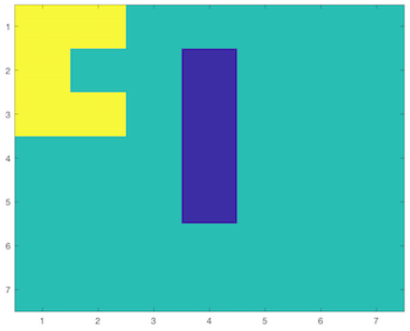
Creating a State

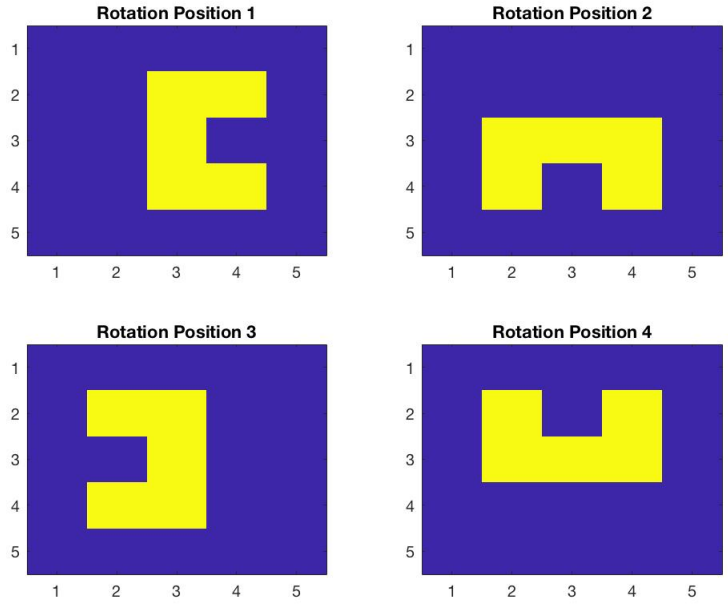
Your first task in this project, will be to write a function `MakeState.m` that creates a 2D grid representation of a current robot's state as the ones shown in the images be



Your function will take six input arguments:

1. The number of rows in the environment grid n.
2. The number of columns in the environment grid m.
3. A $d \times 2$ matrix containing $(row, column)$ locations of d wall cells in the grid.
4. A row position of the centroid of a robot (e.g. In the starting state $row = 2$, in the goal state $row = 6$).
5. A col (column) position of the centroid of a robot (e.g. In the starting state $col = 1$, in the goal state $col = 7$).
6. A scalar rot_idx that indicates, which of the four rotation positions the robot is currently in (see the figure below).

Four possible rotation positions of a robot:



Your function should then return an $n \times m$ dimensional matrix S where wall locations should have the value of -1, every grid occupied by the robot should have the value of 1, and every other cell in S should have a value of 0. Furthermore, your function should handle cases where the produced robot's state is not valid. For instance, if a robot's location is outside of the 2D grid, or if it overlaps with the walls, then the output S should contain ALL zero values.

Your Function

Save Reset MATLAB Documentation (https://www.mathworks.com/help/)

```
function S=MakeState(n,m,walls,cur_row,cur_col,rot_idx)
    % Create a current state
```

```
    % Create a current state
    %
    % Input:
    % - n: number of rows in the environment grid
    % - m: number of columns in the environment grid
    % - walls: d x 2 matrix containing (row,column) positions of the walls in the grid
    % - cur_row: the row position of a robot's centroid
    % - cur_col: the column position of a robot's centroid
    % - rot_idx: a scalar indicating which rotation position the robot is currently in
    % Output:
    % - S: n x m matrix storing the state: wall locations should have the values of -1, every position occupied
    % of 1, the rest of the gridd cells should have zero values. If the
    % provided state is not valid then all the values in S should have zero
    % values

    S=zeros(n,m);

    % Fill in The Wall values
    % convert d (sub2ind) to indices in matrix
    lin_idx_list = sub2ind([n m], walls(1:end,1), walls(1:end,2));
    S(lin_idx_list) = -1;

    % Find the locations in the grid occupied by a robot
    if rot_idx==1
      % get index list (total of 5)
      subs_to_add = [cur_row, cur_col; cur_row - 1, cur_col; ...
                     cur_row + 1, cur_col; cur_row - 1, cur_col+1;...
                     cur_row + 1, cur_col+1];
    elseif rot_idx==2
      subs_to_add = [cur_row, cur_col; cur_row, cur_col - 1; ...
                     cur_row, cur_col + 1; cur_row + 1, cur_col - 1;...
                     cur_row + 1, cur_col + 1];
    elseif rot_idx==3
      subs_to_add = [cur_row, cur_col; cur_row - 1, cur_col; ...
                     cur_row + 1, cur_col; cur_row - 1, cur_col - 1;...
                     cur_row + 1, cur_col - 1];
    elseif rot_idx==4
      subs_to_add = [cur_row, cur_col; cur_row, cur_col - 1; ...
                     cur_row, cur_col + 1; cur_row - 1, cur_col - 1;...
                     cur_row - 1, cur_col + 1];
    end


    pos_valid = true;
    % Check if the robot is fully within the boundaries of a grid
    pos_valid = subs_to_add(1:end,1) > 0 & subs_to_add(1:end,1) < n+1 & ...
                subs_to_add(1:end,2) > 0 & subs_to_add(1:end,2) < m+1;


    % Check if the robot's location doesn't overlap with the locations of the walls
    % use d matrix

    member_vec = ismember(subs_to_add,walls,'rows');
    if sum(member_vec,1) > 0
        pos_valid = false;
    end


    % If the state is not valid return S with all zero values,
    % otherwise return the state S filled in with the correct values
    if pos_valid
      lin_idx_list = sub2ind([n m], subs_to_add(1:end,1), subs_to_add(1:end,2));
      S(lin_idx_list) = 1;
    else
      S=zeros(n,m);
    end
    display(S);
end
```

```
70
```

Code to call your function

Reset

```
%% Create a starting state
rows=7; cols=7;
walls=[2 4; 3 4; 4 4; 5 4];
cur_row=2; cur_col=1; rot_idx=1;
start_S=MakeState(rows,cols,walls,cur_row,cur_col,rot_idx);
```

Run Function

Previous Assessment: All Tests Passed

Submit

- Is The Start State Computed Correctly?
- Is the Goal State Computed Correctly?
- Does the Function Handle Boundary Cases Correctly?
- Does the Function Handle Robot's Overlaps with the Walls Correctly?