```
function [Q,ets] = wgw_w_kings(alpha,sideII,sideJJ,s_start,s_end,wind,MAX_N_EPISODES)
% WGW_W_KINGS - Performs on-policy sarsa iterative action value funtion estimation
% for the windy grid world example with kings moves (diagonal moves).
%
% Written by:
% --
% John L. Weatherwax                2007-12-03
%
% email: wax@alum.mit.edu
%
% Please send comments and especially bug reports to the
% above email address.
%
%-----

PLOT_STEPS = 0;

gamma = 1;    % <- take this is an undiscounted task

epsilon = 0.1;  % for our epsilon greedy policy

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

% on each grid we can choose from among this many actions
% (except on edges where this action is reduced):
% now the number of actions is greater ... we now have diagonal moves
nActions = 8;

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

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 = zeros(MAX_N_EPISODES,1); ts=0;
for ei=1:MAX_N_EPISODES,
  tic;
  if( ei==1 )
    fprintf('working on episode %d...\n',ei);
  else
    fprintf('working on episode %d (ptt=%10.6f secs)...\n',ei, toc); tic;
  end
  ets(ei,1) = ts+1;
  % initialize the starting state
  st = s_start; sti = sub2ind( [sideII,sideJJ], st(1), st(2) );

  % pick action using an epsilon greedy policy derived from Q:
  [dum,at] = max(Q(sti,:));  % at \in [1,2,3,4,5,6,7,8]=[up,down,right,left,NW,NE,SE,SW]
  if( rand<epsilon )         % explore ... with a random action
    tmp=randperm(nActions); at=tmp(1);
  end

  % begin the episode
  while( 1 )
```

```
    ts=ts+1;
    %fprintf('st=(%d,%d); act=%3d...\n',st(1),st(2),at);
    % propagate to state stp1 and collect a reward rew
    [rew,stp1] = stNac2stp1(st,at,wind,sideII,sideJJ,s_end);
    %fprintf('stp1=(%d,%d); rew=%3d...\n',stp1(1),stp1(2),rew);
    % pick the greedy action from state stp1:
    stp1i = sub2ind( [sideII,sideJJ], stp1(1), stp1(2) );
    % make the greedy action selection:
    [dum,atp1] = max(Q(stp1i,:));
    if( rand<epsilon )         % explore ... with a random action
      tmp=randperm(nActions); atp1=tmp(1);
    end
    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*Q(stp1i,atp1) - 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) );
      break;
    end
    %update (st,at) pair:
    if( PLOT_STEPS && ts>8000 )
      num2act = { 'UP', 'DOWN', 'RIGHT', 'LEFT', 'NW', 'NE', 'SE', 'SW' };
      plot( st(2), st(1), 'o', 'MarkerFaceColor', 'g' ); title( ['action =
',num2act(atp1)] );
      plot( stp1(2), stp1(1), 'o', 'MarkerFaceColor', 'k' ); drawnow;
    end
    st = stp1; sti = stp1i; at = atp1;

    %pause;
  end % end policy while
end % end episode loop



function [rew,stp1] = stNac2stp1(st,act,wind,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);

% get the wind amount:
theWind = wind(jj);

% incorporate any actions and fix our position if we end up outside the grid:
%
switch act
 case 1,
  %
  % action = UP/NORTH
  %
  stp1 = [ii-1-theWind,jj];
 case 2,
  %
  % action = DOWN/SOUTH
  %
  stp1 = [ii+1-theWind,jj];
 case 3,
  %
```

```
  % action = RIGHT/EAST
  %
  stp1 = [ii-theWind,jj+1];
 case 4
  %
  % action = LEFT/WEST
  %
  stp1 = [ii-theWind,jj-1];
 case 5,
  %
  % action = NorthWest/NW
  %
  stp1 = [ii-1-theWind,jj-1];
 case 6,
  %
  % action = NorthEast/NE
  %
  stp1 = [ii-1-theWind,jj+1];
 case 7,
  %
  % action = SouthEast/SE
  %
  stp1 = [ii+1-theWind,jj+1];
 case 8,
  %
  % action = SouthWest
  %
  stp1 = [ii+1-theWind,jj-1];
 otherwise
  error(sprintf('unknown value for of action = %d',act));
end

% adjust our position of we have fallen outside of the grid:
%
outOfBnds=1;
switch outOfBnds
 case 1  % just put us back on the grid ... the nearest point
  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
 case 2  % return us to our original location based on x/y corrdinate
  if( stp1(1)<1      ) stp1(1)=st(1); end
  if( stp1(1)>sideII ) stp1(1)=st(1); end
  if( stp1(2)<1      ) stp1(2)=st(2); end
  if( stp1(2)>sideJJ ) stp1(2)=st(2); end
 case 3
  if( (stp1(1)<1) || (stp1(1)>sideII) || (stp1(2)<1) || (stp1(2)>sideJJ) )
    error('this option never seems to converge ...');
    stp1=st;
  end
 otherwise
  error('unknown value of outofBnds');
end

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

```
else
  rew=-1;
end
```