma*lambda*et; % let all traces decay ...
    if( CLEAR_OTHER_TRACES )               % optionally clear other traces ...
      for ai=1:nActions,
        if( ai==action ) continue; end
        et(F(ai,:))=0.0;
      end
    end

    if( mod(tsi,500)==0 ) %mod(tsi,10)==0 )
      fprintf('episode %10d: working on timestep %10d...\n',ei,tsi);
    end
    if( ACC_ET )    % ACCUMULATING eligibility traces
      et(fl_at) = et(fl_at)+1;
    else            % REPLACING eligibility traces
      et(fl_at) = 1;
    end
    %if( mod(tsi,10)==0 ) figure; plot( et, 'o' ); drawnow; pause; end; close all;

    % take action, observe reward r, and next state s':
    [rew,stp1] = next_state(st,action,pos_bnds,mcar_goal_position,vel_bnds);
    %stp1

    if( stp1(1)>=mcar_goal_position )
      break; % the episode is over
    end

    % get our temporal difference:
    delta = rew - Q(action);

    % select the next action using an epsilon greedy policy:
    [Q,actionp,F] = ret_q_in_st(stp1,theta,nActions,dp,dv,nTilings,memory_size, epsilon);

    % update delta/theta/et:
    delta = delta + gamma*Q(actionp);
    %theta = theta + alpha*delta*et;
    theta = theta + (alpha/nTilings)*delta*et;
    %if( mod(tsi,10)==0 ) figure; plot( theta, 'o' ); drawnow; pause; end; close all;

    % update our state/action pair (for the next timestep):
    st     = stp1;
    action = actionp;
    fl_at  = F(actionp,:);

    if( ei==1 && tsi==428 && DO_PLOTS )
```

```
      [the_pos,the_vel,CTG] =
get_ctg(theta,nActions,pos_bnds,dp,vel_bnds,dv,nTilings,memory_size);
      figure; mesh( the_pos, the_vel, CTG );
      %figure; imagesc( the_pos, the_vel, CTG.' ); axis xy; colorbar;
      title( sprintf('episode number %d; timestep 428',ei) ); xlabel( 'position' );
ylabel( 'velocity' ); drawnow;
      %saveas( gcf, sprintf('mnt_ctg_episode_%d_tm_step_328_img',ei), 'png' );
      saveas( gcf, sprintf('mnt_ctg_episode_%d_tm_step_328_mesh',ei), 'png' );
    end

  end % end timestep loop ...
  % print Q at the final timestep of this episode (this should be at the goal)
  %fprintf('Q='); fprintf('%16.6g',Q);  fprintf('\n');
  num_ts(ei) = tsi;

  if( ismember( ei, [ 1, 12, 104, 1000, 9000 ] ) && DO_PLOTS ) % <- plot these cost to go
surfaces
    [the_pos,the_vel,CTG] =
get_ctg(theta,nActions,pos_bnds,dp,vel_bnds,dv,nTilings,memory_size);
    figure; mesh( the_pos, the_vel, CTG );
    %figure; imagesc( the_pos, the_vel, CTG.' ); axis xy; colorbar;
    title( sprintf('episode number %d',ei) ); xlabel( 'position' ); ylabel( 'velocity' );
    drawnow;
    %saveas( gcf, sprintf('mnt_ctg_episode_%d_tm_step_end_img',ei), 'png' );
    saveas( gcf, sprintf('mnt_ctg_episode_%d_tm_step_end_mesh',ei), 'png' );
  end

  %pause;
end

%num_ts
avg_ts_per_episode = mean(num_ts);
```