视频演示稿：迷宫生成与求解程序

尹超

中国科学院大学，北京 100049

Carter Yin

University of Chinese Academy of Sciences, Beijing 100049, China

2025.6

序言

本文为笔者数据结构与算法的课程设计报告的演示视频的讲稿。
望老师批评指正。

目录

视频演示稿：迷宫生成与求解程序

视频时长目标：不超过 3 分钟

(开场：程序 GUI 界面已打开，可以先展示一个已生成的矩形迷宫)

步骤 1：开场白与程序概述 (预计用时: 0:00 - 0:25)

- “大家好！今天我将为大家演示一款基于 *Python* 和 *Tkinter* 图形库开发的迷宫生成与求解程序。这款程序不仅能够动态创建和展示常见的矩形迷宫，还能生成结构独特的三角形迷宫。更重要的是，它集成了两种经典的寻路算法——广度优先搜索（*BFS*）和深度优先搜索（*DFS*）——来高效地找出迷宫中的路径。”
- **(可选，快速演示) 快速点击“Clear Path”清除当前路径，然后点击“Solve (BFS)”或“Solve (DFS)”快速展示一次路径求解效果。**

步骤 2：核心数据结构剖析 (预计用时: 0:25 - 1:05)

- “我们首先深入剖析程序的核心数据结构。迷宫的逻辑抽象主要由一个名为 *Maze* 的类来封装和管理。在该类中，最为关键的属性是 *self.cells*，这是一个 *Python* 字典，它构成了迷宫的数字骨架。”
- **(切换到代码编辑器，高亮显示 Maze 类的定义以及 self.cells 属性的初始化部分。)**
- “*self.cells* 字典的键（*key*）是代表每个迷宫单元格的唯一标识符（*ID*）——例如，对于矩形迷宫，这是一个形如 *(行索引，列索引)* 的元组。而对应的值（*value*）是另一个嵌套字典，详细记录了该单元格的各项属性，主要包括：”
  - “*'walls'*：一个字典，记录当前单元格与其所有潜在邻居之间的墙壁状态。键是邻居单元格的 *ID*，值若为 *True* 则表示存在墙壁，*False* 则表示通路。”
  - “*'visited_gen'* 和 *'visited_solve'*：布尔型标志位，分别用于在迷宫生成算法和路径求解算法执行过程中，标记该单元格是否已被访问。”
  - “*'parent'*：在寻路算法（如 *BFS* 或 *DFS*）执行后，存储构成路径时当前单元格的前驱单元格的 *ID*，这对于从终点回溯以重建完整路径至关重要。”
  - “此外，根据迷宫类型的不同，单元格数据还会包含特定几何信息，例如三角形迷宫单元格的顶点坐标 *'vertices'* 和中心点坐标 *'center_coords'*，这些主要用于在 *Canvas* 上精确绘制迷宫；以及 *'is_up'* 属性，用于区分三角形是指向还是向下。”
- **(可以快速展示代码中 _initialize_rectangular_grid 或 _initialize_triangular_grid 方法内 self.cells 被赋值的片段，或者一个具体单元格字典结构的注释示例。)**

步骤 3：关键算法设计解析 (预计用时: 1:05 - 1:50)

- “接下来，我们解析程序中关键的算法设计。迷宫的生成主要采用的是‘随机深度优先搜索’（*Randomized Depth-First Search*）算法。”
- **(切换到代码，高亮显示 generate_maze_randomly 方法的核心逻辑。)**
- “该算法从一个指定的起始单元格开始，将其标记为已访问，并将其压入栈中。当栈不为空时，查看栈顶单元格，随机选择一个尚未访问的邻居。如果找到这样的邻居，则打通两者之间的墙壁，将该邻居标记为

已访问并压入栈中。如果没有未访问的邻居，则从栈中弹出一个单元格（回溯）。这个过程持续进行，直到栈为空，从而生成一个所有单元格都连通的完美迷宫。”

- “在路径求解方面，程序实现了两种广为人知的图搜索算法：”
  - “**广度优先搜索 (BFS)**：”**[高亮 `solve_bfs` 方法]**“*BFS* 从起点开始，逐层向外探索所有可达的邻居单元格。它使用一个队列（在 *Python* 中常用 `collections.deque` 实现）来管理待访问的单元格。由于其逐层搜索的特性，*BFS* 能够保证找到从起点到终点的最短路径（在无权图中，即边数最少的路径）。”
  - “**深度优先搜索 (DFS)**：”**[高亮 `solve_dfs` 方法]**“*DFS* 则沿着一条路径尽可能深地探索，直到达到终点或遇到无法前进的‘死胡同’时才回溯，尝试其他分支。它通常使用栈（同样可由 `collections.deque` 实现）来管理待访问的单元格。*DFS* 找到的路径不一定是最短的，但它能有效地找到一条连通起点和终点的路径。”
- “这两种求解算法都会依据单元格数据中的 `'walls'` 信息来判断单元格间的可通行性，利用 `'visited_solve'` 状态避免重复搜索和无限循环，并借助 `'parent'` 指针在找到终点后反向构造出完整路径。”

步骤 4：程序功能实操演示 (预计用时: 1:50 - 2:45)

操作与口述（同步进行）：

1. **矩形迷宫演示：**
   - “现在，我们来实际操作演示。首先选择‘*Rectangular*’（矩形）迷宫类型。”**[点击界面上的”Rectangular” 单选按钮]**
   - “我们设定行数为 *12*，列数为 *18*。”**[在对应的输入框中分别输入 12 和 18]**
   - “点击‘*Generate Maze*’按钮。”**[点击”Generate Maze” 按钮]**“大家可以看到，一个新的矩形迷宫已经生成。界面上，起点通常以浅绿色高亮，终点以浅红色高亮。”
   - “接下来，我们使用 *BFS* 算法来求解路径。点击‘*Solve (BFS - Shortest)*’。”**[点击”Solve (BFS - Shortest)” 按钮]**“路径已找到！蓝色的线条清晰地标示出了从起点到终点的最短路径。同时，下方的状态栏也会更新，显示路径的相关信息，如找到路径的算法和步数。”
   - “我们可以点击‘*Clear Path*’按钮来清除当前显示的路径。”**[点击”Clear Path” 按钮]**
   - “现在，我们尝试使用 *DFS* 算法求解同一迷宫。点击‘*Solve (DFS)*’。”**[点击”Solve (DFS)” 按钮]**“*DFS* 也成功找到了路径。请注意，*DFS* 找到的路径可能与 *BFS* 找到的最短路径不同，它呈现的是算法探索过程中的一条可行解。”
   - “再次清除路径，为下一次演示做准备。”**[点击”Clear Path” 按钮]**
2. **三角形迷宫演示：**
   - “下面，我们切换到‘*Triangular*’（三角形）迷宫类型。”**[点击界面上的”Triangular” 单选按钮]**
   - “设定三角形迷宫的行数为 *7* 行。”**[在”Triangle Rows” 输入框中输入 7]**
   - “点击‘*Generate Maze*’。”**[点击”Generate Maze” 按钮]**“一个具有独特蜂窝状结构的三角形迷宫便生成了。同样，起点和终点有颜色标记。”
   - “我们依然可以使用 *BFS* 来求解这个三角形迷宫的路径。”**[点击”Solve (BFS - Shortest)” 按钮]**“路径已成功找到并显示。”
   - “本程序还内置了参数校验机制。例如，如果我们输入一个无效的参数，如非数字或超出预设范围的数值，程序会弹出错误提示框，引导用户输入正确的值。”**[快速尝试在行数或列数输入框中输入一个字母或一个非常大的数字，然后点击”Generate Maze”，展示弹出的 `messagebox` 错误提示，然后关闭提示框。]**

步骤 5：总结与展望 (预计用时: 2:45 - 3:00)

- “综上所述，本程序通过面向对象的编程思想，清晰地构建了迷宫的数据模型和核心操作。它不仅实现了基于随机深度优先搜索的迷宫动态生成逻辑，还集成了广度优先搜索和深度优先搜索两种经典的路径求解算法。所有这些功能都通过 *Tkinter* 图形用户界面进行了直观的可视化展示和便捷的人机交互。未来还可以考虑扩展更多迷宫类型，或引入更高级的寻路算法。我的演示到此结束，感谢大家的观看！”