MAZE GENERATION FINAL REPORT

Individual Matlab Project

Ryan Schwartz

ENGR 133, Team 001-07

Overview

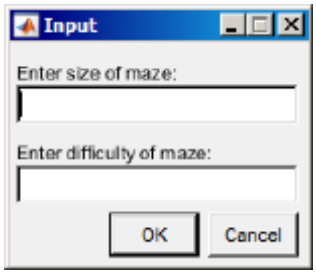
This program uses a randomized version of the depth-first search algorithm to generate a maze of size n and difficulty d. The `main` function controls the rest of this program, so this is the only function that needs to be called in order to generate a maze. Descriptions of the created functions are listed below.

Requirements

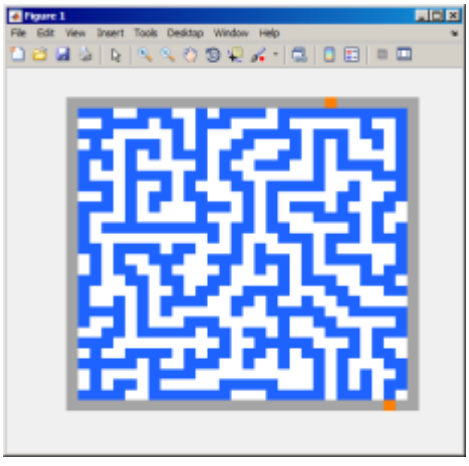
A reference to where each requirement can be found in the list below:

1. User input: `main` function lines 29 - 36
2. Functions: One `main` function with five subfunctions, `setup`, `move`, `validateMove`, `adjustEnd`, `checkNodes`, `dispMaze`
3. For loop: `validateMove` function lines 36 - 42
4. While loop: `main` function lines 32 - 36
5. Embedded loops: `validateMove` function lines 36 - 42
6. Matrix: `main` function lines 39 - 46
7. Conditional structure: `adjustEnd` function lines 27 - 37
8. 100 lines of code: Current code is about 300 lines without the function headers at the top

User Manual



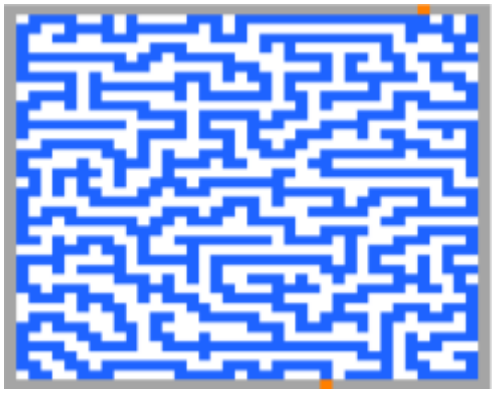
To use this program, simply navigate to the correct folder in `Current Folder`. Then, simply type `main` into the `Command Window` to begin. A window will pop up asking for a desired size and difficulty (as shown to the right). Size must be greater than 6, while difficulty must be between 1 and 10. After this is done, press OK to watch the maze be generated. Recommended values for the size of the maze is between 30 and 50, while the recommended values for the difficulty range is 5 for zero weighted directions.



For this example, a size of 30 and difficulty of 5 will be used. After pressing OK, watch as the maze is generated in real time. You can adjust the window size during this process. After the generation is complete, a message will pop up in the `Command Window` saying `Completed successfully`. At this point, feel free to solve the maze. The walls of the maze are white, while the path you follow is blue. An example maze can be seen to the right. Start at the bottom orange piece, and move your way around the maze to get to the top orange piece.

Screenshots

Size: 30 | Difficulty: 1



Size: 30 | Difficulty: 10

Interaction Diagram

Code Descriptions

main.m

Inputs: None
Outputs: None (Note: displays maze with `dispMaze` function)
This is the main function of the maze generation program. This uses the `setup` function to create an empty maze. Then, it uses the `move` function while there are no more nodes listed in the `nodes` list. The `adjustEnd` function is used to link the exit to the rest of the maze. Lastly, `dispMaze` function is called to format the maze to a viewable format.

setup.m

Inputs: size
Outputs: maze, nodes, startPos, endPos
This function takes a size parameter and generates a nxn matrix of that size. Next, the function creates a border of 8's around the outside edges. It then randomly generates a start and end point (which will be returned at the end of the function). The start point will be located at some point along the bottom row (where it will replace one of the 8's). The end point will be located at some point along the top row (where it will replace one of the 8's). A 1 (path) will be generated directly above the start point, and a node will be created at that point. The function then returns the created maze, a list of nodes, and the start/end positions of the maze.

move.m

Inputs: maze, position
Outputs: maze, position, nodes
This function takes a maze and generates one new point along its current path. This function is recursively called within the main function until there are zero nodes. This function first calls `validateMove` to get a list of possible directions the function can go. If there are 0 possible directions and the current position is a node, the node is removed and the current position is set to the last node on the list. If there are 0 possible directions and the current position is not a node, the current position is set to the last node on the list. If there are 1, 2, 3, or 4 possible directions, the function does a couple of things. It first finds a random direction using the weighting described in the movement section of this README. It then finds the position of the previous position (whatever point was created just before this one). It then calls the `checkNodes` function. If it returns `true`, then add the current point to the `node` list. Regardless, the `currentPoint` is set equal to the `futurePoint` and that point in the maze is set to 1. maze, pos, and nodes are all returned in order to make this function easy to be called recursively.

validateMove.m

Inputs: maze, position
Outputs: position
The purpose of this function is to tell the `move` function where the current position could possibly go. This function will return a matrix called `directions` that will return either a `1` or a `0` for each direction. The format for `directions` is `up, down, left, right`. Valid movements are checked using a valid movement matrix outlined below.

adjustEnd.m

Inputs: maze, endPos
Outputs: maze
This function connects the end piece to the rest of the maze. Because the movement constraints are set to not directly touch the border (8's), a connection must be made to the generated maze. This is done by traversing straight down from the end position until a connection has been made to the rest of the maze. This is indicated by looking left, right, and down to see if there is a 1 next to it. This outputs the adjusted maze at the end of this function.

checkNode.m

Credit to Bailey Keel for helping me come up with this idea.
Inputs: futurePoint, previousPoint
Outputs: result
This function checks to see if the movement would result in the creation of a node. Most simply, a node is created when there is a change in the direction of a path. In all cases, the generated point (`futurePoint`) will be in a different row and column than the last point (`previousPoint`, not `currentPoint`). If a node is found, returns `true`.

dispMaze.m

Inputs: maze
Outputs: None (Note: displays maze)
This function formats the matrix to be graphed via heatmap. This is a two dimensional graph in which there are different colors based on the value in the matrix. That way, the walls can be grey, paths blue, and start/end orange.

point.m

Point class
Properties: row, col
Methods: point(a, b), adjust(maze, position, value), mazeValue(maze, position, a, b), same(a, nodes)
The purpose of this class is to prevent the need for x and y components for all points

throughout the maze. These functions are mostly one liners whos purpose is to make adjustments to the maze and positions simpler.

Code

main

```
function [] = main()
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ENGR 132 Program Description
%       This function This is the main function of the maze generation
program.
%   This uses the setup function to create an empty maze. Then, it uses the
%   move function while there are no more nodes listed in the nodes list.
%   At the very end, it uses the adjustEnd function to link the exit to the
%   rest of the maze.
%
% Function Call
%   function [] = main()
%
% Input Arguments
%       1. None
%
% Output Arguments
%       1. None
%
% Assignment Information
%       Assignment:         MATLAB Individual Project
%       Author:             Ryan Schwartz, schwar95@purdue.edu
%       Team ID:            001-07
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% INITIALIZATION ---
% Grabs user input for size and difficulty
% Constraints for size: Positive number greater than 5
% Constraints for difficulty: 1 < d < 10
res = inputdlg({'Enter size of maze:', 'Enter difficulty of maze:'}, 'Input',
1);
size = str2num(res{1});
difficulty = str2num(res{2});
% Verify numbers are correct
while size <= 5 || difficulty < 1 || difficulty > 10
    res = inputdlg({'Enter size of maze:', 'Enter difficulty of maze:'},
'Input', 1);
    size = str2num(res{1});
    difficulty = str2num(res{2});
end

%% CALCULATIONS ---
[maze, nodes, position, endPoint] = setup(size);
% Runs generation process until there are no more nodes left
while numel(nodes) > 0
    [maze, position, nodes] = move(maze, position, nodes, difficulty);
    dispMaze(maze);
end
maze = adjustEnd(maze, endPoint);
```

```
%% FORMATTED TEXT & FIGURE DISPLAYS ---
dispMaze(maze);
fprintf('Completed successfully\n');

%% COMMAND WINDOW OUTPUTS ---


%% ACADEMIC INTEGRITY STATEMENT ---
% I/We have not used source code obtained from any other unauthorized
% source, either modified or unmodified.  Neither have I/we provided
% access to my/our code to another. The project I/we am/are submitting
% is my/our own original work.
%
```

setup

```
function [maze, nodes, position, endPos] = setup(size)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ENGR 132 Program Description
%       This function This function takes a size parameter and generates a nxn
%   matrix of that size. Next, the function creates a border of 8's around
%   the outside edges. It then randomly generates a start and end point
%   (which will be returned at the end of the function). The start point
%   will be located at some point along the bottom row (where it will
%   replace one of the 8's). The end point will be located at some point
%   along the top row (where it will replace one of the 8's). A 1 (path)
%   will be generated directly above the start point, and a node will be
%   created at that point. The function then returns the created maze, a
%   list of nodes, and the start/end positions of the maze.
%
% Function Call
%   function [maze, nodes, position, endPos] = setup(size)
%
% Input Arguments
%       1. size: User-defined maze size
%
% Output Arguments
%       1. maze: Blank maze template of size x size dimensions
%   2. nodes: Initial list of nodes
%   3. position: Initial position
%   4. endPos: Position of ending block
%
% Assignment Information
%       Assignment:         MATLAB Individual Project
%       Author:             Ryan Schwartz, schwar95@purdue.edu
%       Team ID:            001-07
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% INITIALIZATION ---
% Generate maze, create borders of 8s
maze = zeros(size);
maze(1, 1:end) = 8;
maze(1:end, 1) = 8;
maze(end, 1:end) = 8;
maze(1:end, end) = 8;

%% CALCULATIONS ---
```

```
% Generate random starting and ending points
position = point(size, randi([2 (size - 1)]));
maze = setMazePosition(maze, position, 3);
position = adjust(position, -1, 0);
maze = setMazePosition(maze, position, 1);
nodes(1, 1) = position.row;
nodes(2, 1) = position.col;

endPos = point(1, randi([2 (size - 1)]));
maze = setMazePosition(maze, endPos, 4);

%% FORMATTED TEXT & FIGURE DISPLAYS ---


%% COMMAND WINDOW OUTPUTS ---


%% ACADEMIC INTEGRITY STATEMENT ---
% I/We have not used source code obtained from any other unauthorized
% source, either modified or unmodified.  Neither have I/we provided
% access to my/our code to another. The project I/we am/are submitting
% is my/our own original work.
%
```

move

```
function [maze, position, nodes] = move(maze, position, nodes, difficulty)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ENGR 132 Program Description
%       This function his function takes a maze and generates one new point
along its current path. This function is
%   recursively called within the main function until there are zero nodes.
This function first calls validateMove to
%   get a list of possible directions the function can go. If there are 0
possible directions and the current position
%   is a node, the node is removed and the current position is set to the
last node on the list. If there are 0 possible
%   directions and the current position is not a node, the current position
is set to the last node on the list. If
%   there are 1, 2, 3, or 4 possible directions, the function does a couple
of things. It first finds a random direction
%   using the weighting described in the movement section of this README. It
then finds the position of the previous
%   position (whatever point was created just before this one). It then calls
the checkNodes function. If it returns
%   true, then add the current point to the node list. Regardless, the
currentPoint is set equal to the futurePoint and
%   that point in the maze is set to 1. maze, pos, and nodes are all returned
in order to make this function easy to be
%   called recursively.
%
% Function Call
%   function [maze, position, nodes] = move(maze, position)
%
% Input Arguments
%       1. maze: Current maze
%   2. position: Current position
```

```
%   3. nodes: Current list of nodes
%   4. difficulty: User-defined difficulty (1 low, 10 high)
%
% Output Arguments
%       1. maze: Updated maze
%   2. position: Updated position
%   3. nodes: Updated list of nodes
%
% Assignment Information
%       Assignment:         MATLAB Individual Project
%       Author:             Ryan Schwartz, schwar95@purdue.edu
%       Team ID:            001-07
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% INITIALIZATION ---
% directions = [up, down, left, right]
% up, down, left, right can be 1 for OK or 0 for NO
[directions] = validateMove(maze, position);

%% CALCULATIONS ---
% Check if that route can continue or not
if any(directions) == 0
    if same(position, nodes) == 1
        % Remove last node because all positions are exhausted
        nodes = nodes(:, 1 : end - 1);
    else
        position = point(nodes(1, end), nodes(2, end));
    end
else
    % sum(directions) = 1, 2, 3
    % Random direction
    locations = directions .* floor(100.0/sum(directions));
    % Adusts difficulty
    % Increase in upward tendency if difficulty is 1
    locations(1) = locations(1) * (1 - ((difficulty - 5.0)/14));
    locations(2) = locations(2) * (1 + ((difficulty - 5.0)/14));
    locations(3) = locations(3) * (1 + ((difficulty - 5.0)/14));
    locations(4) = locations(4) * (1 + ((difficulty - 5.0)/14));
    if sum(locations) ~= 100
        for k = 1:4
            if directions(k) == 1
                locations(k) = locations(k) + 100 - sum(locations);
            end
        end
    end
    % Choose random direction
    r = randi([1 100]);
    if r <= locations(1)
        futurePosition = point(position.row - 1, position.col);
    elseif r <= locations(2) + locations(1)
        futurePosition = point(position.row + 1, position.col);
    elseif r <= locations(3) + locations(2) + locations(1)
        futurePosition = point(position.row, position.col - 1);
    else
        futurePosition = point(position.row, position.col + 1);
    end
```

```
    % Find previous number 1
    if mazeValue(maze, position, -1, 0) == 1
        previousPosition = point(position.row - 1, position.col);
    elseif mazeValue(maze, position, 1, 0) == 1
        previousPosition = point(position.row + 1, position.col);
    elseif mazeValue(maze, position, 0, -1) == 1
        previousPosition = point(position.row , position.col - 1);
    else % mazeValue(maze, position, 0, 1) == 1
        previousPosition = point(position.row, position.col + 1);
    end

    % Check if node created
    if checkNode(futurePosition, previousPosition) == 1
        nodes(1, end + 1) = position.row;
        nodes(2, end) = position.col;
    end

    position = futurePosition;
    maze = setMazePosition(maze, position, 1);
end

%% FORMATTED TEXT & FIGURE DISPLAYS ---


%% COMMAND WINDOW OUTPUTS ---


%% ACADEMIC INTEGRITY STATEMENT ---
% I/We have not used source code obtained from any other unauthorized
% source, either modified or unmodified.  Neither have I/we provided
% access to my/our code to another. The project I/we am/are submitting
% is my/our own original work.
%
```

validateMove

```
function [positions] = validateMove(maze, position)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ENGR 132 Program Description
%       This function telsl the move function where the current position could
%   possibly go. This function will return a matrix called directions that
%   will return either a 1 or a 0 for each direction. The format for
%   directions is up, down, left, right. Valid movements are checked using
%   a valid movement matrix outlined on GitHub.
%
% Function Call
%   function [positions] = validateMove(maze, position)
%
% Input Arguments
%       1. maze: Current maze
%   2. position: Current position
%
% Output Arguments
%       1. positions: Matrix that shows what available directions can be used.
%       Formatted in [up down left right], where 1's are viable movements.
%
```

```
% Assignment Information
%       Assignment:         MATLAB Individual Project
%       Author:             Ryan Schwartz, schwar95@purdue.edu
%       Team ID:            001-07
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% INITIALIZATION ---
positions = [1 1 1 1]; % Up, down, left, right
[numrow, numcol] = size(maze);

%% CALCULATIONS ---
% Check up
if position.row == 2
    positions(1) = 0;
elseif position.row == 3
    for z = [-1 0 1]
        if mazeValue(maze, position, -1, z) == 1
            positions(1) = 0;
        end
    end
else
    for k = [-2 -1]
        for z = [-1 0 1]
            if mazeValue(maze, position, k, z) == 1
                positions(1) = 0;
            end
        end
    end
end

% Check down
if position.row == numrow - 1
    positions(2) = 0;
elseif position.row == numrow - 2
    for z = [-1 0 1]
        if mazeValue(maze, position, 1, z) == 1
            positions(2) = 0;
        end
    end
else
    for k = [1 2]
        for z = [-1 0 1]
            if mazeValue(maze, position, k, z) == 1
                positions(2) = 0;
            end
        end
    end
end

% Check left
if position.col == 2
    positions(3) = 0;
elseif position.col == 3
    for z = [-1 0 1]
        if mazeValue(maze, position, z, -1) ==1
            positions(3) = 0;
        end
```

```
    end
else
    for k = [-1 0 1]
        for z = [-2 -1]
            if mazeValue(maze, position, k, z) == 1
                positions(3) = 0;
            end
        end
    end
end

% Check right
if position.col == numcol - 1
    positions(4) = 0;
elseif position.col == numcol - 2
    for z = [-1 0 1]
        if mazeValue(maze, position, z, 1) == 1
            positions(4) = 0;
        end
    end
else
    for k = [-1 0 1]
        for z = [1 2]
            if mazeValue(maze, position, k, z) == 1
                positions(4) = 0;
            end
        end
    end
end

%% FORMATTED TEXT & FIGURE DISPLAYS ---

%% COMMAND WINDOW OUTPUTS ---


%% ACADEMIC INTEGRITY STATEMENT ---
% I/We have not used source code obtained from any other unauthorized
% source, either modified or unmodified.  Neither have I/we provided
% access to my/our code to another. The project I/we am/are submitting
% is my/our own original work.
%
```

adjustEnd

```
function [maze] = adjustEnd(maze, endPosition)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ENGR 132 Program Description
%       This function This function connects the end piece to the rest of the
%   maze. Because the movement constraints are set to not directly touch
%   the border (8's), a connection must be made to the generated maze. This
%   is done by traversing straight down from the end position until a
%   connection has been made to the rest of the maze. This is indicated by
%   looking left, right, and down to see if there is a 1 next to it. This
%   outputs the adjusted maze at the end of this function.
%
% Function Call
```

```
%   function [maze] = adjustEnd(maze, endPosition)
%
% Input Arguments
%       1. maze: Current maze
%   2. endPosition: (point class) Position of end point
%
% Output Arguments
%       1. maze: Current maze
%
% Assignment Information
%       Assignment:         MATLAB Individual Project
%       Author:             Ryan Schwartz, schwar95@purdue.edu
%       Team ID:            001-07
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% INITIALIZATION ---
% Set position below end point to be a 1 (start the connection process)
position = point(endPosition.row + 1, endPosition.col);
maze = setMazePosition(maze, position, 1);
connection = 0;
while connection == 0
    % Search if there are any 1's in down, left, right positions
    if mazeValue(maze, position, 1, 0) == 1
        connection = 1;
    elseif mazeValue(maze, position, 0, -1) == 1
        connection = 1;
    elseif mazeValue(maze, position, 0, 1) == 1
        connection = 1;
    else
        % If there are no surrounding 1's, move down
        position = adjust(position, 1, 0);
        maze = setMazePosition(maze, position, 1);
    end
end

%% CALCULATIONS ---


%% FORMATTED TEXT & FIGURE DISPLAYS ---


%% COMMAND WINDOW OUTPUTS ---


%% ACADEMIC INTEGRITY STATEMENT ---
% I/We have not used source code obtained from any other unauthorized
% source, either modified or unmodified.  Neither have I/we provided
% access to my/our code to another. The project I/we am/are submitting
% is my/our own original work.
%
```

checkNode

```
function [result] = checkNode(fPosition, pPosition)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ENGR 132 Program Description
```

```
%       This function This function checks to see if the movement would result
%   in the creation of a node. Most simply, a node is created when there is
%   a change in the direction of a path. In all cases, the generated point
%   (futurePoint) will be in a different row and column than the last point
%   (previousPoint, not currentPoint). If a node is found, returns true.
%
% Function Call
%   function [result] = checkNode(fPosition, pPosition)
%
% Input Arguments
%       1. fPosition: Future position the maze is thinking about going to
%   2. pPosition: Previous position the maze was just at
%
% Output Arguments
%       1. result: 1 or 0 based on if there is a node or not (changed both row
%       and column)
%
% Assignment Information
%       Assignment:         MATLAB Individual Project
%       Author:             Ryan Schwartz, schwar95@purdue.edu
%       Team ID:            001-07
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% INITIALIZATION ---


%% CALCULATIONS ---
if fPosition.row ~= pPosition.row && fPosition.col ~= pPosition.col
    result = 1;
else
    result = 0;
end

%% FORMATTED TEXT & FIGURE DISPLAYS ---


%% COMMAND WINDOW OUTPUTS ---


%% ACADEMIC INTEGRITY STATEMENT ---
% I/We have not used source code obtained from any other unauthorized
% source, either modified or unmodified.  Neither have I/we provided
% access to my/our code to another. The project I/we am/are submitting
% is my/our own original work.
%
```

dispMaze

```
function [] = dispMaze(maze)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ENGR 132 Program Description
%       This function This function formats the matrix to be graphed via
%   heatmap. This is a two dimensional graph in which there are different
%   colors based on the value in the matrix. That way, the walls can be
%   grey, paths blue, and start/end orange.
%
```

```
% Function Call
%   function [] = dispMaze(maze)
%
% Input Arguments
%       1. maze: Current maze
%
% Output Arguments
%       1. None
%
% Assignment Information
%       Assignment:         MATLAB Individual Project
%       Author:             Ryan Schwartz, schwar95@purdue.edu
%       Team ID:            001-07
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% INITIALIZATION ---
cmap = [1 1 1; .12 .39 1; 1 1 1; 1 .5 0; 1 .5 0; 1 1 1; 1 1 1; .65 .65 .65];

%% CALCULATIONS ---
hmo = imagesc(maze);
colormap(cmap);
set(gca, 'XColor', 'none');
set(gca, 'Ycolor', 'none');
drawnow

%% FORMATTED TEXT & FIGURE DISPLAYS ---


%% COMMAND WINDOW OUTPUTS ---


%% ACADEMIC INTEGRITY STATEMENT ---
% I/We have not used source code obtained from any other unauthorized
% source, either modified or unmodified.  Neither have I/we provided
% access to my/our code to another. The project I/we am/are submitting
% is my/our own original work.
%
```

point

```
classdef point
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ENGR 132 Program Description
%       The purpose of this class is to prevent the need for x and y
components
%   for all points throughout the maze. These functions are mostly one
%   liners whos purpose is to make adjustments to the maze and positions
%   simpler.
%
% Assignment Information
%       Assignment:         MATLAB Individual Project
%       Author:             Ryan Schwartz, schwar95@purdue.edu
%       Team ID:            001-07
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    properties
        row
```

```
        col
    end
    methods
        % Constructor
        function [obj] = point(a, b)
            obj.row = a;
            obj.col = b;
        end

        % Functions
        function [maze] = setMazePosition(maze, position, value)
            maze(position.row, position.col) = value;
        end
        function [positionNew] = adjust(position, row1, col1)
            positionNew = point(position.row + row1, position.col + col1);
        end
        function [val] = mazeValue(maze, position, a, b)
            val = maze(position.row + a, position.col + b);
        end
        function t = same(a, nodes)
            if a.row == nodes(1, end) && a.col == nodes(2, end)
                t = 1;
            else
                t = 0;
            end
        end
    end
end
```