```
function [Q,ets,cr] =
ex_9_4_dynaQplus(alpha,epsilon,gamma,kappa,nPlanningSteps,mz_fn,s_start,s_end,MAX_N_STEPS
)
% EX_9_4_DYNAQPLUS - Runs the modified (as according to exercise 9.4) dynaQ+ planning
algorithm on a "maze" MZ
%
% Input:
%   gamma: the discount factor
%   epsilon: our epsilon greedy policy
%
% Output:
%   cr: cummulative reward
%
% Ones correspond to locations we can be.
%
% 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.
%
%-----

PLOT_STEPS = 0;

MZ = mz_fn(0); % <- get our initial maze
[sideII,sideJJ] = size(MZ);

% the maximal number of states:
nStates = sideII*sideJJ;

% on each grid we can choose from among at most this many actions:
nActions = 4;

% An array to hold the values of the action-value function
Q = zeros(nStates,nActions);
Q = -3*ones(nStates,nActions);

% for planning we need to storage for the sequence of states we have observed
%  (these are stored as indices ... as produced by the matlab command "sub2ind.m")
seen_states = [];
% for planning we need storage for the sequence of actions taken in each state
%  (0=>this action was never taken; 1=>action was taken)
act_taken   = zeros(nStates,nActions);

% Some arrays to hold the model of the environment (assuming it is determinastic):
Model_ns = zeros(nStates,nActions); % <- next state we will obtain
Model_nr = zeros(nStates,nActions); % <- next reward we will obtain

if( PLOT_STEPS )
  figure; imagesc( zeros(sideII,sideJJ) ); colorbar; hold on;
  plot( s_start(2), s_start(1), 'x', 'MarkerSize', 10, 'MarkerFaceColor', 'k' );
  plot( s_end(2), s_end(1), 'o', 'MarkerSize', 10, 'MarkerFaceColor', 'k' );
end

% keep track of how many timestep we take per episode:
```

```
ets = []; ts=0;

% keep track of how many times we have solved our problem in this number of timesteps:
cr = zeros(1,MAX_N_STEPS+1); cr(1) = 0;

% for exploration: we need to keep track of the number of timesteps elapsed between
% each "real" evaluation of this state and action pair:
%nts_since_visit = +Inf*ones(nStates,nActions);
%nts_since_visit = +10*ones(nStates,nActions);     % <- a very optimistic reward ... to
encourage exploration ...
nts_since_visit = zeros(nStates,nActions);     % <- don't visit a state until we have
visited it at least ONCE ...

% initialize the starting state
st = s_start; sti = sub2ind( [sideII,sideJJ], st(1), st(2) );
for tsi=1:MAX_N_STEPS,
  tic;
%   if( tsi==1 )
%     fprintf('working on step %d...\n',tsi);
%   else
%     fprintf('working on step %d (ptt=%10.6f secs)...\n',tsi, toc); tic;
%   end
  if( 0 && mod(tsi,100)==0 )
    fprintf('working on step %d (ptt=%10.6f secs)...\n',tsi, toc); tic;
  end

  %--
  % Action section:
  %--
  % use an epsilon greedy policy derived FROM Q but modified by how long it has been
since we have visited
  % this state/action pair in a REAL interaction with the environment ... (this is
similar to dynaQplus)
  actSelQ = Q + kappa*sqrt( nts_since_visit );
  [dum,at] = max(actSelQ(sti,:));  % at \in [1,2,3,4]=[up,down,right,left]
  if( rand<epsilon )               % we explore ... with a random action
    tmp=randperm(nActions); at=tmp(1);
  end

  % for planning: keep track of the states/action seen
  if( ~ismember(sti,seen_states) ) seen_states = [ sti; seen_states ]; end
  act_taken( sti, at ) = 1;

  % keep track of the number of timesteps since we visited this state:
  %
  nts_since_visit = nts_since_visit + 1; % increment all non visited state/action pair:
  nts_since_visit(sti,at) = 0;           % reinitialize the one that we DID visit

  %nts_since_visit
  %pause;

  % propagate to state stp1 and collect a reward rew
  MZ = mz_fn(tsi);             % <- get the current maze for this timestep
  [rew,stp1,stp1i] = stNac2stp1(st,at,MZ,sideII,sideJJ,s_end);
  %fprintf('stp1=(%d,%d); rew=%3d...\n',stp1(1),stp1(2),rew);

  % update our action-value function:
  if( ~( (stp1(1)==s_end(1)) && (stp1(2)==s_end(2)) ) ) % stp1 is not the terminal state
    Q(sti,at) = Q(sti,at) + alpha*( rew + gamma*max(Q(stp1i,:)) - Q(sti,at) );
  else                                                  % stp1 IS the terminal state ...
```

```
no Q(s';a') term in the sarsa update
    Q(sti,at) = Q(sti,at) + alpha*( rew - Q(sti,at) );
  end

  % update our model of the environment:
  Model_ns(sti,at) = stp1i;
  Model_nr(sti,at) = rew;

  %--
  % perform some PLANNING STEPS:
  %--
  for pi=1:nPlanningSteps,
    % pick a random state we have seen "r_sti":
    tmp = randperm(length(seen_states)); r_sti = seen_states(tmp(1));

    % pick a random action from the ones that we have seen (in this state) "r_ati":
    pro_action = act_taken(r_sti,:)/sum(act_taken(r_sti,:)); % <- the probabilty of each
specific action ...
    r_ati = sample_discrete( pro_action, 1, 1 );

    % get our models predition of the next state (and reward) "model_sprimei",
"model_rew":
    model_sprimei   = Model_ns(r_sti,r_ati);
    [ii,jj]         = ind2sub( [sideII,sideJJ], model_sprimei );
    model_sprime(1) = ii; model_sprime(2) = jj;
    model_rew       = Model_nr(r_sti,r_ati);
    % use ONLY the modeled reward ...in planning ... just like plane dynaQ:
    model_rew       = model_rew;
    %fprintf( 'm_sprimei=%10d, m_sprimt=(%10d,%10d)\n', model_sprimei, model_sprime(1),
model_sprime(2) );

    % update our action-value function:
    if( ~( (model_sprime(1)==s_end(1)) && (model_sprime(2)==s_end(2)) ) ) % model_sprime
is not the terminal state
      Q(r_sti,r_ati) = Q(r_sti,r_ati) + alpha*( model_rew + gamma*max(Q(model_sprimei,:))
- Q(r_sti,r_ati) );
    else                                                    % model_sprime IS the terminal
state ... no Q(s';a') term in the sarsa update
      Q(r_sti,r_ati) = Q(r_sti,r_ati) + alpha*( model_rew - Q(model_sprimei,r_ati) );
    end

    if( PLOT_STEPS && ts>8000 )
      num2act = { 'UP', 'DOWN', 'RIGHT', 'LEFT' };
      plot( st(2), st(1), 'o', 'MarkerFaceColor', 'g' ); title( ['action =
',num2act(atp1)] );
      plot( stp1(2), stp1(1), 'o', 'MarkerFaceColor', 'k' ); drawnow;
    end

    %pause;
  end % end planning loop

  % shift everything by one (this completes one "step" of the algorithm):
  st = stp1; sti = stp1i; ts=ts+1;

  % for continual planning ... if we have "solved" our maze we will start over:
  if( ( (stp1(1)==s_end(1)) && (stp1(2)==s_end(2)) ) ) % stp1 is the terminal state
    st = s_start; sti = sub2ind( [sideII,sideJJ], st(1), st(2) );
    % record that we took "ts" timesteps to get to the solution (end state)
    ets = [ets; ts]; ts=0;
    % record that we got to the end:
```

```
    cr(tsi+1) = cr(tsi)+1;
  else
    % record that we did not get to the end and our cummulative reward count does not
change:
    cr(tsi+1) = cr(tsi);
  end

end % end episode loop


function [rew,stp1,stp1i] = stNac2stp1(st,act,MZ,sideII,sideJJ,s_end)
% STNAC2STP1 - state and action to state plus one and reward
%

% convert to row/column notation:
ii = st(1); jj = st(2);

% incorporate any actions and fix our position if we end up outside the grid:
%
switch act
 case 1,
  %
  % action = UP
  %
  stp1 = [ii-1,jj];
 case 2,
  %
  % action = DOWN
  %
  stp1 = [ii+1,jj];
 case 3,
  %
  % action = RIGHT
  %
  stp1 = [ii,jj+1];
 case 4
  %
  % action = LEFT
  %
  stp1 = [ii,jj-1];
 otherwise
  error(sprintf('unknown value for of action = %d',act));
end

% adjust our position of we have fallen outside of the grid:
%
if( stp1(1)<1      ) stp1(1)=1;      end
if( stp1(1)>sideII ) stp1(1)=sideII; end
if( stp1(2)<1      ) stp1(2)=1;      end
if( stp1(2)>sideJJ ) stp1(2)=sideJJ; end

% if this trasition has placed us at a forbidden place in our maze no transition takes
place:
if( MZ(stp1(1),stp1(2))==1 )
  stp1 = st;
end

% convert to an index:
stp1i = sub2ind( [sideII,sideJJ], stp1(1), stp1(2) );
```

```
% get the reward for this step:
%
if( (ii==s_end(1)) && (jj==s_end(2)) )
  rew=0;
  %rew = 1;
else
  rew=-1;
  %rew = 0;
end
```