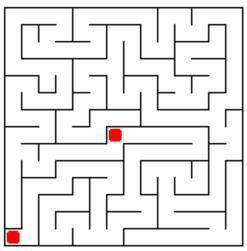
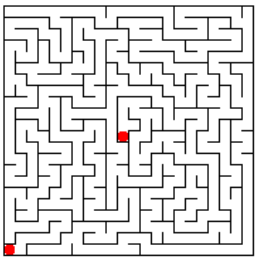
Exercise 1: Perfect maze

Write a program that takes a command-line argument n, and generates a random n-by-n perfect maze. A maze is *perfect* if it has exactly one path between every pair of points in the maze, i.e., no inaccessible locations, no cycles, and no open spaces.





Here's a nice algorithm to generate such mazes. Consider an n-by-n grid of cells, each of which initially has a wall between it and its four neighboring cells. For each cell (x, y), maintain a variable `north[x][y]` that is **`true`** if there is wall separating (x, y) and $(x, y + 1)$. We have analogous variables `east[x][y]`, `south[x][y]`, and `west[x][y]` for the corresponding walls. Note that if there is a wall to the north of (x, y) then `north[x][y] = south[x][y+1] = true`. Construct the maze by knocking down some of the walls as follows:

1. Start at the lower level cell $(1, 1)$.
2. Find a neighbor at random that you haven't yet been to.
3. If you find one, move there, knocking down the wall. If you don't find one, go back to the previous cell.
4. Repeat steps 2 and 3 until you've been to every cell in the grid.

Hint: maintain an $(n + 2)$-by-$(n + 2)$ grid of cells to avoid tedious special cases.

Exercise 2: Connected Components in an Undirected Graph

Using the `Graph` class and `DepthFirstPaths` or `BreadthFirstPaths` classes, determine the number of connected components in an undirected graph. Additionally, list the vertices in each connected component.

Key Concepts: A **connected component** is a subgraph in which any two vertices are connected by paths, and which is connected to no additional vertices in the rest of the graph. This problem explores graph traversal techniques to identify and group connected components.

Input:

- An integer V: the number of vertices in the graph.
- An integer E: the number of edges in the graph.
- A list of E edges, where each edge is represented as a pair of integers (v, w), indicating an undirected edge between vertices v and w.

Output:

- The total number of connected components in the graph.
- For each connected component, list the vertices that belong to it.

Example:

Input:

```
6 4
0 1
0 2
3 4
4 5
```

Output:

```
Number of connected components: 2
Component 1: 0 1 2
Component 2: 3 4 5
```

Instructions:

- Use the `Graph` class to represent the undirected graph.
- Use either `DepthFirstPaths` or `BreadthFirstPaths` to explore the graph.
- Implement the logic to count and list the connected components.