Zeid Kootbally
University of Maryland
College Park, MD

zeidk@umd.edu

RWA-2 on Lecture 5: Functions

ENPM809Y : Fall 2019
Due Tuesday, October 8, 2019

Contents

Problem

- A robot is asked to navigate a maze. It starts at a specific position in the maze (the starting position) and is asked to try to reach another position in the maze (the goal position).
- Positions in the maze will either be open or blocked with an obstacle.
- Positions are identified by (x, y) coordinates.

Robot Motion

- 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 the start position to the goal position (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 and a predefined design. The maze is represented by a matrix of characters, as depicted below, and can be found in the text file (maze.txt).

```
30  ##############################################
29  #              #                             #
28  #  ##########  #############  #######  ####  #
27  #           #                 #        #     #
26  #  #######  #############  ###################
25  #  #     #  #           #  #                 #
24  #  #  #  #  #  #  #######  #  #############  #
23  #  #  #  #  #  #  #     #  #  #  #        #  #
22  #  #  ####  #  ##########  #  #  ####  #  #  #
21  #  #     #  #              #           #  #  #
20  #  ####  #  ################  #############  #
19  #     #  #                    #              #
18  ####  #  ######################  ##########  #
17  #  #  #                    #     #     #  #  #
16  #  #  ####  #############  #  ####  #  #  #  #
15  #  #  #     #     #     #  #  #     #     #  #
14  #  #  #  #######  ####  #  #  #  ##########  #
13  #     #  #              #  #  # G            #
12  ####  #  #  ##########  #  #  # +#############
11  #  #  #  #  #  #        #  #  # +#           #
10  #  #  #  #  #  #######  #  #  # +#  #######  #
9   #  #  #  #  #  #     #  #  #  # +#  #     #  #
8   #  #  #  #  #  #######  #  #  # +#  #  #  #  #
7   #  #  #  #  #           #  #  # +#  #  #  #  #
6   #  #  #  #  #############  #  # +#  ####  #  #
5   #     #  #                 #  # +#        #  #
4   #  ####  #  ################  # +#######  #  #
3   #  #     #  #        #        # ++++  #   #  #
2   #  ####  #  #######  #  ########## + ####### #
1   #        #           #  #    S++++++         #
0   ######################  ######################
    0123456789111111111122222222223333333333444444
              0123456789012345678901234567890123456
```

Representation

- Coordinates for the maze are represented in the Cartesian coordinate system.
  - The character `S` is positioned at $(29, 1)$.
  - The character `G` is positioned at $(32, 13)$.
- The ASCII character `#` represents a wall (forbidden position for the robot).
- Empty characters represent a position where the robot can be.
- The character `S` is the start position of the robot.
- The character `G` is the goal position the robot must reach (if a valid path is available).
- 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) with recursion.
Remember that a recursive algorithm has at least two parts:

- Base case(s) that determine 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 (North, East, South, 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`$(x = 29, y = 1)$ to the goal position `G`$(x = 32, y = 13)$, we can just ask `FindPath` to try to find a path from the North, East, South, and West (in this order) of $(x = 29, y = 1)$:

  ```
  FindPath(x1,y1);//--(x1,y1): north coordinates of (x,y)
  FindPath(x2,y2);//--(x2,y2): east coordinates of (x,y)
  FindPath(x3,y3);//--(x3,y3): south coordinates of (x,y)
  FindPath(x4,y4);//--(x4,y4): west coordinates of (x,y)
  ```

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:
  - The algorithm stops 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) 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) 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=2 in the following maze (in the body of `FindPath(2,2)`).

```
5 #+####
4 #+#  #
3 #+#  #
2 #++# #
1 ###
0 G   ##
  012345
```

- After marking, first, it will try to find a path to the goal from the position North of x=2, y=2, calling `FindPath(2,3)`.
- Since the North position is not open, the call to `FindPath(2,3)` will return `false`, and then it will go back (backtrack) to `FindPath(2,2)`) and resume at the step just after it went North.
- Next, it will go East of x=2, y=3, calling `FindPath(3,2)`.
  - Position (x=3,y=2) is blocked, the algorithm will backtrack to `FindPath(2,2)`) and resume at the step just after it went East.
- Next, it will go South of x=2, y=2, calling `FindPath(2,1)`.
  - Position (x=2,y=1) is blocked, the algorithm will backtrack to `FindPath(2,2)`) and resume at the step just after it went South.
- Next, it will go West of x=2, y=2, calling `FindPath(1,2)`.
  - Position (x=1,y=2) is not open (no empty space), the algorithm will backtrack to `FindPath(2,2)`) and resume at the step just after it went West.
    - Since West is the last direction to search from x=2, y=2, it will unmark x=2, y=2, and backtrack to the previous call, `FindPath(1,2)`.

Assignment Instructions

Rules

- This is a group assignment.
- You are not allowed to hard code the maze in your program. You will need to read maze.txt and store it in your program.
- Prompt the user to enter the coordinates for the start position `S` and the goal position `G`.
  - 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, prompt the user to enter a new location for either `S`, or `G`, or both.
- Implement and call the recursive function `FindPath`.
  - If no path is found, display a message "Path not found" and display the maze with the partial path (from `S` to where it stopped).
  - 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 of your program.
- Code should be documented (Doxygen).
- Divide and conquer
  - Work together to split the tasks. Examples of tasks:
    - ∗ Read maze.txt and store it in your program. What data structure will you use to store the maze?
    - ∗ Deal with invalid user inputs (use enters strings instead of int), non-open locations for `S` and `G`, re-prompt the user for new locations, etc
    - ∗ Implement the recursive function and test it.
    - ∗ Display the maze with partial or solution path.
- Zip file should be properly formatted. Zip file should contain .cpp file(s), Doxyfile, and Readme.txt, which describes how to run your code.