Comparison of Hand Follower and Dead-End Filler Algorithm in Solving Perfect Mazes

Y F Hendrawan

Informatics Department, University of Trunojoyo Madura, Bangkalan, Indonesia

E-mail: yonathan.hendrawan@trunojoyo.ac.id

Abstract. A maze is a collection of walls and spaces arranged in such a way that they form a path/paths commonly connecting an entry point to a goal. There are various kinds of maze solving algorithms. These methods are used in practice to help navigate an autonomous system to find a path within maze to find a target such as an exit point, special item, or location to the next level. This research explored two maze solving algorithms: Hand/Wall Follower and Dead-End Filler. Experiments were conducted to measure the time needed for each algorithm in solving 50 perfect mazes of various sizes. The result showed that Hand/Wall Follower algorithm is faster than Dead End Filler algorithm; for example, in solving 800 x 800 cell mazes, the former required an average time of 98.74 milliseconds while the later needed 215.86 milliseconds.

Keywords: Maze, Hand/Wall Follower, Dead-End Filler, Maze Solving Algorithm

1. Introduction

Maze is a collection of walls and spaces arranged in such a way that they form a path/paths commonly connecting an entry point to a goal [1]. There are several types of mazes. One of them is perfect maze which is a type of mazes that does not have not only loops/circular paths, but also insulated parts which no other parts can access to [2]. Figure 2 shows an example of a text-based perfect maze where the walls are represented by “x” characters and the path are represented by space (“ ”) characters.

The generating and solving of mazes can be done using certain algorithms. In gaming, these algorithms are part of Procedural Content Generation: creating the level/world to be played on the fly and in random manner by following a procedure/algorithm with no/minimal human intervention [3]. Sadik et.al found that Graph Theory methods (Flood Fill and DFS) was superior than Non-Graph Theory methods (Wall Follower) [4]. It means that different solver produces different performance. However, there is no complete performance report for all Maze solvers. This research attempts to fill this gap by reporting experimental results for Hand/Wall Follower and Dead-End Filler solver.

2. Method

In order to measure a solver’s performance, the application must have the ability to generate the test mazes. This research built an application that can generate maze layouts dynamically using Recursive Backtracker Method. After that, the application measured the performance of those two solver methods by averaging their run-time in solving 50 different mazes.
2.1. Recursive Backtracker

In generating maze layouts, recursive backtracker method views the maze as a graph: walls as vertices and cells as nodes. The method traverses the graph according to the algorithm’s rule. Every time the path goes through a wall, the wall is deleted. Recursive backtracker algorithm is shown in figure 1.

The process of creating a maze using the method is shown in figure 2.

Initialization: pick one cell as the current cell. Put it into a stack.
While not (all cells have been visited):
  If there is (an) unvisited neighbour cell(s) to the current cell:
    - Randomly carve a path to the cell’s an unvisited neighbour cell.
    - Put the neighbour cell into the stack.
    - Make the neighbour cell to be the current cell.
  Else (all neighbour cells are already visited):
    - Remove current cell from the stack.
    - Pick a cell from the stack and make it current.

Figure 1. Recursive backtracker Algorithm.

Figure 2. From Top-left to bottom-right: the making of a 3 x 3 maze using Recursive Backtracker.

In this research, the entry point is located at top left cell and the exit point is located at bottom right cell.

2.2. Hand/Wall Follower

Wall Follower algorithm solves a maze by placing an agent inside the maze. The agent explores the paths to find the target while keeping itself constantly sticking to right or left side walls. The agent keeps moving until it reaches the target point. In this research, the application uses the Right Wall Follower method. The algorithm is shown in figure 3. The process of creating a maze using the method is shown in figure 4.

Initialization: put the traversing agent at the entry point.
While not (the agent at the exit point):
  - Go forward and mark the cell
    If the right cell is empty (at a right turn):
      - Turn right
    Else:
      - If the front cell is a wall (at a left turn):
        - Turn left
      - If the front cell is a wall (at a dead end):

Figure 3. Hand/Wall Follower Algorithm.
2.3. Dead End Filler

In solving a maze, Dead End Filler algorithm scans its layout, finds, and fills in every dead end. Dead end is a cell whose three out of its four neighbour cells (top, right, bottom, left) are walls. When an agent/player reach a dead-end cell, the only available path for it is to move backwards. For the filling of the dead ends, the algorithm starts making walls from the dead-end cell, moving along the path until reaching a junction. The algorithm is shown in figure 5.

```
Initialization: create and set true to a variable named change.
While change:
    - Set change to false
    For every row:
        - For every column:
            - If cell[row][column] equals entry or exit point:
                - continue
            - If cell[row][column] is a dead end:
                - While current cell is a dead end:
                    - Set current cell to become wall
                    - Find the only neighbour cell which is not a wall
                    - Set the neighbour to be the current cell
                - Set change to true
```

The path traversed by the agent in figure 5 are represented by the “o” characters.

Since Dead End Filler performs its maze solving by building walls, there are always more walls than paths at the end of the process.
3. Experiment and Result
For performance measurement, this study used a laptop with the following specification:
- Chipset: Intel® Express Chipset
- CPU: Intel® Core™ i5-4210U 1.70GHz
- Memory: 4 GB RAM
- Operating System: Windows® 10 Home, version 1803
- Programming Language: Java SE version 1.8.0_161.

The testing was done by averaging the time it takes to solve fifty different-and-on-the-fly generated mazes. Table 1 depicts the results.

| Maze Size (n * n) | Hand/Wall Follower (nanoseconds) | Dead End Filler (nanoseconds) |
|------------------|---------------------------------|-------------------------------|
| 100              | 2,831,446                       | 3,346,342                     |
| 200              | 5,601,081                       | 13,825,612                    |
| 300              | 13,674,536                      | 31,633,519                    |
| 400              | 22,334,861                      | 51,796,606                    |
| 500              | 31,327,674                      | 80,195,358                    |
| 600              | 48,082,438                      | 116,950,871                   |
| 700              | 74,415,445                      | 161,063,004                   |
| 800              | 98,742,608                      | 215,863,773                   |

In general, both algorithms solved the maze speedily: less than a quarter of a second for solving 800 x 800 maze. Both algorithms needed more time as the maze’s size increased. The rate of rise is not constant: approximately twice for Wall Follower and thrice for Dead End Filler when the size went up from 100 to 200; more than half for Wall Follower and less than half for Dead End Filler when the size went up from 500 to 600. Figure 7 shows the result portrayed in table 1 into a graph.

Figure 7. The average time required to solve the mazes.

Comparing both algorithms, Wall Follower algorithm is faster than Dead End Filler algorithm. Throughout all maze sizes, the speed difference is around twice as fast for Wall Follower.
One thing to note here is that the application only used the Right Wall version of the Wall Follower. It might be that the Left Wall version is more appropriate to solve the mazes’ layout where the entry is located at top left and the exit is located at bottom right.

4. Conclusion

The application built generated different maze layouts using Recursive Backtracker method. These layouts were then used as experiment tests for two maze solving algorithm: Wall Follower and Dead-End Filler. By averaging time needed to solve 50 maze layouts for eight different maze sizes, it is found that Wall Follower is faster than Dead End Filler algorithm. The result shows that Wall Follower is approximately twice as fast as Dead-End Filler.

References

[1] Hendrawan Y F 2018 A Maze Game on Android Using Growing Tree Method Journal of Physics: Conference Series 953 012148
[2] Buck J 2015 Mazes for Programmers: Code Your Own Twisty Little Passages (Dallas-Raleigh: Pragmatic Bookshelf)
[3] Shaker N, Togelius J, and Nelson M J 2016 Procedural content generation in games (Switzerland: Springer International Publishing)
[4] Sadik A M, Dhali M A, Farid H M, Rashid T U, and Syeed A 2010 A comprehensive and comparative study of maze-solving techniques by implementing graph theory International Conference on Artificial Intelligence and Computational Intelligence pp 52-56