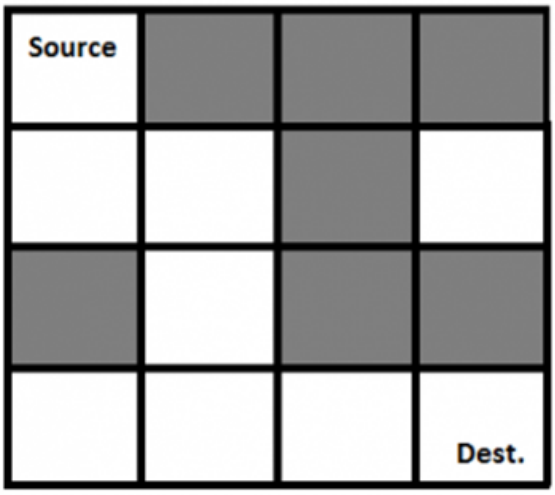
Maze Solver



A maze is a type of puzzle involving a collection of paths, usually where a player has to find a route from start to finish.

A huge variety of algorithms exist for generating and solving mazes. These are not only fun to implement, but also are a good way to familiarise yourself with programming techniques, algorithms, and languages.

Everyone seems to love a good maze - we've been into them for thousands of years at this point.

Webster defines 'maze' as a confusing intricate network of passages, and something confusingly elaborate or complicated.

Many of us have experienced this confusion, whether on paper, in video games, or for those who prefer total immersion, at one of these mazes you can visit in-person.

It's a nice feeling of accomplishment when you finally weave your way to finding the exit, but if that seems like a lot of work, you could always have a computer solve the maze for you.

There are many Algorithms to solve mazes, but here we have used **The Shortest Path-Finding Algorithms**, which is ***Breadth-First Search (BFS)*** and ***Depth-First Search (DFS)*** approach.

Shortest path algorithm

When a maze has multiple solutions, the solver may want to find the shortest path from start to finish. There are several algorithms to find shortest paths, most of them coming from graph theory. One such algorithm finds the shortest path by implementing a breadth-first search, while another, the A* algorithm, uses a heuristic technique. The breadth-first search algorithm uses a queue to visit cells in increasing distance order from the start until the finish is reached. Each visited cell needs to keep track of its distance from the start or which adjacent cell nearer to the start caused it to be added to the queue. When the finish

location is found, follow the path of cells backwards to the start, which is the shortest path. The breadth-first search in its simplest form has its limitations, like finding the shortest path in weighted graphs.

Breadth-First Search :

Breadth-first search, bfs, spreads out like a wave over the maze and can be used for finding the shortest path from one point to another in a graph.

With breadth-first search, you look at each neighbor of a point in the graph that is on the same level before moving on to the next level, which creates the wave effect.

Depth-First Search :

It is a graph/tree traversal algorithm that follows a path as far as it can until it either, reaches the goal or has nowhere else to go. It’s almost like running as far as you can in one direction until you hit a wall.

Assumptions

Each maze has a starting point and an ending point :

This assumption is mostly for simplicity’s sake. We could handle edge cases all day. Like how there may not be any path that leads to an exit

in the maze but that isn’t the point of this article. Instead, we’re going to keep our constraints as close to an actual maze as possible. Surely an actual maze would have an exit. :)

Mazes are composed of walls :

These walls will be considered untraversable. This means we, unfortunately, can’t code our solution in a way that bypasses walls. So unless we’ve recently found a way to walk through matter, we’ve defined a constraint. Therefore, we need to find some way to alert the computer where walls exist or rather what paths are available at any given point in the maze.