```
function [rho,Q,Qmax,Act] = R_learn_acq(alpha,beta,h,p,n_servers,MAX_N_EPISODES)
% R_LEARN_ACQ - R learning for the access control que problem
%
% 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.
%
%-----

epsilon = 0.1;  % for our epsilon greedy policy

% the probability of getting each type of client:
op = (1.0-h)/3; p_client = [ op, op, op, h ];

% the client rewards:
client_rew = [ 1, 2, 4, 8 ];

n_diff_clients = length(client_rew);

% a state is the number of free servers + the type of client next in line
nStates         = (n_servers+1)*n_diff_clients;

nActions        = 2; % our agent can reject(==0) or accept(==1)

% Initialize the things we will learn:
rho = 0.0;
Q   = zeros(nStates,nActions);

% the number of free servers ... we start with everyone of them free:
n_free = n_servers;

% get the first client type:
cIndx = sample_discrete( p_client, 1, 1 );
st    = [ n_free, cIndx ];
sti   = sub2ind( [n_servers+1,n_diff_clients], st(1)+1, st(2) );

for ei=1:MAX_N_EPISODES,

  % pick action to take using an epsilon greedy policy derived from Q:
  %
  if( n_free>0 )
    if( Q(sti,1)~=Q(sti,2) )          % Q's for all actions are not equal ... pick the
greedy option
      [dum,at] = max(Q(sti,:));
      if( rand<epsilon )              % explore ... with a random action
        tmp=randperm(nActions); at=tmp(1);
      end
    else                              % Q's for both actions are equal ... randomly pick
one.
      tmp=randperm(nActions); at=tmp(1);
    end
    at = at-1;                       % maps action to the range 0,1
  else                               % we have no space so we must reject
    at = 0;
```

```
  end
  ati = at+1;                       % the action INDEX

  % observe r and s':
  if( at==1 ) % accept
    rew = client_rew(cIndx);
  else        % reject
    rew = 0;
  end

  % get the next client in line:
  cIndxP = sample_discrete( p_client, 1, 1 );
  % compute how many servers will expire (finish their job) during this time step:
  n_busy = n_servers - n_free;
  if( n_busy > 0 )
    finished   = sample_discrete( [1-p,p], n_busy, 1 ); % 1 => not finshed; 2 => finished
    n_finished = sum( finished-1 );                     % subtract one to map "not"
finshed to zeros and one to "finished".
  else
    n_finished = 0;
  end; clear n_busy;
  n_freeP = n_free+n_finished;
  if( at==1 )                                           % ... record that we accepted a
new client
    n_freeP = n_freeP-1;
  end
  stp = [ n_freeP, cIndxP ];                            % compute the next state

  stpi = sub2ind( [n_servers+1,n_diff_clients], stp(1)+1, stp(2) );

  % get max(Q(t+1,:)):
  if( n_freeP > 0 )
    maxQP = max(Q(stpi,:));
  else
    maxQP = Q(stpi,1);
  end

  % update Q:
  Q(sti,ati) = Q(sti,ati) + alpha*( rew-rho + maxQP - Q(sti,ati) );

  % get max(Q(t,:)):
  if( n_free > 0 )
    maxQ = max(Q(sti,:));
  else
    maxQ = Q(sti,1);
  end

  % update rho:
  if( abs( Q(sti,ati)-maxQ ) < 1.0e-9 )
    rho = rho + beta*( rew-rho + maxQP - maxQ );
  end

  % update the current state:
  n_free = n_freeP; cIndx = cIndxP; st = stp; sti = stpi;

end % end for episode loop

[Qmax,Act] = max( Q, [], 2 ); Act=Act-1;

% after reshape, dimensions will now be [0,1,2,\cdots,10] x [worst cust,\cdots,best
```

```
cust.] =
%                                          [n_free servers] x [worst,best cust ranking]
Qmax = reshape( Qmax, [n_servers+1,n_diff_clients] );
Act  = reshape( Act, [n_servers+1,n_diff_clients] );

% order the output so it is [worst cust. \cdots best cust] x [0,1,2,\dots,10] ... as the
output from the book
Qmax = Qmax.';
Act  = Act.';

% -- delete the zero column corresponding to zero free servers:
Qmax(:,1) = []; Act(:,1) = [];
```