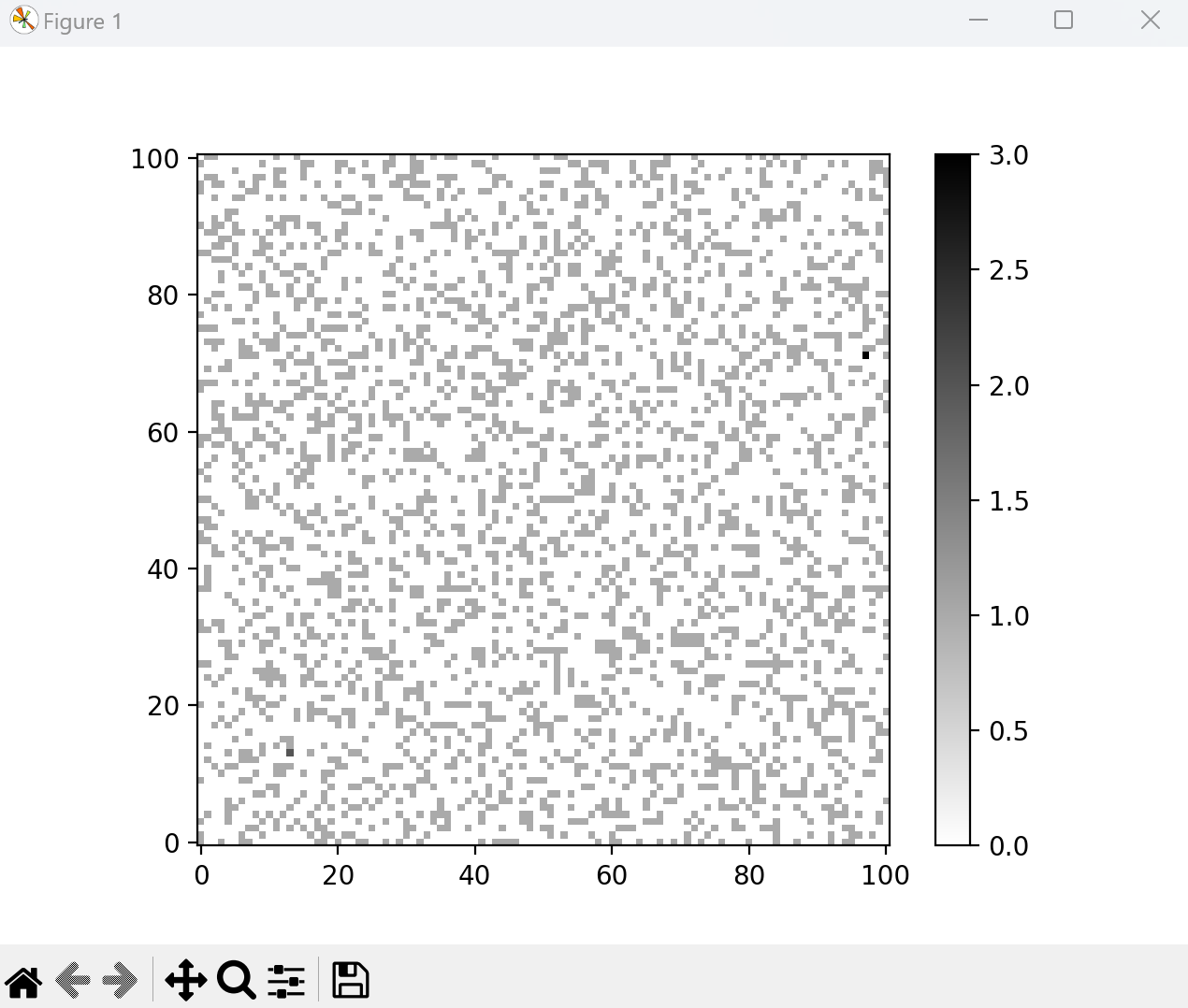
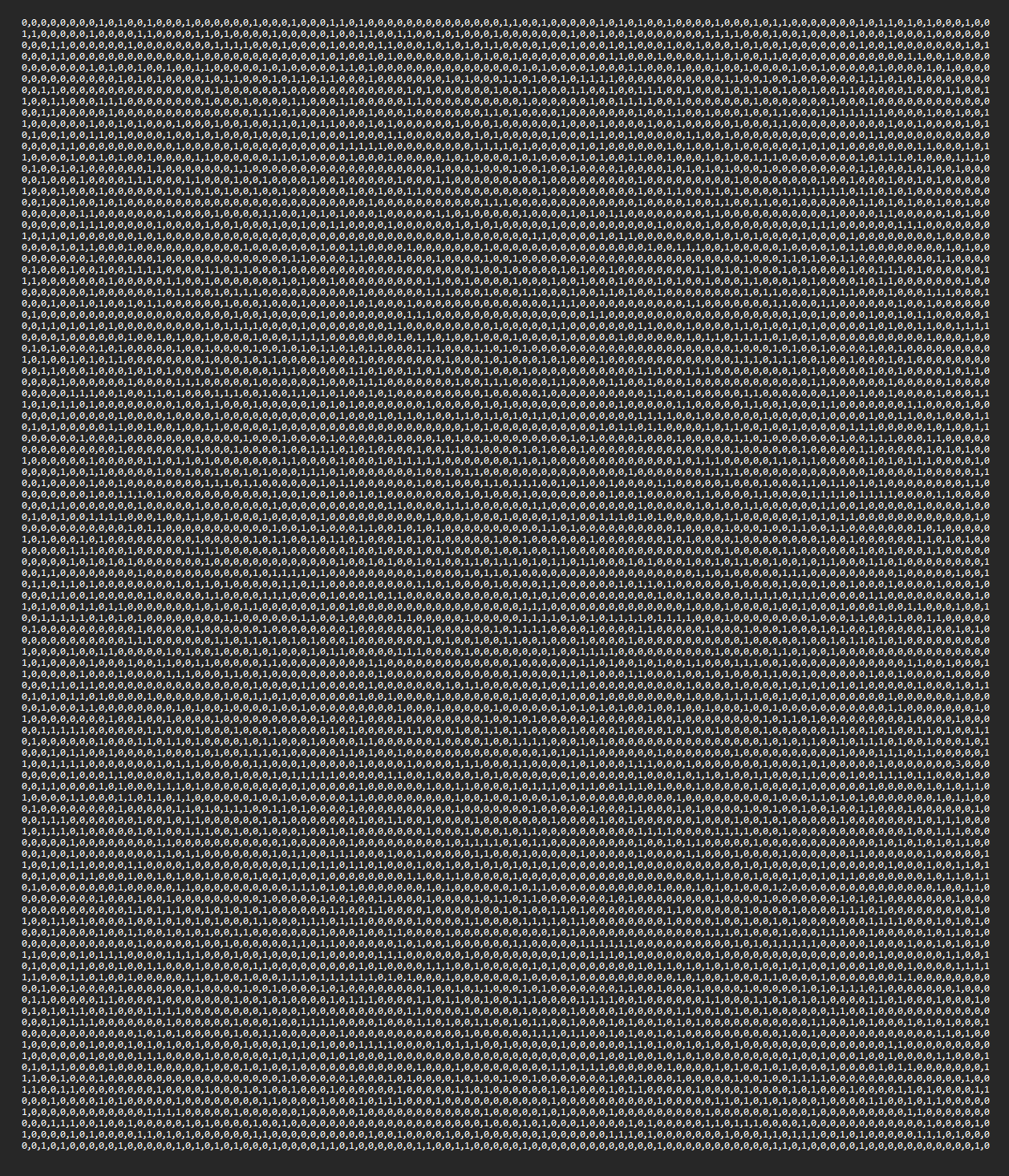
Rough version (To put into LaTex)

Problem 0:

1. **Importing Libraries**:
   * The code begins by importing necessary libraries: **numpy** for array operations, **random** for random number generation, and **matplotlib.pyplot** for visualization.
2. **Function Definitions**:
   * **generateMaze (num\_rows, num\_cols)**: This function generates a maze-like structure with a specified number of rows and columns. It uses a depth-first search approach with random tie-breaking to create the maze. It ensures that each maze has a unique start and end point.
   * **generateMultiple num\_mazes, maze\_size)**: This function generates multiple maze-like structures with unique start and end points. It calls **generateMaze** internally.
   * **visualize\_maze(maze)**: This function visualizes a maze using matplotlib's **imshow** function. It displays the maze as a binary image, where blocked cells are black and unblocked cells are white.
   * **save\_maze\_to\_file(maze, filename)**: This function saves a maze to a text file using numpy's **savetxt** function. It allows for storing the maze structure for later use.
3. **Generating Maze Environments**:
   * The code calls **generateMultiple** to generate 50 grid world environments, each with a set maze size.
   * Each generated maze is stored as a tuple containing the maze array, the start point, and the end point.
4. **Visualizing and Saving Mazes**:
   * The code iterates over the generated mazes, visualizing each maze using **visualize\_maze** and saving it to a text file using **save\_maze\_to\_file**.
   * For visualization, matplotlib's **imshow** function is used to display the maze, and **colorbar** is added to show the color mapping for blocked and unblocked cells.

Example maze:

Numpy easy for smaller maze sizes for testing purposes.

Matplotlib better for visualization of maze.