ENGR 151

Fall 2022

Lab 11

Files to turn in

This assignment is due November 29th, 2022 at 11:59 PM EST

Please turn in following files on Canvas under Lab 11 assignment:

- maze_solver.m
- maze_solver.txt
- num_islands.m (optional)
- num_islands.txt (optional)

Exercise 1: Maze Solver

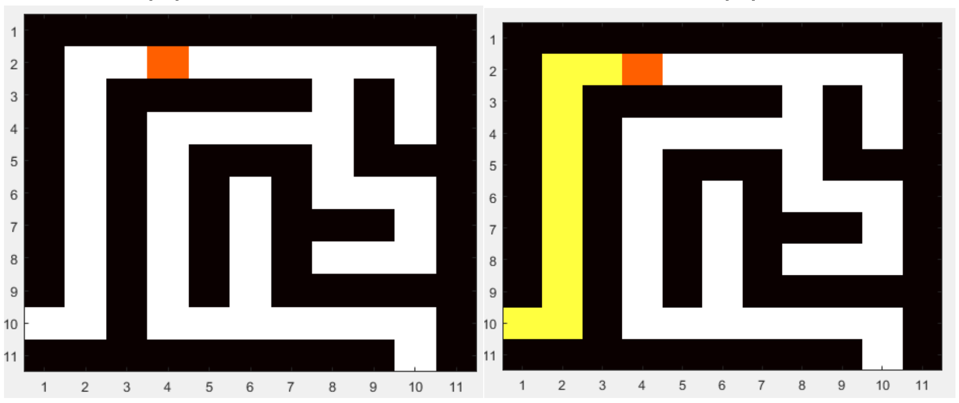
This exercise is intended to be algorithmically interesting, but not stressful. It is challenging and open-ended for you to think of your own solution. However, we will be evaluating your submission based on a thoughtful strategy and a whole-hearted attempt at executing your strategy, rather than perfectly working code. You will submit

1. **maze_solver.txt** with a description of your algorithm for solving the problem and
2. **maze_solver.m** in which you implement the described strategy.

The text file should explain your algorithm and why it solves the problem. Thorough but concise is good - no essays please! Have fun!

You get to simulate a maze-solving robot! There is some starter code available to you, which is explained below. Using it, you can visualize the status of the robot in the maze as shown below.

(A) **(B)**



"medium" maze with (A) robot only and (B) robot and markers at locations where robot has visited

In your program, the robot should start at the given start location for that maze and move until it reaches an exit. The only requirement is that the robot does not move randomly and is not hard-coded to only solve the example mazes given.

You are given 3 files to use:

- `make_map.m`
- `plotter.m`
- `starter_code.m`

`make_map.m`

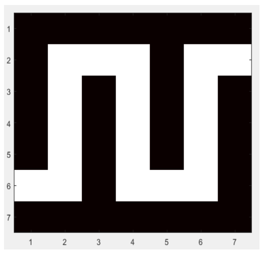
This is a function file. Use this function to create a matrix that represents the map of the maze. There are 4 pre-created mazes which you can use. **The function takes a string input and returns a matrix**. The string input indicates which maze you want to use. There are 4 options named: "easy", "medium", "hard", and "multi-exit". The matrix that is returned has 1's to mark where the white path is and 0's to mark the boundaries.

Below is the matrix that would be created with the following function call (and the corresponding map that it represents):

```
mazemap = make_map(“easy”);
```

```
mazemap =

     0     0     0     0     0     0     0
     0     1     1     1     0     1     1
     0     1     0     1     0     1     0
     0     1     0     1     0     1     0
     0     1     0     1     0     1     0
     1     1     0     1     1     1     0
     0     0     0     0     0     0     0
```



plotter.m

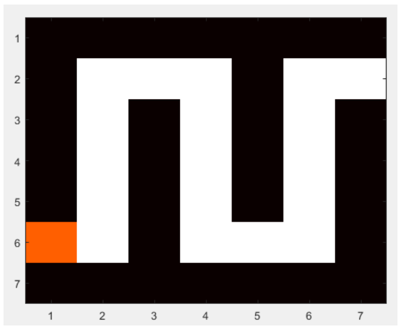
This is another function file. Use this function to plot the maze and visualize the robot in the maze. This is the definition of the `plotter` function:

```
function [] = plotter(maze_name, robot_row, robot_column,
                      robot_color, markers, marker_color)
```

- The input `maze_name` is a string with the name of the maze you want to use (either “easy”, “medium”, “hard”, or “multi-exit”)
- `robot_row`, `robot_column`, and `robot_color` will be used to draw the robot in the maze. A square will be plotted at position (`robot_row`, `robot_column`) of the color `robot_color`. The color should be any number in the range (0,1) (note: range not inclusive or the robot will blend in with the maze)
- There is also the option to put markers at any spot on the path. This is available to help you visualize aspects of your algorithm if that is helpful. The input `markers` is a matrix with the same dimensions as the mazemap. It starts as all zeros, and you can add a 1 at every location you want to draw a marker. The marker will be drawn with the color specified in the input `marker_color`.The color should be any number in the range (0,1) (again, range is not inclusive)

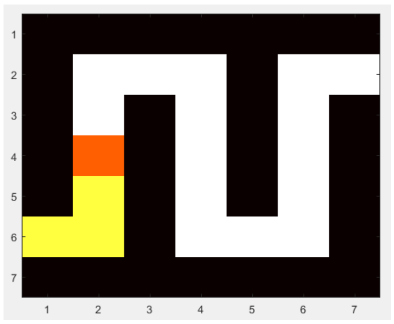
  The graph below would be generated with the code:

  ```
  [maze_rows, maze_cols] = size(make_map(“easy”));
  markers = zeros(maze_rows, maze_cols);
  plotter(“easy”, 6, 1, .5, markers, .8);
  ```



Adding the following code would produce a graph like the one below:

```
markers(6,1) = 1;
markers(6,2) = 1;
markers(5,2) = 1;
plotter("easy", 4, 2, .5, markers, .8);
```



`starter_code.m`

This is a script to help you get started. It should be well commented so you can understand what's going on. It includes examples of how to use the different functions. You can add your code to this file, but make sure you understand what is going on here before adding to it.

A few more details you'll need before starting...

Start position

These are the starting locations for each map:

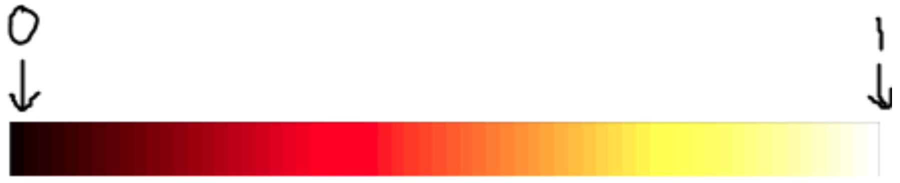
- "easy" map → (6,1)
- "medium" map → (10,1)
- "hard" map → (1,4)
- "multi-exit" map → (4,1)

End condition

You will know the robot is at an exit of the maze if either the row or column equals "1" or "end" and it is *not* at the same place as the start location.

Testing
You may want to use the `pause()` function between calls to the plotter function so you can see the robot as it moves. Also, the levels of difficulty of the maps are to help your testing to identify problems in your code based on how the robot behaves.

Colors
The map is plotted with the colormap "hot" shown below. You can make the robot and marker any color except 0 or 1 (since 0 and 1 are used for the boundary and path, respectively)



A potential place to start...
If you're having trouble with where to start, 2 suggestions are:
- Write a function that identifies the possible moves a robot could take given its current location (i.e. sides where there is no boundary).
- Draw out or talk to someone about your ideas as you come up with an algorithm for solving the maze.

Path Tracking
You should keep track of a path as an array of robot coordinates. After you have solved the maze, you should be able to loop through this array to show the robot moving from start to finish. (Note that you can adjust the path as you go along so that the replay does not include all the sidetracks the robot goes down while solving the maze)

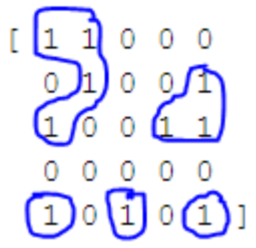
Challenge
Feeling up for an extra challenge? You can expand your code to find the exit with the shortest path from the start. You can use the "multi-exit" map to test this or edit other maps by adding extra exits.

Challenge Problem (optional): Number of Islands

This problem can be surprisingly similar to the previous problem if done recursively. Write a function to find how many islands there are on a map. The map will be given as a boolean matrix where 1 is land and 0 is water. An "island" is defined as a group of connected 1's. Two elements are said to be connected if each element is one of the 8 surrounding elements of the other.

For example, in the matrix below, there are 5 islands:

```
[ 1 1 0 0 0
  0 1 0 0 1
  1 0 0 1 1
  0 0 0 0 0
  1 0 1 0 1 ]
```



Your 2 files for this part should be named **num_islands.txt** and **num_islands.m**.