```
function [V] = rw_offline_tdl_learn(lambda,alpha,n_rw_states,nEpisodes)
% RW_OFFLINE_TDL_LEARN - Performs td(lambda) learning of the random walk example
%
% 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.
%
%-----

% the number of episodes (in general this would need to be much larger to gaureentee
complete learning):
%nEpisodes = 1000;

% the number of non terminal states plus two terminal states:
nStates=n_rw_states+2;

% initialize the nonterminal states ... terminal states are defined to be ZERO
%V = 0.5*ones(1,nStates); V(1)=0; V(end)=0;
V = zeros(1,nStates); V(1)=0; V(end)=0;

gamma     = 1.0;
%gamma     = 0.9;

for ei=1:nEpisodes,

  % perform a random walk getting the sequence of rewards recieved and the
  % sequence of states we transition through:
  %
  [rewseen,stateseen] = rw_episode(n_rw_states);

  % compute the lambda return R^{\lambda}_t for t=0,1,2,\cdots,T-1
  %
  % R^{\lambda}_t = (1-\lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} R_^{(n)}_t + \lambda^{T-
t-1} R_t ,
  %
  % by first computing R_t^{(n)} at each state visited (t=0,1,2,\cdots,T-1) and for
n=1,2,\cdots,T-t
  % (the n-step return for these t's and all possible n's).  Here
  %
  % R_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{n-1}
r_{t+n} +
  %             \gamma^n V_t(s_{t+n})
  %
  % and update V(\cdot):
  %
  T = length(rewseen); deltaV = zeros(1,nStates);
  for t=0:(T-1),    % <- STATE index ...
    Rtn = zeros(1,T-t);
    for n=1:(T-t),
      % get the rewards recieved from t+1 to t+n:
      n_rews    = min(n,T-t);
      rs        = rewseen( (t+1) : (t+n_rews) );
      % get the state value function at the state at t+n:
      st_tpn    = stateseen( t+1+n_rews );
```

```
      Vst_tpn   = V(st_tpn);
      gammaPowV = gamma.^[ 0:(n_rews-1) ];
      % compute this n-step reward:
      Rtn(n) = dot( rs, gammaPowV(1:n_rews) ) + (gamma^n_rews)*Vst_tpn;
    end

    % compute R^{\lambda}_t:
    %
    lambdaPow = lambda.^[ 0:(T-t-1) ];
    Rtl       = (1-lambda)*dot( lambdaPow(1:end-1), Rtn(1:end-1) ) +
lambdaPow(end)*Rtn(end);

    % this is an offline algorithm so save all V updates (to be applied when the episode
ends):
    st    = stateseen(t+1);
    deltaV(st) = deltaV(st) + alpha*( Rtl-V(st) );

  end % end state sequence loop ...

  % apply the update stored in deltaV accumulated from this episode:
  V = V + deltaV;

end % end episode loop ...

% remove the terminal states:
V = V(2:end-1);
```