**Simple Code Explanation for BFS (Breadth-First Search) in a Maze**

**1. Necessary Libraries Imported:**

* **pyamaze:** This library helps generate and visualize mazes.
* **deque**: A special queue that allows adding and removing elements from both ends, making it efficient for BFS.
* COLOR, agent, textLabel: These are used to set up the visual appearance of the maze and display information.

**2. BFS Function:**

* **Start Cell:** The search begins from the bottom-right corner of the maze.
* **Frontier**: A deque keeps track of the cells that need to be explored next.
* **Explored**: A list that stores cells that have already been visited to avoid revisiting them.
* **BFS** **Path**: A dictionary that tracks the path taken from the start cell to each visited cell.
* **Goal** **Detection**: The algorithm stops once it reaches the goal (usually the top-left corner of the maze).
* **Path** **Reconstruction**: Once the goal is reached, the BFS path is reconstructed from the goal to the start using the BFS Path dictionary. This creates the shortest path.

**3. Main Section:**

* **Maze** **Creation**: The maze is generated using the pyamaze library and a file (e.g., mazetest.csv).
* **BFS** **Execution**: The BFS function is called to find the order in which the cells are explored, the path taken, and the shortest path from the goal to the start.
* **Agents**:
  + Agent1: Visualizes the BFS search order (how BFS explores the maze).
  + Agent 2: Traces the full BFS path.
  + Agent 3: Shows the forward path, moving from the goal back to the start.
* **Trace** **Path**: This function is used to animate the movement of the agents along their respective paths.
* **Labels**: Labels are displayed to show the lengths of the BFS search path and the forward path.
* **m.run():** This command starts the visualization, showing how BFS explores the maze and traces the path.