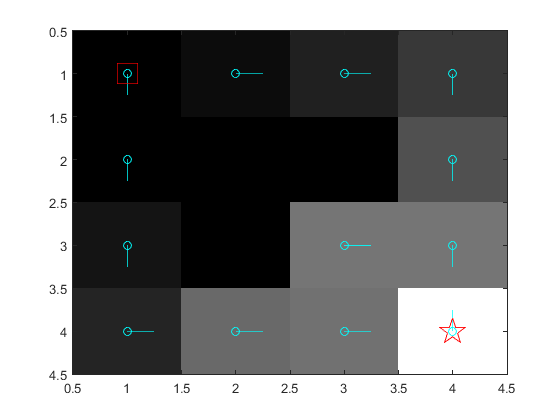
Ep = 0.4

Alpha = 0.2



……………………………………………………………………………………………………………………………………………………….

E =

Q learning:

function [v, pi] = qLearning(model, maxit, maxeps)

% initialize the value function

Q = zeros(model.stateCount, 4);

pi = ones(model.stateCount, 1);

Alpha =0.0001;

pi\_changing = ones(model.stateCount, 1);

for i = 1:maxeps,

% every time we reset the episode, start at the given startState

s = model.startState;

for j = 1:maxit,

% PICK AN ACTION

a = 1;

p = 0;

r = rand;

for s\_ = 1:model.stateCount,

p = p + model.P(s, s\_, a);

if r <= p,

break;

end

end

% s\_ should now be the next sampled state.

% IMPLEMENT THE UPDATE RULE FOR Q HERE.

a\_ = epsilon\_greedy\_policy(Q(s, :), j);

reward = model.R(s,a);

Q(s,a) = Q(s,a) + Alpha \* [ reward + model.gamma \* max( Q(s\_, a\_) ) - Q(s,a)];

s = s\_;

a = a\_;

% SHOULD WE BREAK OUT OF THE LOOP?

[~, max\_action] = max(Q(s,:));

pi\_changing(s) = max\_action;

v\_changing = Q( : ,max\_action);

% SHOULD WE BREAK OUT OF THE LOOP?

if s == model.goalState

break;

end

end

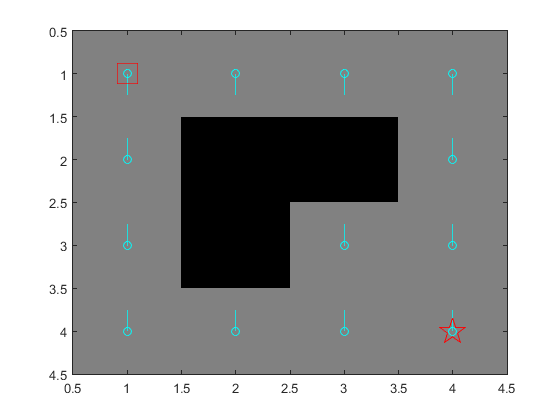
end

% REPLACE THESE

v = v\_changing;

pi = pi\_changing;

end



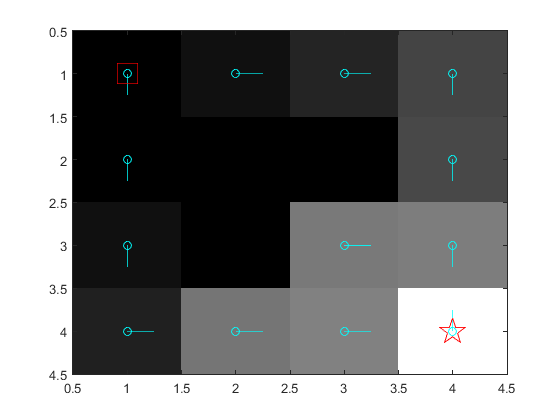
………………………………………………………………………………………………………………..

Changing value of Alpha….

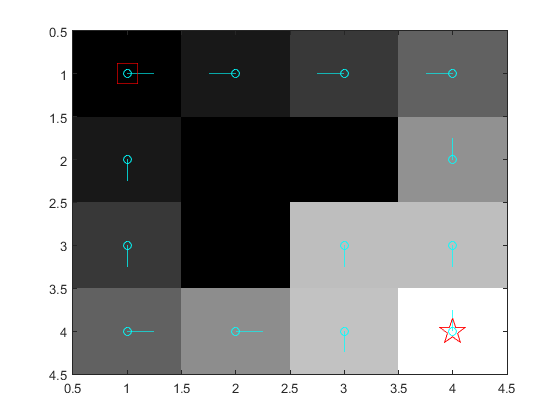
reward = model.R(s,a);

Alpha = 1/j ;

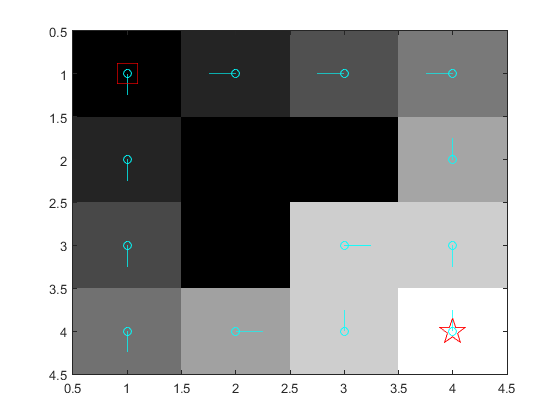
Q(s,a) = Q(s,a) + Alpha \* [ reward + model.gamma \* max( Q(s\_, a\_) ) - Q(s,a)];



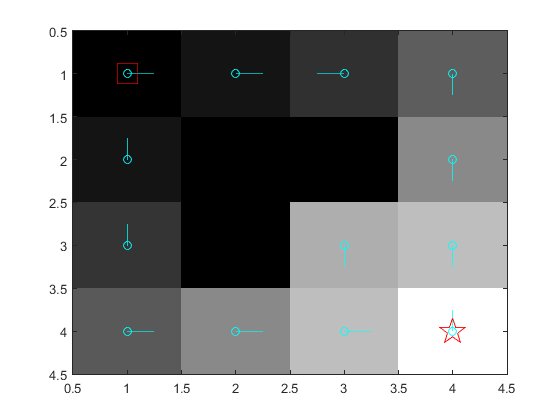
**SARSA**

Alpha = 0.001 & E = 0.0001

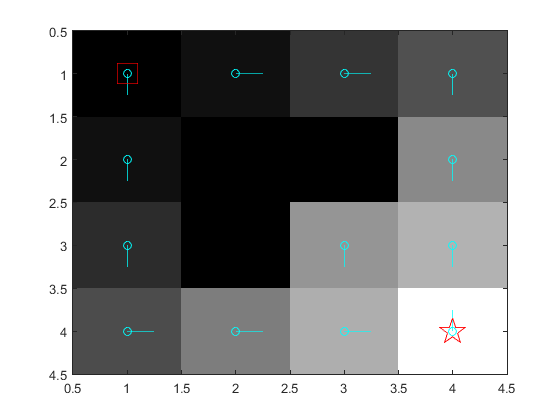
Alpha = 0.0001 & E = 0.0001



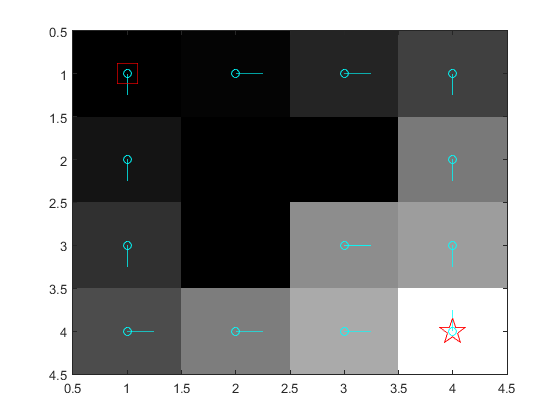
Alpha = 0.01 & E = 0.0001



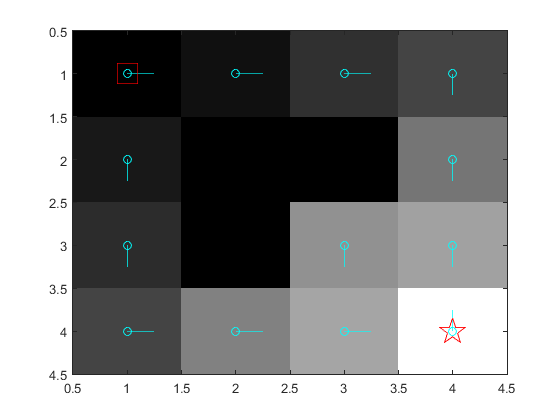
Alpha = 0.1 & E = 0.0001



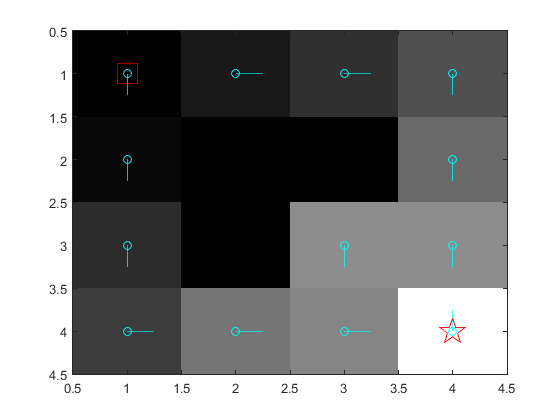
Alpha = 0.1 & E = 0.001



Alpha = 0.1 & E = 0.01



Alpha = 0.1 & E = 0.1



Q-Learning:

