Hoang Tuan Anh Vu
Thuy Tran
Group 10
CS146

Project 3 - Maze

Proposed Solution:

My group will create a program that will automatically generate, print a new random maze, and solve the maze every time we run the program. We will solve the maze in two different ways by using depth-first search (DFS) and breath-first search (BFS). Lastly, we would use the JUnit test to check if the solution is correct or not.

A. Generating a maze: randomly generate an unpredictable maze with only one path between the starting room and the finishing room. The maze should have enough removal walls so that the finishing room is reachable from the starting room but make sure that not too many walls are removed cause it might make the maze too easy to solve.
B. Model a maze: we will represent the maze as a graph data structure, where rooms or cells are vertices and removed walls are edges between vertices.
C. Solving the maze: we will use depth-first search (DFS) and breath-first search (BFS) to solve the maze by finding a path from the starting room to the finishing room.
D. Printing the maze in the output: After using those algorithms to solve the maze, we will output the order in which rooms were visited, indicate the shortest solution path from starting to the finishing room, and the length of the path. However, there are some different ways of displaying the mazes including:
    a. The first maze: We need to make sure that the maze will display like the example, which rooms should be printed with the low order digit of the visitation order number. The starting room is ‘0’. Unvisited rooms should remain a space character.
    b. The second maze: will be output the maze with the shortest path by using the ‘#’ character to represent the path from the starting room to the finishing room. The path will be given on a single line and consists of the index of printed cells, separated by single spaces.

Approach followed:

In order to start and finish this program correctly, we have to review all the files and lectures about the depth-first search (DFS) and breath-first search (BFS) algorithms. We have to understand those algorithms carefully so that we can how to apply them to this program, and how to use them to solve the maze. We also need to learn and understand how to generate and solve the maze physically. However, we still need to learn to write a program that will generate the maze randomly and unpredictably.

Cases to consider:

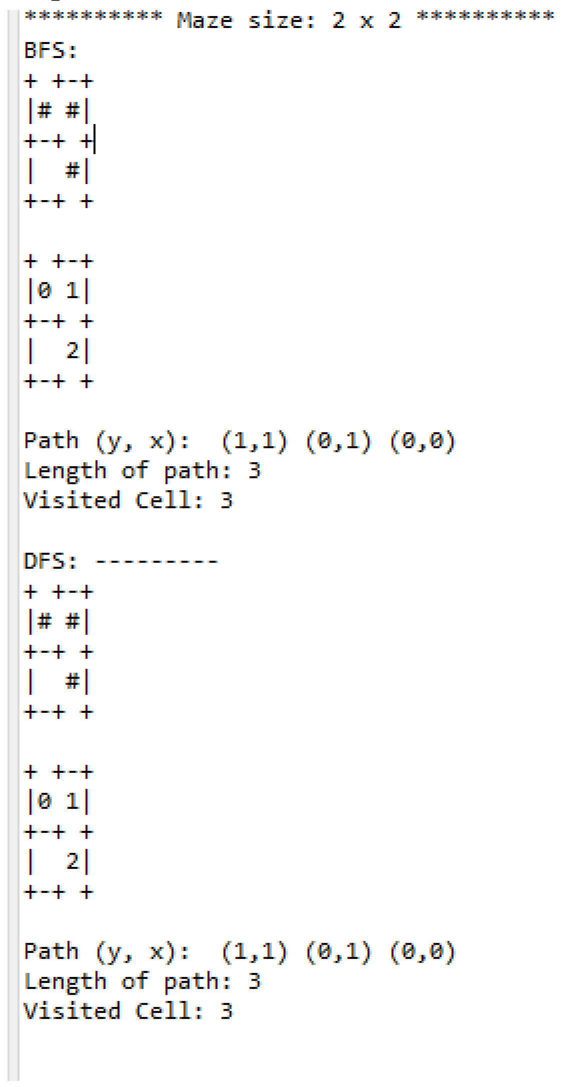
In this project, we need to consider how to use the depth-first search (DFS) and breath-first search (BFS) algorithms to solve the maze with the shortest path solution. We also need to consider how to generate the maze randomly with hard and unpredictable. In order to use JUnit to double-check, we have to make sure that all the paths, solutions that are solved by those algorithms for the maze are completely correct.

Conclusions:

After trying to figure out how to write the program and researching all the things that we need to do this program, we start writing the program. However, there are some mistakes during the process of the program.

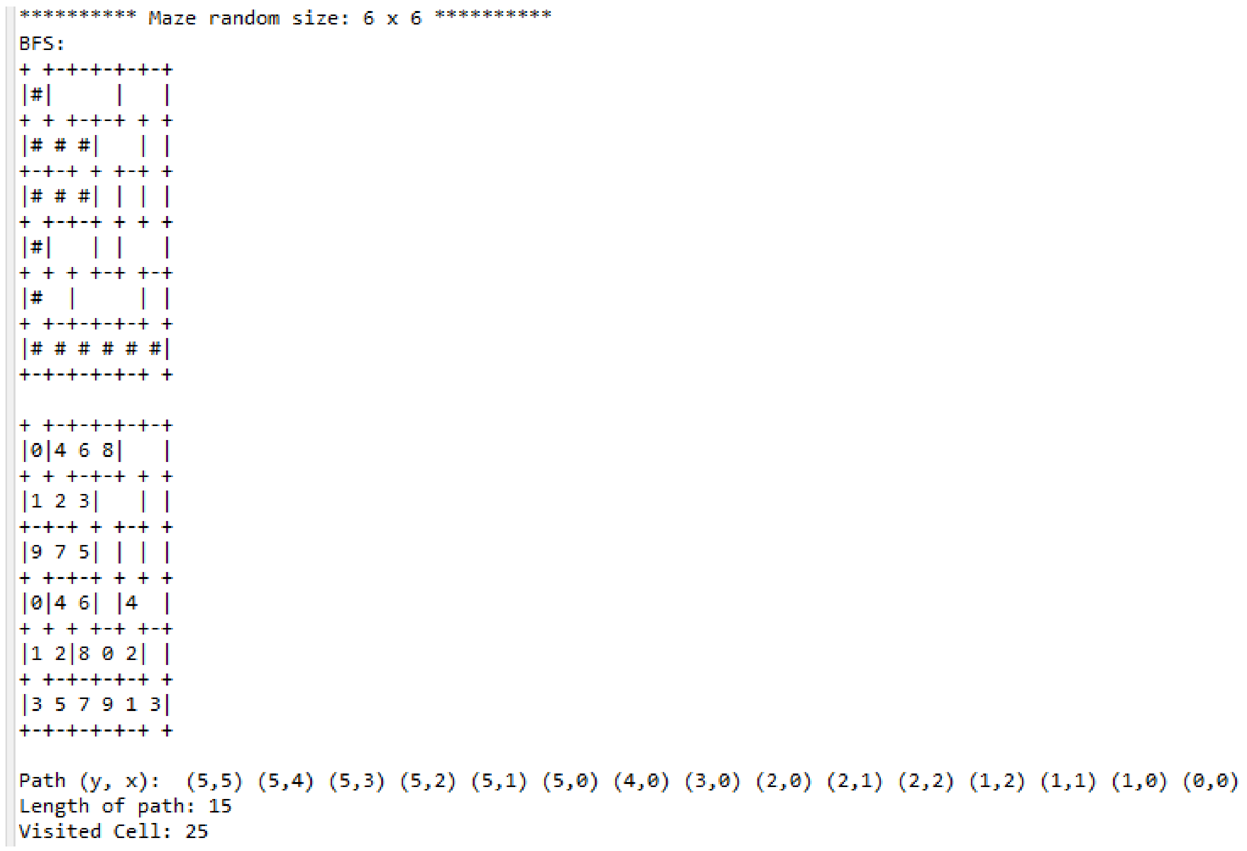
After this project, we learned a lot about the depth-first search (DFS) and breath-first search (BFS) algorithms. We also learned about the logistical of the maze, even with the difficult maze. After researching and trying to solve the maze with the given algorithms, we finally know how to apply those algorithms logistically. However, we also learn about some new JUnit case tests through this project.

Output:

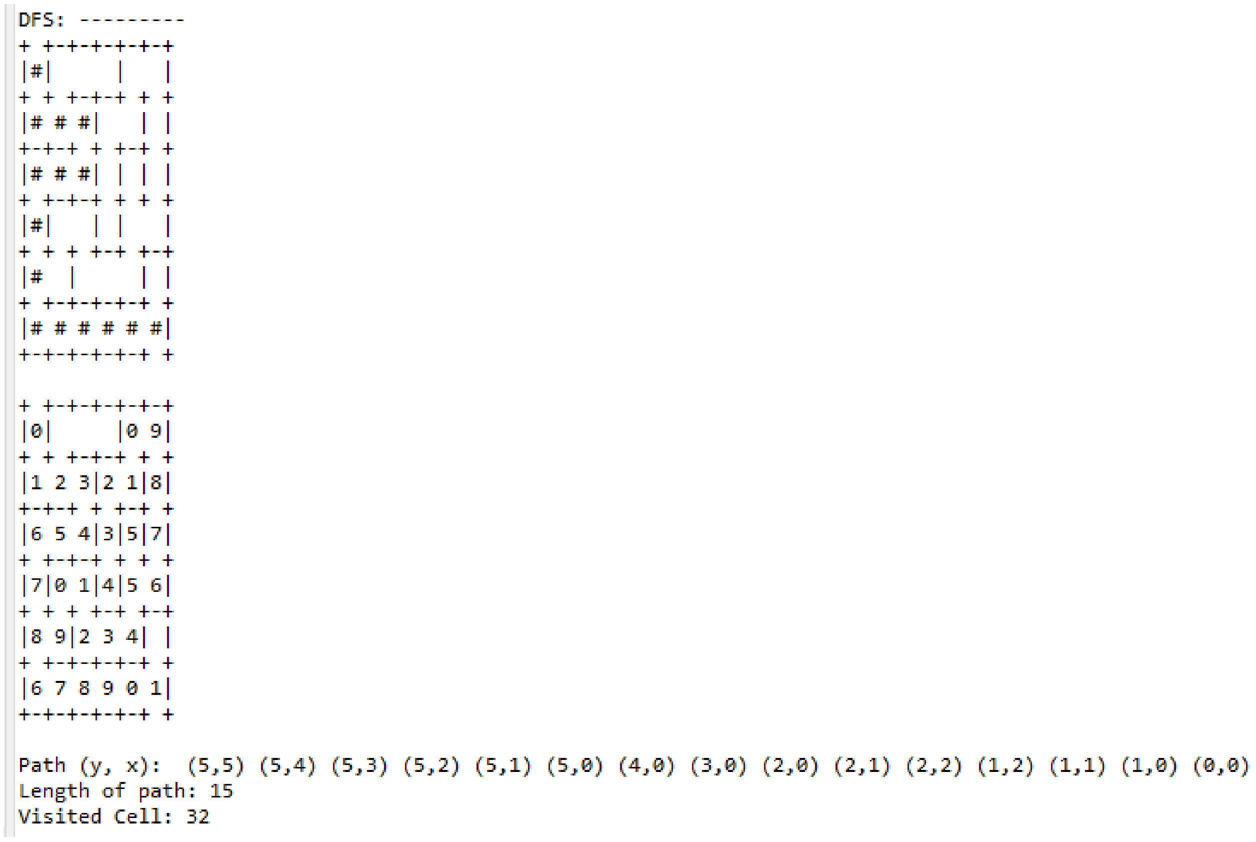


```
********** Maze size: 2 x 2 **********
BFS:
+ +-+
|# #|
+-+ +
|  #|
+-+ +

+ +-+
|0 1|
+-+ +
|  2|
+-+ +

Path (y, x):  (1,1) (0,1) (0,0)
Length of path: 3
Visited Cell: 3

DFS: ---------
+ +-+
|# #|
+-+ +
|  #|
+-+ +

+ +-+
|0 1|
+-+ +
|  2|
+-+ +

Path (y, x):  (1,1) (0,1) (0,0)
Length of path: 3
Visited Cell: 3
```

```
********** Maze size: 3 x 3 **********
BFS:
+ +-+-+-+
|# #|   |
+-+ +-+ +
| |# # #|
+ +-+-+ +
|# # # #|
+ +-+-+-+
|# # # #|
+-+-+-+ +

+ +-+-+-+
|0 1|7 5|
+-+ +-+ +
|1|2 3 4|
+ +-+-+ +
|0 9 8 6|
+ +-+-+-+
|2 3 4 5|
+-+-+-+ +

Path (y, x):  (3,3) (3,2) (3,1) (3,0) (2,0) (2,1) (2,2) (2,3) (1,3) (1,2) (1,1) (0,1) (0,0)
Length of path: 13
Visited Cell: 16

DFS: ---------
+ +-+-+-+
|# #|   |
+-+ +-+ +
| |# # #|
+ +-+-+ +
|# # # #|
+ +-+-+-+
|# # # #|
+-+-+-+ +

+ +-+-+-+
|0 1|6 5|
+-+ +-+ +
|1|2 3 4|
+ +-+-+ +
|0 9 8 7|
+ +-+-+-+
|2 3 4 5|
+-+-+-+ +

Path (y, x):  (3,3) (3,2) (3,1) (3,0) (2,0) (2,1) (2,2) (2,3) (1,3) (1,2) (1,1) (0,1) (0,0)
Length of path: 13
Visited Cell: 16
```



```
********** Maze random size: 6 x 6 **********
BFS:
+ +-+-+-+-+-+
|#|     |   |
+ + +-+-+ + +
|# # #|   | |
+-+-+ + +-+ +
|# # #| | | |
+ +-+-+ + + +
|#|   | |   |
+ + + +-+ +-+
|#  |     | |
+ +-+-+-+-+ +
|# # # # # #|
+-+-+-+-+-+ +

+ +-+-+-+-+-+
|0|4 6 8|   |
+ + +-+-+ + +
|1 2 3|   | |
+-+-+ + +-+ +
|9 7 5| | | |
+ +-+-+ + + +
|0|4 6| |4  |
+ + + +-+ +-+
|1 2|8 0 2| |
+ +-+-+-+-+ +
|3 5 7 9 1 3|
+-+-+-+-+-+ +

Path (y, x):  (5,5) (5,4) (5,3) (5,2) (5,1) (5,0) (4,0) (3,0) (2,0) (2,1) (2,2) (1,2) (1,1) (1,0) (0,0)
Length of path: 15
Visited Cell: 25
```



```
DFS: ---------
+ +-+-+-+-+-+
|#|     |   |
+ + +-+-+ + +
|# # #|   | |
+-+-+ + +-+ +
|# # #| | | |
+ +-+-+ + + +
|#|   | |   |
+ + + +-+ +-+
|#  |     | |
+ +-+-+-+-+ +
|# # # # # #|
+-+-+-+-+-+ +

+ +-+-+-+-+-+
|0|     |0 9|
+ + +-+-+ + +
|1 2 3|2 1|8|
+-+-+ + +-+ +
|6 5 4|3|5|7|
+ +-+-+ + + +
|7|0 1|4|5 6|
+ + + +-+ +-+
|8 9|2 3 4| |
+ +-+-+-+-+ +
|6 7 8 9 0 1|
+-+-+-+-+-+ +

Path (y, x):  (5,5) (5,4) (5,3) (5,2) (5,1) (5,0) (4,0) (3,0) (2,0) (2,1) (2,2) (1,2) (1,1) (1,0) (0,0)
Length of path: 15
Visited Cell: 32
```