Homework 8 – A*

Jasen Carroll

```
%% Function AStar_jcc329()
% map- matrix full of zeros denoting free space and ones denoting
%  obstacles.
% start- start point (row vector)
% qgoal- end point   (row vector)
% path- path from start to goal
% Takes in a map, start point, and goal point and returns a path from the
% start to the goal via the Wavefront planner algorithm.
function path = AStar_jcc329(start, qgoal,map)

% The hueristic function will be a function for ease of calculation.
% Pass it a point and it returns the heuristic at that point. (for 8-point
% adjancy)
heur = @(q) max(abs(qgoal - q));

% List of neighbor offsets. Add a row to a point to get the point's
% neighbor.
nei = [-1 -1;-1 0;-1 1;0 -1;0 1;1 -1;1 0;1 1]; %8-point

%Open list stores [pointX, pointY, (distance from start), (hueristic to goal
+ distance from start)]
open = [start(1) start(2) 0 heur([start(1) start(2)])];

%Negate the map
map = -map;

%size of the map for reference in back function
[mapn mapm] = size(map);

% The back function will be a cell. Each entry in the cell corresponds to
% an entry in the map. Each entry will store either nothing (ie the point
% is not in the closed list) or the previous point with the distance from
% the start.
% Closed list isn't actually used- it's easier to test the back function
% for existence. ie if isempty(back{1,1}) is true, that point has never
% been expanded.
% If you do not know how to use cells, you can either consult MATLAB help
% or substitute for your own solution. Essentially, each element in a cell
% is its own matrix.
back = cell(mapn,mapm);
back{start(1),start(2)} = -1; %flag the path start

while(~isempty(open))   %main loop

    %sort the open list in accordance with the actual distance to point
    %plus the heuristic of the point (column 4 of the list)
    open = sortrows(open,4);

    %There is a possibilty of repeats in the open list. Since the list is
    %sorted, we can remove the lesser-performing repeats with these lines.
    [~, I, ~] = unique(open(:,1:2),'rows','first');
```

```
    I = sort(I);
    open = open(I,:); %Now the open list is sorted and unique.

    imagesc(map) % use these two lines to view the progression of the
    drawnow;     % algorithm

    %% The rest of the loop is on you.

    % Pop the first point off the open list
    [p, open] = popOffStack(open);
    x = [p(1) p(2)];

    %Check to see if point x is the goal. If so, break.
    if x == qgoal
        disp('You have reached the goal!')
        break
    end

    % Assign this point in the map as its distance from the start point.
    map(x(1),x(2)) = p(3);

    % Expand current point x by adding it's valid neighbors to the open
    % list and back path. So, for each neighbor,
    for i=1:8
        n = x+nei(i,:);

        % Check to make sure the neighbor exists and is not an obstacle
        if ~(all(n >= 1) && n(1) <= mapn && n(2) <= mapm)
            continue;
        end
        if map(n(1),n(2)) == -1
            continue;
        end

        % Check to see if this neighbor has been added to the closed list.
        % (Does this point have an entry in the back path?)
        % If the point is not in the closed list, add it to Both the open
        % list and the back path (which doubles as the closed list.
        if isempty(back{n(1),n(2)})
            dist = p(3)+1; % new distance
            h = heur([n(1) n(2)]); % cost
            temp = [n(1) n(2) dist dist+h]; % new stack
            open = addToStack(temp, open); % add new stack
            back{n(1),n(2)} = x; % add to back
        end
    end
end

%% We're done the algorithm. Now we back-search for the start from the
%  goal.
if(isempty(back{qgoal(1),qgoal(2)})) %the goal has no back path; FAIL
    disp('Unreachable');
    path = -1;
else
    %The goal has a backpath; therefore, we can get to the start from the
```

```
    %goal.

    % Check the goal and add it's back path to a temporary variable. Repeat
    % process for that point and consecutive points until the start is
    % found.
    tmp = qgoal;
    while(back{tmp(end,1),tmp(end,2)} ~= -1)
        tmp(end+1,:) = back{tmp(end,1),tmp(end,2)}([1,2]);
    end

    %path is the tmp variable backwards
    path = tmp(end:-1:1,:);

    %Plot the path.
    imagesc(map); colormap jet;
    hold on;
    plot(path(:,2),path(:,1),'k','LineWidth',3);
    hold off;
end


end
```