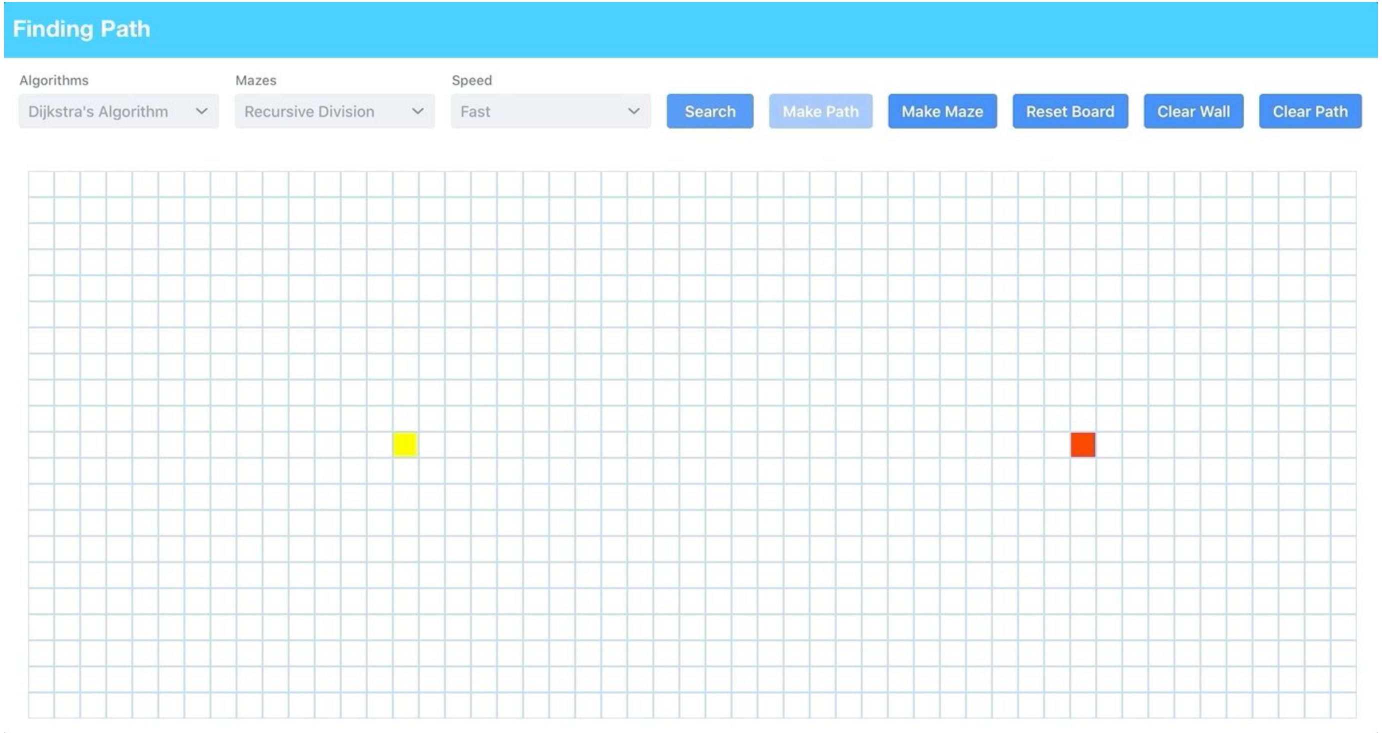
Path Finding

60090021 Nipat Liampisan
61090026 Phustita Sanguansethakul



Finding Path

Algorithms: Dijkstra's Algorithm

Mazes: Recursive Division

Speed: Fast

What is a path finding algorithm?

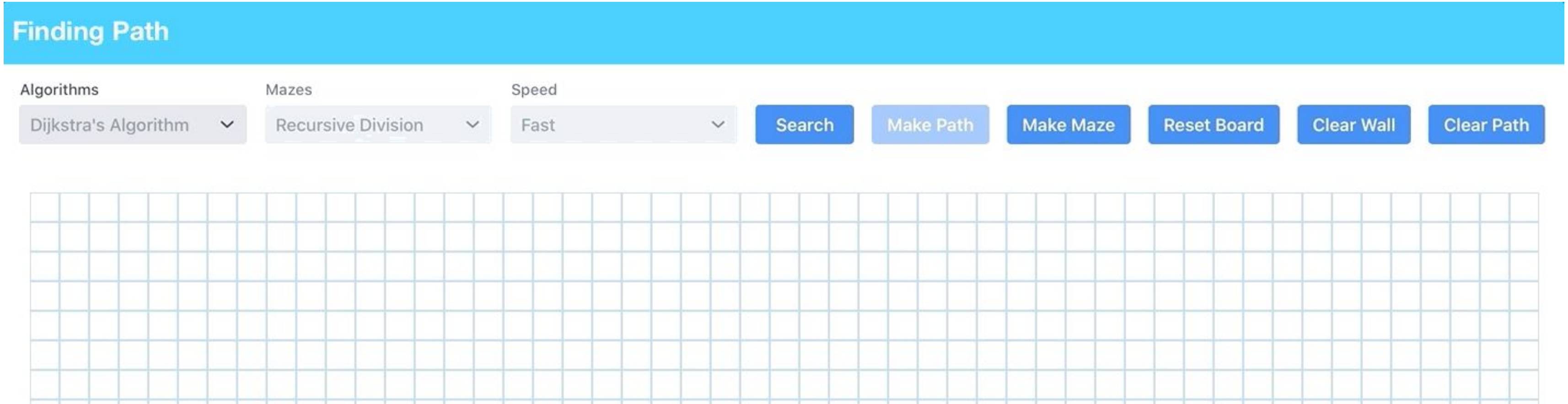
A pathfinding algorithm is an algorithm to find the shortest route between two points. This application visualizes various pathfinding algorithms.

Framework

- Vaadin is an open source platform for web development.
- Includes a set of web components and Java web framework
- Allows the implementation of HTML5 web user interface using Java programming language

Features

- 5 shortest path finding algorithms
- 3 maze generation algorithms
- 3 level of speed
- Allows user to freely draw a wall and move the position of start and target node



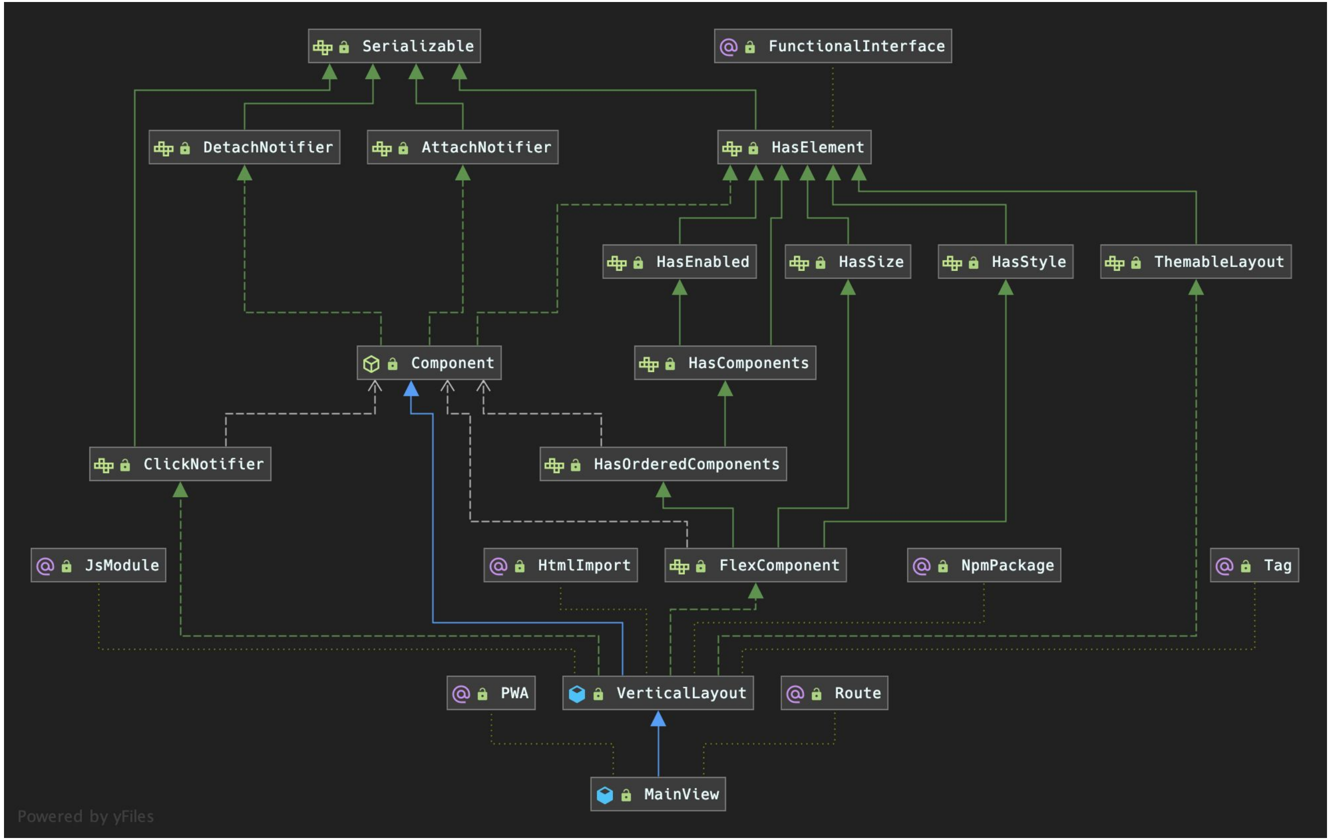
Finding Path

Algorithms: Dijkstra's Algorithm

Mazes: Recursive Division

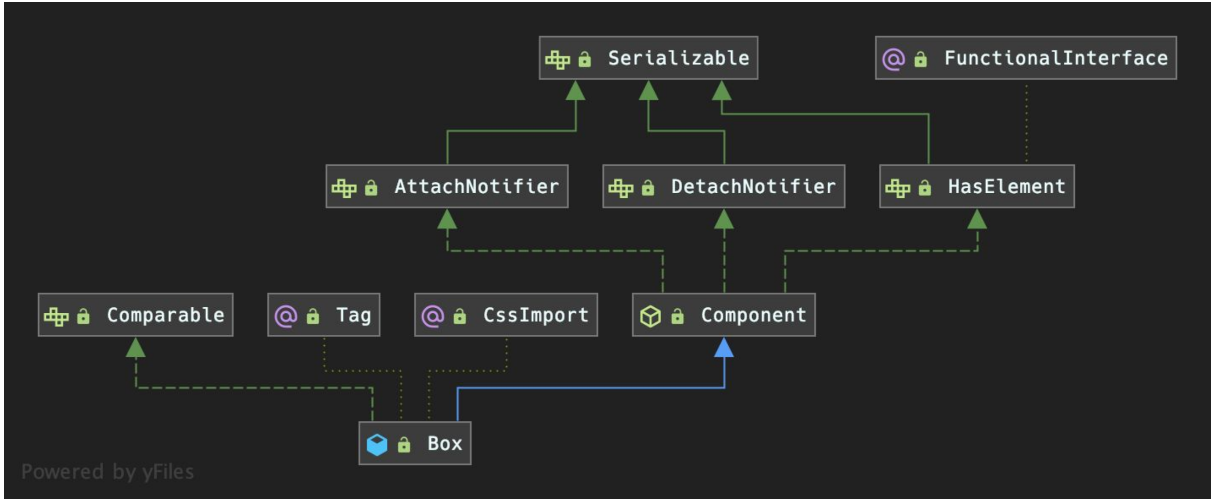
Speed: Fast

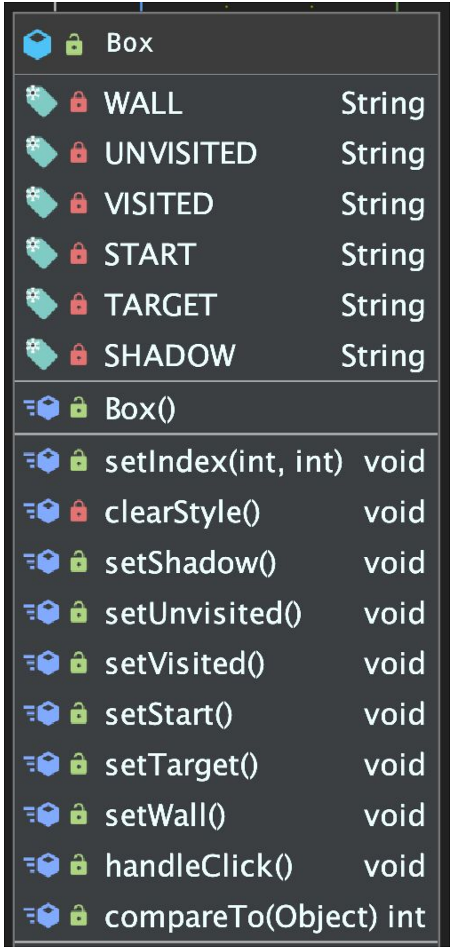
Main Class Diagram



Box Class

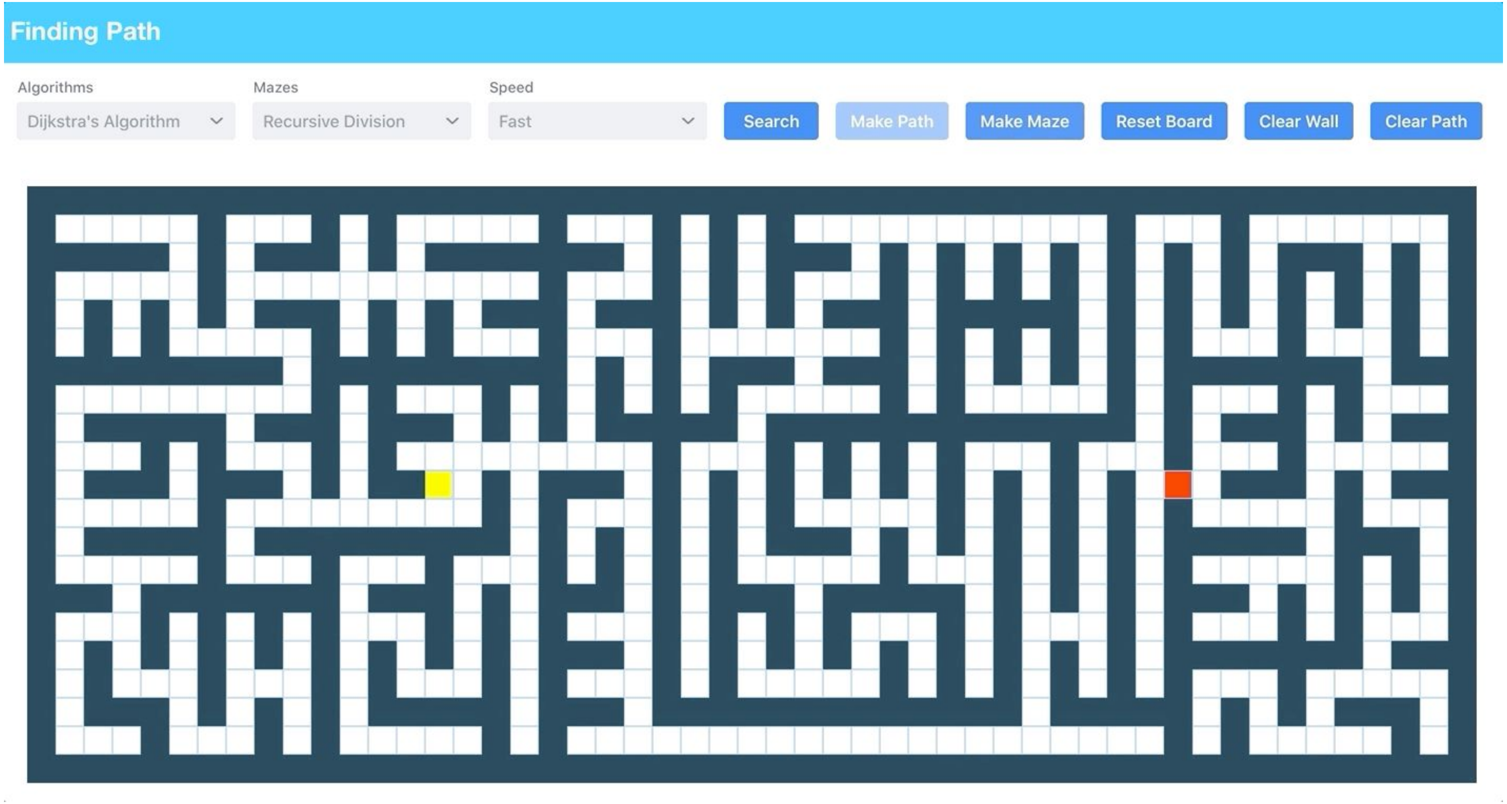
- Represent each node in the grid
- Methods to set status of a box
- Methods to set distance of a box
- Mouse events handler





Box	
WALL	String
UNVISITED	String
VISITED	String
START	String
TARGET	String
SHADOW	String
Box()	
setIndex(int, int)	void
clearStyle()	void
setShadow()	void
setUnvisited()	void
setVisited()	void
setStart()	void
setTarget()	void
setWall()	void
handleClick()	void
compareTo(Object)	int

Maze Classes



Finding Path

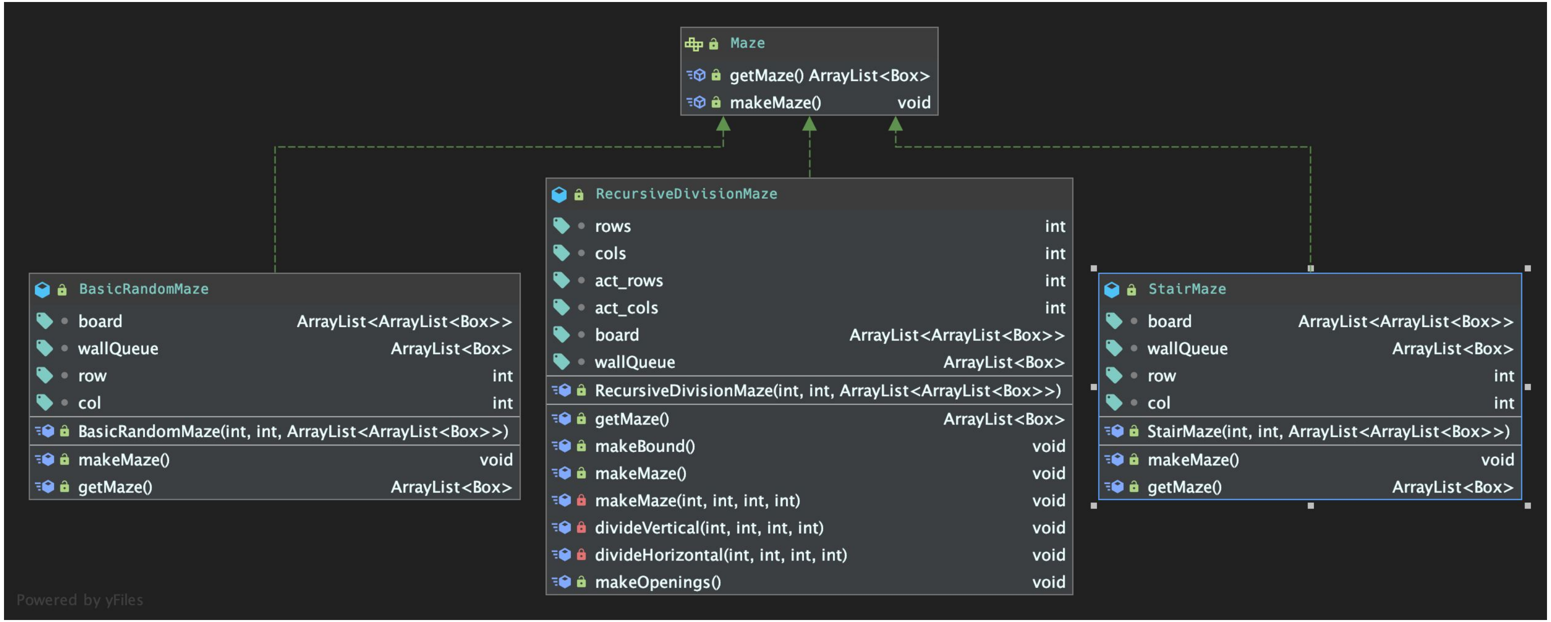
Algorithms: Dijkstra's Algorithm

Mazes: Recursive Division

Speed: Fast

Maze Classes

- Implementing Maze interface
- Recursive division, Basic random and Stair Maze



Maze	
getMaze()	ArrayList<Box>
makeMaze()	void

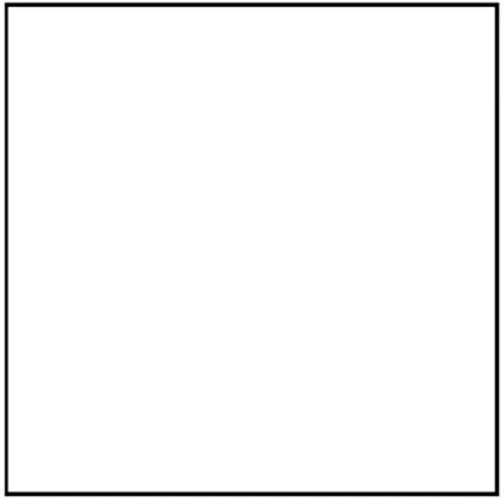
RecursiveDivisionMaze	
rows	int
cols	int
act_rows	int
act_cols	int
board	ArrayList<ArrayList<Box>>
wallQueue	ArrayList<Box>
RecursiveDivisionMaze(int, int, ArrayList<ArrayList<Box>>)	
getMaze()	ArrayList<Box>
makeBound()	void
makeMaze()	void
makeMaze(int, int, int, int)	void
divideVertical(int, int, int, int)	void
divideHorizontal(int, int, int, int)	void
makeOpenings()	void

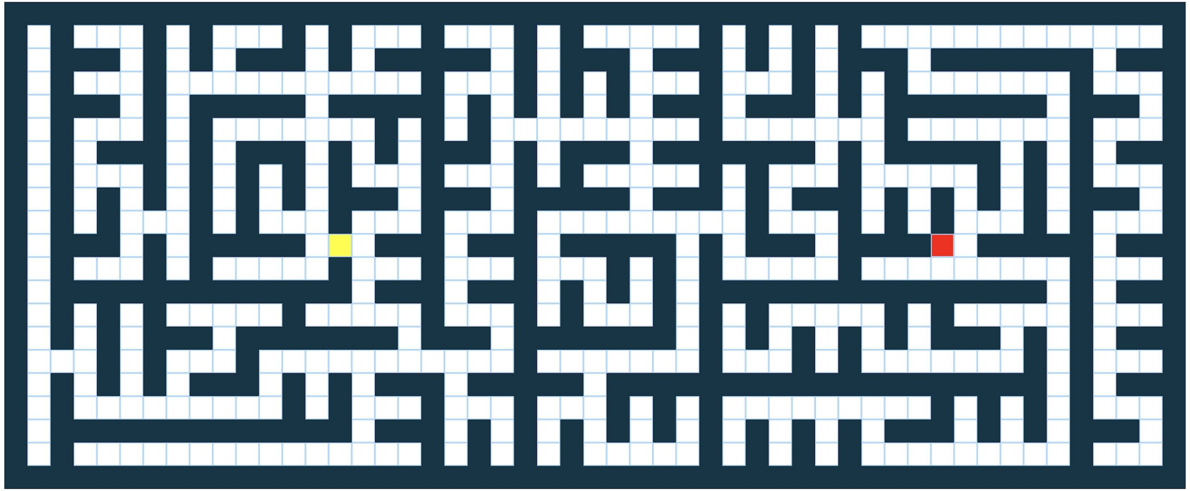
BasicRandomMaze	
board	ArrayList<ArrayList<Box>>
wallQueue	ArrayList<Box>
row	int
col	int
BasicRandomMaze(int, int, ArrayList<ArrayList<Box>>)	
makeMaze()	void
getMaze()	ArrayList<Box>

StairMaze	
board	ArrayList<ArrayList<Box>>
wallQueue	ArrayList<Box>
row	int
col	int
StairMaze(int, int, ArrayList<ArrayList<Box>>)	
makeMaze()	void
getMaze()	ArrayList<Box>

Recursive Division Maze

- Must be implemented as a wall adder
- Randomly draw a vertical/horizontal wall and randomly add an opening to each wall
- Bisecting the larger wall for another vertical/horizontal wall
- Repeat until can't be divided any further

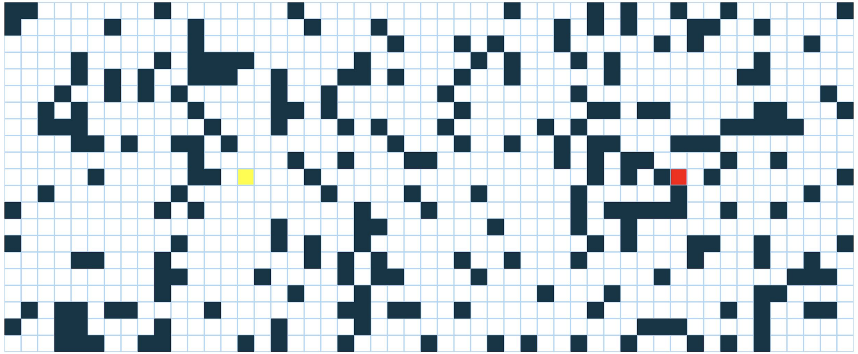




Basic Random and Stair Maze

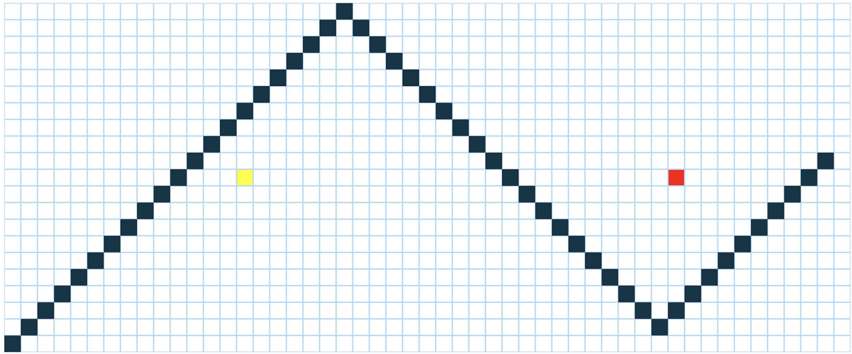
Basic Random Maze

- Pick a random number n between 1 - number of column
- If number of box % n == 0 or (number of box % n) / 3 == 0 then that box is set to be a wall



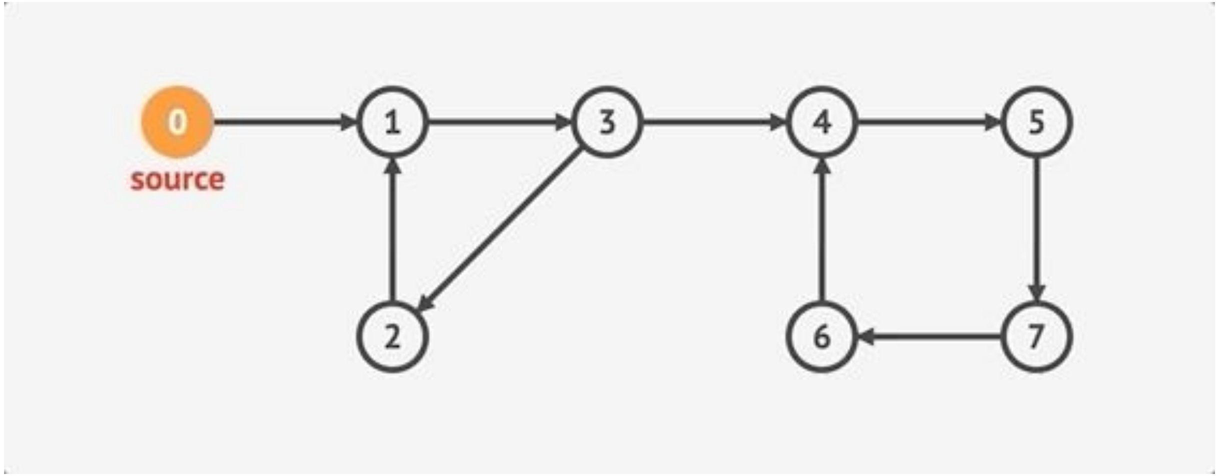
Stair Maze

- Starting from the last row up to 0
- Then goes down to the last row - 1
- Then repeat until the number of column is equal to number of column - 1
- Each step increase number of column by 1



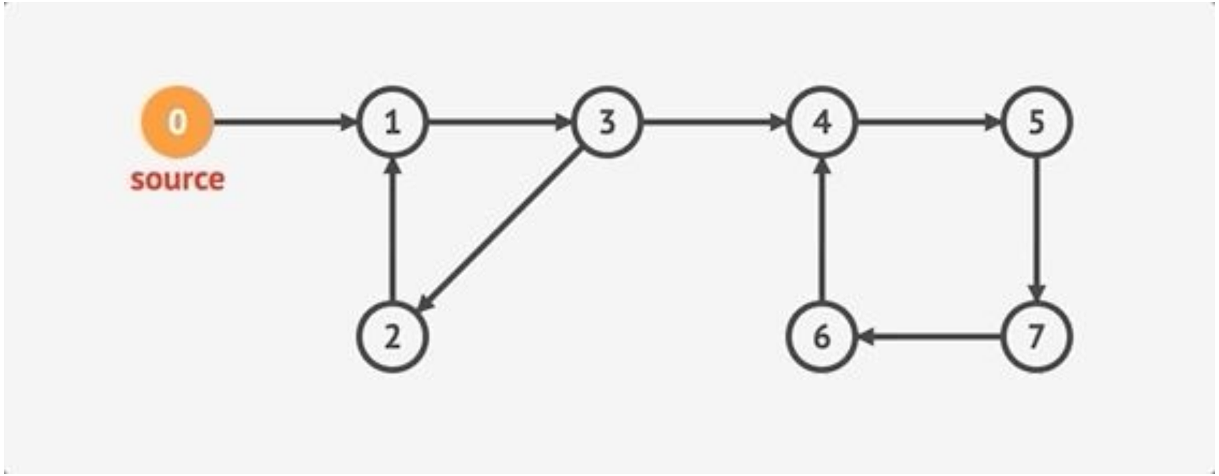
Depth First Search

- Will traverse down a single path, one child at a time and goes to the deepest node until it can't anymore, then backtraces (stack)
- Good for finding whether a path exists between two nodes
- Does not guarantee the shortest path
- Uses less memory/ Fast



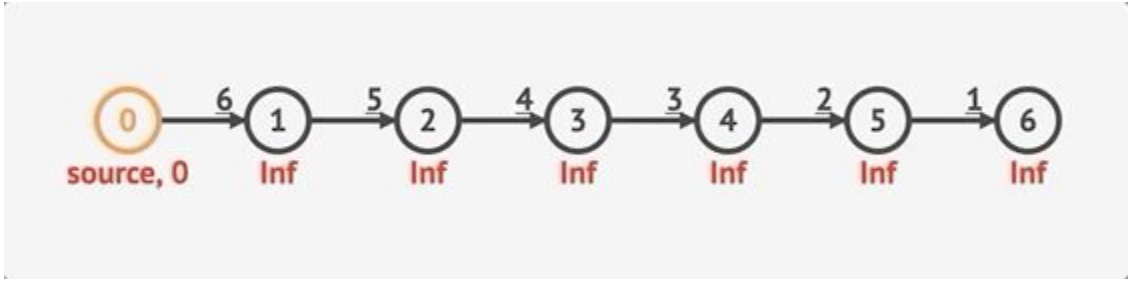
Breadth First Search

- Traverse through the graph one level of children at a time(queue) ; Evaluates all possible paths equally and simultaneously comparing them even during execution
- Guarantees shortest path since all options are exhausted
- Uses large amount of memory/ slow
- Use-case: Flood Fill in paint/photoshop program



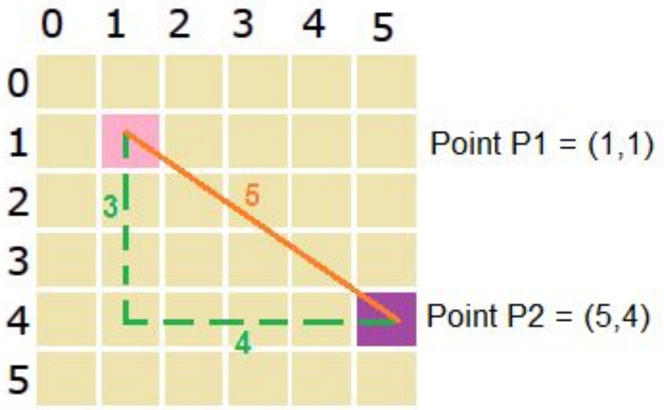
Dijkstra’s Algorithm

- Guarantees shortest path
- Similar to Breadth First Search except edges are weighted
- Uses a priority queue to pick the next vertex with the lowest distance
- The distance between nodes is updated when a new one is less than the original



Greedy Best First Search

- Does not guarantee shortest path
- Different from other searches because it uses a heuristic (estimate)
- Manhattan heuristic based on how far it is from goal
- Instead of choosing the vertex closest to node, it selects the vertex closest to the goal
- Much faster than Dijkstra’s Algorithm



Euclidean distance = $\sqrt{(5-1)^2 + (4-1)^2}$ = 5

Manhattan distance = $|5-1| + |4-1|$ = 7

A* Algorithm

- Does not always find shortest path, depends on approximation of h(n)
- Combination of Dijkstra’s Algorithm and Greedy Best First Search
- Examines the vertex that has the lowest f(n) cost
- f(n) = g(n) +h(n)
- g(n) = cost from current node to vertex
- h(n) = heuristic cost of vertex

tiny.cc/7d8hgz

You can try and visualize algorithms yourself here!
Note that the program **does not support mobile device.**
Please open **full screen** via web browser!