Zeid Kootbally
University of Maryland
College Park, MD

zeidk@umd.edu

RWA-2: An a-maze-ing problem.

ENPM809Y : Fall 2020
Due Thursday, October 15, 2020 by 7 pm

Contents

Problem

- The maze problem is a pretty common exercise in programming. This assignment was inspired from https://www.cs.bu.edu/teaching/alg/maze/.
- Part of this assignment can be re-used for the final project.
- A robot is asked to navigate a maze. It starts at a specific position in the maze (the starting position or `S`) and is asked to try to reach another position in the maze (the goal position or `G`).
- Positions in the maze will either be open or blocked with an obstacle.
- Positions are identified by (x, y) coordinates.

Robot

- At any given moment, the robot can only move 1 step in one of 4 directions.
- Valid moves are:
  - Go North
  - Go East
  - Go South
  - Go West
- The robot can only move to positions without obstacles and must stay within the maze.
- The robot should search for a path from `S` to `G` (a solution path) until it finds one or until it exhausts all possibilities.
- In addition, it should mark the path it finds (if any) in the maze.

The Maze

The maze used in the assignment has a predefined size (6×6) and a predefined design.
The maze is represented by a matrix of characters (C++ `char`), as depicted below.

```
5  # # # # # # # #
4  # . . # . . . #
3  # . # . G # . .
2  # . # # . # . #
1  . S # . . # . .
0  # . . . # # . #
   0 1 2 3 4 5 6 7
```

Maze

- Coordinates for the maze are represented in the Cartesian coordinate system.
  - As an example, the start position `S` is at $(1, 1)$.
  - As an example, the goal position `G` is at $(4, 3)$.
- `#` characters represent walls. The robot cannot be at the same location as walls or go through walls.
- `.` characters represent open positions, where the robot can be.
- A solution or partial path in the maze can be marked by the `+` symbol.

Algorithm

This problem must be solved (finding and marking a solution) using a depth-first search algorithm with recursion.
Remember that a recursive algorithm has at least two parts:

- Base case(s) to tell the algorithm when to stop.
- Recursive part that calls the same function (i.e., itself) to assist in solving the problem.

Recursive Part

- From the start position `S`, move in one of the four directions. Your algorithm **has** to search in this order: 1) North, 2) East, 3) South, and 4) West.
- From the new position, move into one of the four directions.
- Repeat this behavior until one of the base cases is reached.
- The prototype of the recursive function is:

```
bool FindPath(int x, int y);
```

- To find a path from the start position `S`(1,1) to the goal position `G`(4,3), we can just call the function `FindPath` as follows:

```
FindPath(1,2);//--cell North of S(1,1)
FindPath(1,3);//--cell North of S(1,2)
FindPath(1,4);//--cell North of S(1,3)
...
```

Base Cases

- It is not enough to know how to use `FindPath` recursively to advance through the maze.
- We also need to determine when `FindPath` must stop.
- The algorithm stops when any of the following conditions is encountered:
  - When the goal is reached.
  - `FindPath` returns `false` if the computed position is outside the boundaries of the maze.
  - `FindPath` returns `false` if the computed position is an obstacle.

Pseudocode

```
Function FindPath(int x, int y):
    if (x,y) outside of the maze then
        return false
    end
    if (x,y) is goal then
        return true
    end
    if (x,y) is obstacle then
        return false
    end
    Mark (x,y) with + char as part of the solution path
    if FindPath(north of x,y) is true then
        return true
    end
    if FindPath(east of x,y) is true then
        return true
    end
    if FindPath(south of x,y) is true then
        return true
    end
    if FindPath(west of x,y) is true then
        return true
    end
    Unmark (x,y) with . char as part of the solution path
    return false
```

Backtracking

An important capability that the recursive parts of the algorithm will give us is the ability to backtrack.
Suppose the algorithm just marked position x=2, y=4 as follows:

```
5  # # # # # # # #
4  # + + # . . . #
3  # + # . G # . .
2  # + # # . # . #
1  . S # . . # . .
0  # . . . # # . #
   0 1 2 3 4 5 6 7
```

The following steps are performed by the algorithm.

- Look for a path **North** of (2,4), calling `FindPath(2,5)`.
  - Position (2,5) is not open, the call to `FindPath(2,5)` will return `false`, and then it will go back (backtrack) to `FindPath(2,4)`) and resume at the step just after it went North.
- Next, it will look for a path **East** of (2,4), calling `FindPath(3,4)`.
  - Position (3,4) is blocked, the algorithm will backtrack to `FindPath(2,4)`) and resume at the step just after it went East.
- Next, it will look for a path **South** of (2,4), calling `FindPath(2,3)`.
  - Position (2,3) is blocked, the algorithm will backtrack to `FindPath(2,4)`) and resume at the step just after it went South.
- Next, it will look for a path **West** of (2,4), calling `FindPath(1,4)`.

- Position (1,4) is not open (occupied with `+`), the algorithm will backtrack to `FindPath(2,4)`) and resume at the step just after it went West.
  - Since West is the last direction to search from (2,4), it will unmark (2,4), and backtrack to the previous call, `FindPath(1,4)`.

Assignment Instructions

Instructions

- Individual assignment.
- You can hardcode the maze in your program using a data structure. You will need to do some research on how to code a 2D data structure (matrix).
- Prompt the user to enter the coordinates for the start position `S (x,y)` and the goal position `G (x,y)` (they should be `int`).
  - Check that `S` and `G` are not outside the maze nor placed where an obstacle is located.
  - If any of these two cases is encountered, re-prompt the user to enter a new location for either `S` or `G` or both (do-while loop?).
- Implement and call the recursive function `FindPath`.
  - If no path is found, display a message "Path not found". Try this by adding a `#` at location (4,2).
  - If a path is found, display the maze with the solution path (from `S` to `G`).
- Do not display the maze every time your function is called. The maze should be displayed only at the end.
- Code should be documented (Doxygen).
- Upload zip file which contains: 1) Doxyfile, 2) cpp file(s) with your code.
- Name zip file as follows: rwa2-lastname.zip.