****

课程设计报告书

**题目：C++大作业迷宫游戏**

**学 院 计算机科学与工程学院**

**专 业 计算机类**

**学生姓名 周家泰**

**学生学号 202330711001**

**指导教师 徐红云**

**课程编号 045101571**

**课程学分 2.0**

**起始日期 2024-12-02**

|  |  |
| --- | --- |
| 教  师  评  语 | 教师签名：  日期： |
| 成  绩  评  定 |  |
| 备  注 |  |

**C++大作业迷宫游戏**

**一、选题背景**

本课题旨在设计一个迷宫游戏程序，主要解决的问题是如何根据用户自定义生成迷宫，然后考虑如何在给定迷宫入口的情况下，判断是否存在出口，展示搜索路径，并通过合适的算法找到从入口到出口的路径，或者在无出口时回退到入口。技术要求包括：能够生成任意大小的迷宫，迷宫至少有一个不同于入口的出口；通过递归函数实现迷宫的遍历，并实时显示迷宫的状态；最终根据是否有出口输出相应的提示信息。

本设计的指导思想是采用面向过程的编程思想，通过函数模块化来实现迷宫的生成、遍历和打印。将复杂的任务的每部分分解为多个的函数，每个函数负责完成一个特定的功能。通过主函数main调用这些函数，实现整个程序的运行。同时，结合多文件编程，将程序的不同模块分别放在不同的文件中，提高代码的可读性。

**二、方案论证(设计理念)**

1.设计原理

迷宫游戏的核心是迷宫的生成和遍历。采用面向过程的编程思想，将迷宫的生成、遍历和打印分别封装为函数。通过主函数调用这些函数，实现整个程序的运行。同时通过分文件编写来使代码可读性和规范性进一步提升。

2.方案选择及特点

选择面向过程的编程思想实现迷宫游戏的原因在于其能够更好地组织代码结构。迷宫游戏程序作为一个简短的程序，个人认为在现阶段直接使用面向过程的编程思想会比面向对象的编程思想，更利于编写。通过函数模块化，将迷宫的生成、遍历等封装为独立的函数，便于后续的扩展和修改。每个函数负责完成一个特定的功能，使得程序的逻辑更加清晰。同时，结合多文件编程，将程序的不同模块分别放在不同的文件中，进一步提高代码的组织性规范性。

3.功能及要求

3.1.迷宫生成：根据用户输入的迷宫的宽度和高度，函数生成一个随机迷宫，迷宫中“#”表示墙，“.”表示可通行的过道，且迷宫至少有一个不同于入口的出口，为了方便统一，限定一个入口和一个出口。

3.2.迷宫遍历：从入口开始，按照深度优先搜索策略进行遍历，用“X”标记已走过的路径，并实时输出迷宫的状态，为了方便用户查看，增加了按任意键走一步的功能。

3.3.结果输出：如果存在出口，输出“成功走出迷宫”及行走路径；如果不存在出口，输出“回退到入口”及行走过程。

4.安全性、数据完整性及运行环境

4.1.安全性：程序在运行过程中对用户输入进行严格校验，确保输入的迷宫尺寸合法，避免因非法输入导致程序崩溃或生成迷宫异常。程序在生成迷宫和探索迷宫时用new和delete来管理内存，避免了内存泄露等问题。

4.2.数据完整性：迷宫的生成和遍历过程中，通过函数调用确保迷宫状态的完整性和一致性。

4.3.运行环境：程序基于C++语言开发，采用面向过程的编程思想，可在支持C++的编译环境中运行源文件，如Visual Studio、Dev-C++、Visual Studio Code等，也可以直接运行exe可执行文件。程序对硬件资源要求较低，可以在普通个人计算机上运行。

三、过程论述

1.多文件编程结构

在考虑如何编写迷宫游戏程序的时候，为了更好地组织代码，想到了学习过的多文件编写，打算将其实际应用。所以本设计采用多文件编程，将程序的不同模块分别放在不同的文件中。具体文件结构如下：

main.cpp：主程序文件，负责调用其他模块的函数，完成迷宫的生成、遍历和结果输出。

maze.h：迷宫问题程序的头文件，包含使用的头文件如ctime，命名空间的使用，函数的声明，external关键字声明的在全程序范围内使用的变量等。

maze.cpp：包含迷宫生成，迷宫遍历，迷宫打印等函数。

多文件的建立也是我编写程序的第一步，在这一步中我学习了许多课本上理论学习但是并未进行实践的内容，比如说头文件的编写中使用到的条件编译功能和external关键字的使用，以及在多文件编写时命名空间的处理等。

以下是头文件maze.h的内容，在这里使用了条件编译来防止头文件反复编译，将需要的头文件和使用的命名空间放在这里，然后在maze.cpp和main.cpp中就只需要使用maze.h即可，简便了操作。

#ifndef \_MAZE\_H\_

#define \_MAZE\_H

#include<iostream>

#include<ctime>

using namespace std;

extern int xStart;

extern int yStart;

extern int xEnd;

extern int yEnd;

extern int flagSuccess;

void mazeGenerator(char\*\* array, int length, int width);

void mazeTraverse(char\*\* arr1, int\*\* arr2, int length, int width, int x, int y);

void printMaze(char\*\* arr1, int length, int width);

#endif

本来是使用vs来编写程序，后来打算尝试使用更轻量化工具vscode来进行程序编写，在这个过程中我也学习了多文件的编译连接，如直接使用c\_cpp\_properties.json、launch.json和tasks.json来进行管理[1]，也学习了使用Makefile文件来执行编译连接的操作[2]，以下是Makefile文件的编写，为了让程序在不同环境下都可以正常运行，使用了-static 静态链接[3]。

MazeProblem.exe: maze.o main.o

    g++ maze.o main.o -static -o MazeProblem.exe

maze.o: maze.cpp maze.h

g++ maze.cpp -static -c

main.o: main.cpp maze.h

    g++ main.cpp -static -c

2.主函数main的编写

在本课题设计中，main的运行框架就是整个程序的运行框架，首先整个代码的运行流程图理清如下：

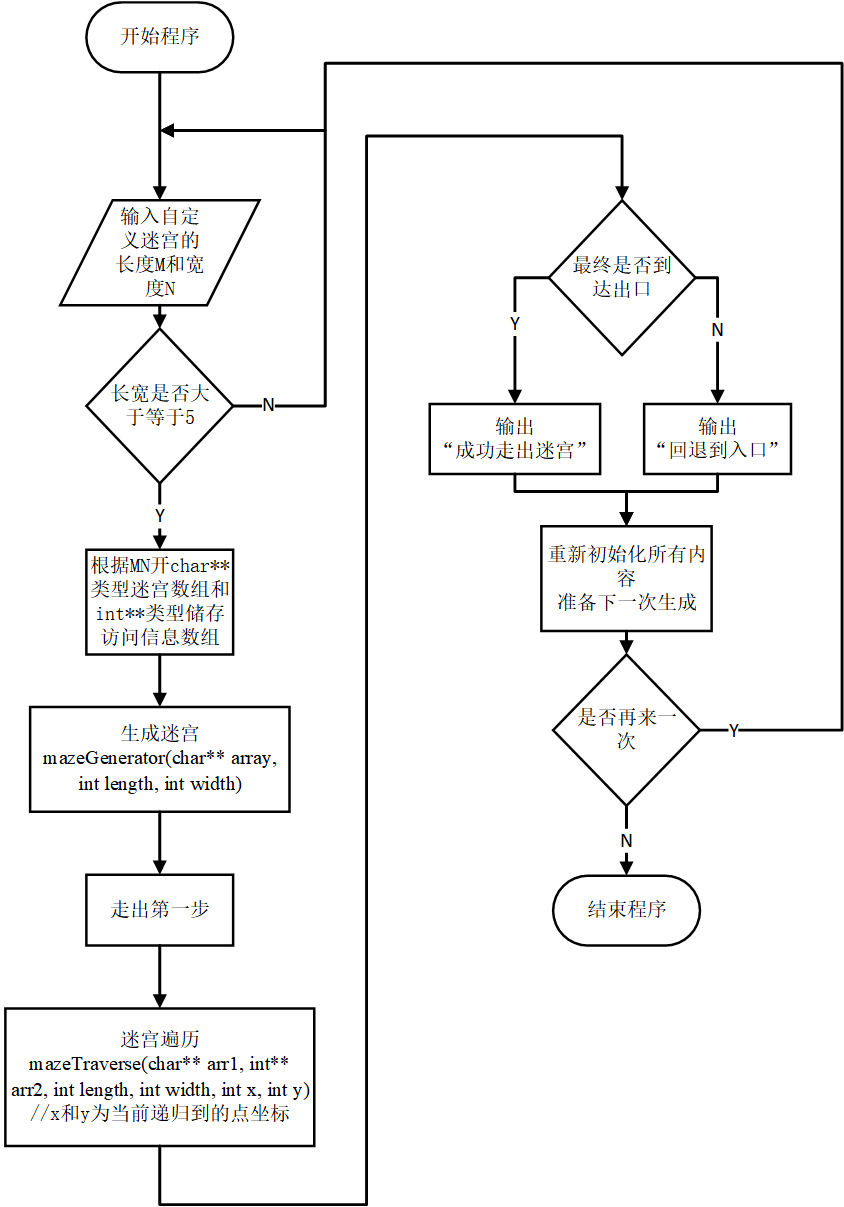


图 1 main函数框架

其中打印函数printMaze由于在各个环节都需要使用，所以在流程图中未提及。根据流程图可以搭建起迷宫游戏设计的总框架，完成main.cpp文件的编写。然后再将生成迷宫和迷宫遍历部分补充完整就完成整个程序的编写。

在内存管理上，编写程序使用了new和delete来进行动态管理：

        char\*\* arrayMaze = new char\* [N];

        for (int i = 0; i < N; i++)

            arrayMaze[i] = new char[M];

        int\*\* arrayVisit = new int\* [N];

        for (int i = 0; i < N; i++)

            arrayVisit[i] = new int[M]();

//开辟

        for (int i = 0; i < N; i++)

            delete[] arrayMaze[i];

        delete[] arrayMaze;

        for (int i = 0; i < N; i++)

            delete[] arrayVisit[i];

        delete[] arrayVisit;

//释放

然后在流程管理上为了防止使用goto导致内容混乱，所以程序使用了do...while;来进行流程管理，直至输入满足题目要求，确保了安全性。为了方便程序的使用，增加了最后判断是否需要再来一次的选项，避免了反复执行exe文件的繁琐。以下是关于重复生成迷宫部分的代码：

int main() {

    char again = '0';

    do{

        //

...

//

        cout << "还要再来一次吗？" << endl;

        cout << "1:再来一次" << endl;

        cout << "0:退出" << endl;

        cin >> again;

        system("cls");

        while (again != '1' && again != '0') {

            cout << "请输入'1'或'0'" << endl;

            cin >> again;

        }

    }

    while (again);

    cout << "欢迎下次使用" << endl;

    return 0;

}

3.迷宫生成模块

迷宫的生成是通过一个函数mazeGenerator实现的。该函数接收储存迷宫内容的数组，以迷宫的宽度和高度作为参数，随机生成一个迷宫，并确保迷宫至少有一个入口和一个出口。

函数的声明如下：void mazeGenerator(char\*\* array, int length, int width);

一开始在思考生成迷宫问题的时候，打算直接使用随机数来生成0和1，通过时间作为生成随机数的种子，然后根据01来填入’#’和’.’，然后将四面外围格子填充’#’作为墙壁，再随机生成两个出口入口，确定下坐标在全程序中使用。后来发现由于使用这种办法的话，往往不能够很好地生成一个迷宫，生成的迷宫会出现迷宫中无法出现完整通路的情况，迷宫路况非常糊涂。使用该方法生成的迷宫内容如下：

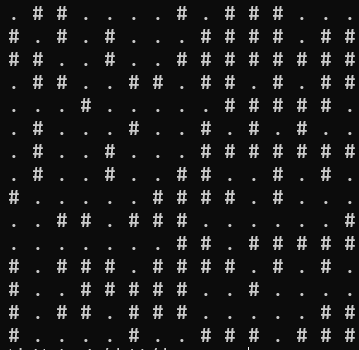


图 2 直接使用随机数生成迷宫

为了解决这种情况，在经过查阅资料后，我最终使用了DFS来生成迷宫[4]。首先将整个迷宫填充为墙，将靠近边缘随机选取状态为墙的道路点，作为起点，然后沿着四个方向跨两个寻找同样为墙的道路点，如果找到，则将它们之间的墙体打通，然后将该点作为新的起点，然后继续进行递归。如果四个方向都没有找到，则不断回退到上一点，直到找到有一点其他方向满足条件，然后继续填充。当运行到最后无法填充的时候，迷宫也就生成完毕了。

这个DFS生成迷宫的功能作为一个小的函数，声明为dfsGenerate函数，在mazeGenerator中被调用，负责迷宫内容部分的填充。

相关代码如下：

void mazeGenerator(char\*\* array, int length, int width) {

    // 初始化迷宫，全部设为墙

    for (int i = 0; i < length; ++i) {

        for (int j = 0; j < width; ++j) {

            array[j][i] = '#';

        }

    }

    // 随机选择一个起点

    int startX = rand() % ((length - 1) / 2) \* 2 + 1;

    int startY = rand() % ((width - 1) / 2) \* 2 + 1;

array[startY][startX] = '.';

    // 开始DFS生成迷宫

dfsGenerate(array, startX, startY, length, width);

    //下面开始生成出入口

    int aStart = rand() % (4), bStart = 0;//aStart表示在哪一面墙 bStart表示在具体第几格 先初始化

    switch (aStart) {

    case 0: {//入口在上方

        bStart = rand() % (length-2)+1;

        array[0][bStart] = '.';

        yStart = 0, xStart = bStart;

        break;

    }

    case 1: {//入口在右方

        bStart = rand() % (width-2)+1;

        array[bStart][length - 1] = '.';

        yStart = bStart, xStart = length - 1;

        break;

    }

    case 2: {//入口在下方

        bStart = rand() % (length-2)+1;

        array[width - 1][bStart] = '.';

        yStart = width - 1, xStart = bStart;

        break;

    }

    case 3: {//入口在左方

        bStart = rand() % (width-2)+1;

        array[bStart][0] = '.';

        yStart = bStart, xStart = 0;

        break;

    }

    }

//生成入口

    int aEnd = rand() % (4), bEnd = 0;

    do{

        switch (aEnd) {

        case 0: {

            bEnd = rand() % (length-2)+1;

            array[0][bEnd] = '.';

            yEnd = 0, xEnd = bEnd;

            break;

        }

        case 1: {

            bEnd = rand() % (width-2)+1;

            array[bEnd][length - 1] = '.';

            yEnd = bEnd, xEnd = length - 1;

            break;

        }

        case 2: {

            bEnd = rand() % (length-2)+1;

            array[width - 1][bEnd] = '.';

            yEnd = width - 1, xEnd = bEnd;

            break;

        }

        case 3: {

            bEnd = rand() % (width-2)+1;

            array[bEnd][0] = '.';

            yEnd = bEnd, xEnd = 0;

            break;

        }

        }

    }

    while (aStart == aEnd && bStart == bEnd);//如果出入口相同就重新分配出口

}

以下是dfsGenerate函数的内容：

int directions[4] = { 0, 1, 2, 3 };//DFS生成迷宫时候用来打乱顺序

// 深度优先搜索生成迷宫

void dfsGenerate(char\*\* maze, int x, int y, int length, int width) {

    // 定义方向数组

    const int dx[4] = { 0, 1, 0, -1 };

    const int dy[4] = { 1, 0, -1, 0 };

    srand((unsigned)time(NULL));

    // 随机打乱方向顺序

    for (int i = 0; i < 4; ++i) {

        int randIndex = rand() % 4;

        int temp = directions[i];

        directions[i] = directions[randIndex];

        directions[randIndex] = temp;

    }

    for (int i = 0; i < 4;i++) {

        int nx = x + dx[directions[i]] \* 2;//找到下一个点的坐标

        int ny = y + dy[directions[i]] \* 2;

        if (nx > 0 && nx < length - 1 && ny > 0 && ny < width - 1 && maze[ny][nx] == '#') {

            maze[ny][nx] = '.';

            maze[y + dy[directions[i]]][x + dx[directions[i]]] = '.';//将原来的点与新选取的点打通

            dfsGenerate(maze, nx, ny, length, width);

        }

    }

}

不过上面的代码为最终版，一开始的代码如下：

// 深度优先搜索生成迷宫

void dfsGenerate(char\*\* maze, int x, int y, int length, int width) {

    // 定义方向数组

    const int dx[4] = { 0, 1, 0, -1 };

const int dy[4] = { 1, 0, -1, 0 };

    int directions[4] = { 0, 1, 2, 3 };//DFS生成迷宫时候用来打乱顺序

    srand((unsigned)time(NULL));

    // 随机打乱方向顺序

    for (int i = 0; i < 4; ++i) {

        int randIndex = rand() % 4;

        int temp = directions[i];

        directions[i] = directions[randIndex];

        directions[randIndex] = temp;

    }

    for (int i = 0; i < 4;i++) {

        int nx = x + dx[directions[i]] \* 2;//找到下一个点的坐标

        int ny = y + dy[directions[i]] \* 2;

        if (nx > 0 && nx < length - 1 && ny > 0 && ny < width - 1 && maze[ny][nx] == '#') {

            maze[ny][nx] = '.';

            maze[y + dy[directions[i]]][x + dx[directions[i]]] = '.';//将原来的点与新选取的点打通

            dfsGenerate(maze, nx, ny, length, width);

        }

    }

}

然后出现过了下图所示的生成结果：

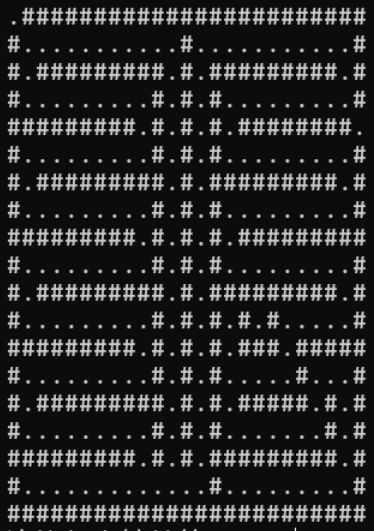


图 3 使用DFS生成迷宫时的意外状况

在这个生成结果中，迷宫生成过于规律化，显然是生成的方向没有随机，然后开始debug。首先将srand注释掉，结果生成的迷宫又变得随机了，确定是迷宫随机数的问题。然后在网上查找随机数相关资料[5]，然后发现每次调用srand()都将重新设置随机数种子，由于time(NULL)返回值是从1970年1月1日至今所经历的时间(以秒为单位)，而两次生成随机数的时间间隔小于1s，两次设置的种子一致，导致两次输出的随机数序列相同，得到相同的随机数序列。所以不能根据时间来直接得到随机的方向。所以我将int directions[4] = { 0, 1, 2, 3 };放在了函数的外面，这样在生成方向的时候都是在前一次方向的基础上进行打乱，改正后结果如下：

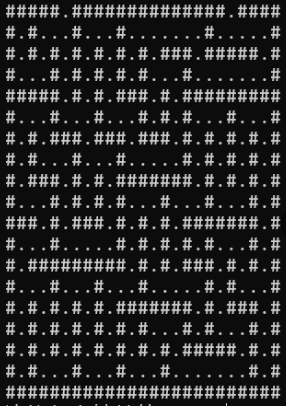


图 4 修改后输出迷宫

在改正后的迷宫显然变得更加随机，不过也因为这种办法所生成的结果为奇数长度的迷宫，所以当输入迷宫长宽为偶数和长宽小于5格时，生成的迷宫效果不佳，所以在主函数的MN输入范围那里加入了对于长度的限定。而对于奇偶没有限定，因为如果是奇数情况下，往往迷宫都可以由入口找到出口，所以如果想要验证“回退到入口”这个功能，就可以通过故意输入偶数长宽来导致出口和入口不互通。

4.迷宫遍历模块

迷宫的遍历是通过一个递归函数mazeTraverse实现的。该函数从入口开始，按照深度优先搜索策略DFS进行遍历，用“X”标记已走过的路径，并实时输出迷宫的状态，每一次标记都会完成一次输出，然后当遇到错路回溯时取消标记也会进行一次输出。以下为整个函数的代码：

void mazeTraverse(char\*\*arr1,int\*\*arr2,int length,int width,int x,int y) {

    if (x == xEnd && y == yEnd) {

        flagSuccess = 1;

        return;

    }

    int xMove[4] = {1,0,-1,0};

    int yMove[4] = {0,1,0,-1};//DFS搜索的方向数组

    for (int i = 0; i < 4; i++) {

        int xTemp = x+xMove[i];

        int yTemp = y+yMove[i];

        if ((xTemp >= 0 && xTemp < length) && (yTemp >= 0 && yTemp < width)&&flagSuccess==0)

        if (arr1[yTemp][xTemp] == '.' && arr2[yTemp][xTemp] == 0) {

            arr2[yTemp][xTemp] = 1;

            arr1[yTemp][xTemp] = 'X';//走过的路标识

            printMaze(arr1, length, width);

            mazeTraverse(arr1, arr2, length, width, xTemp, yTemp);

            if(flagSuccess==0){

                arr2[yTemp][xTemp] = 0;

                arr1[yTemp][xTemp] = '.';//回退

                printMaze(arr1, length, width);

            }

        }

    }

}

首先从入口出发，然后按照右下左上的方向开始寻找迷宫对应内容为’.’的格子和标记访问的数组对应内容为0即未访问的格子，找到以后就将该格子设置为’X’和1，标记访问过了，然后以此格为新的起点进行递归，然后当找不到可以行走的格子，遇到死路的时候进行回退，回到上一个递归中，标记原来走过的路重新为’.’和0，当最终到达起点或者到达终点时整个函数结束，搜索迷宫的工作就结束了，回到main函数中。

5.迷宫打印

前期在打印迷宫的时候，没有在每个单元格间输出空格，导致前文中的部分图比较拥挤，最后在整体修改的时候加上了。为了让每动一步都能更好地展示，添加了system(“pause”)和system(“cls”)，以便于每按一步清除以前输出，输出新迷宫内容。

void printMaze(char\*\* array, int length, int width) {

    system("pause");

    system("cls");//每按一次清除控制台

    for (int i = 0; i < width; i++) {

        for (int j = 0; j < length; j++) {

            cout << array[i][j]<<' ';

        }

        cout << endl;

    }

    cout << "入口坐标为：第" << xStart + 1 << "列,第" << yStart + 1 << "行" << endl;

    cout << "出口坐标为：第" << xEnd + 1 << "列,第" << yEnd + 1 << "行" << endl;

}

1. **结果分析**

完整使用该程序的整个流程如下：

1.长度和宽度输入：在长度或宽度小于5时会提醒重新输入，长度和宽度可以为偶数，但是建议为奇数，这样更容易生成有出口的迷宫，输入中存在偶数长度的时候可以用来验证回退到终点的功能。

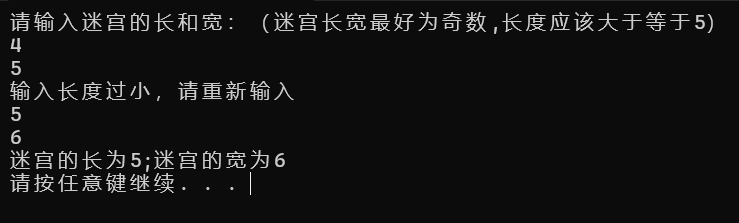


图 5 输入功能演示

2.迷宫生成：在接受了正确的长度宽度输入以后，按任意键可以清屏并输出迷宫内容，存在偶数长度边的迷宫会额外多生成一堵墙，所以建议使用奇数长度，偶数边长适于验证回退到起点功能。

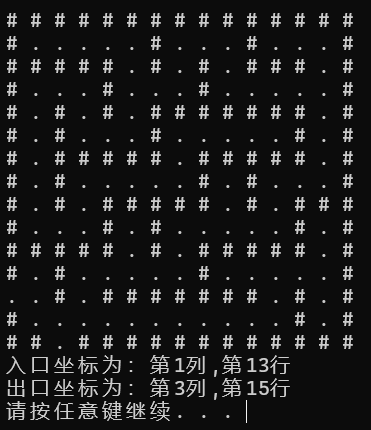


图 6 15\*15的迷宫生成

3.迷宫遍历：在生成迷宫以后，每按一次任意键都可以使搜索进行下一步。

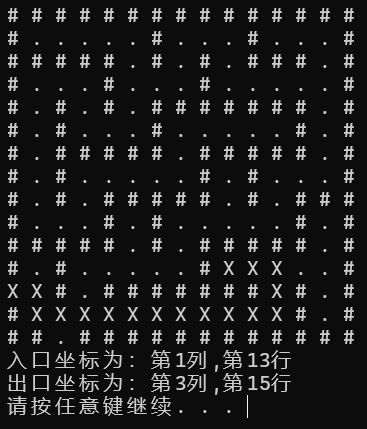


图 7 迷宫遍历中

4.结束之后输出结果：在成功找到出口之后，会输出成功走出迷宫没能走到出口，回到起点的时候会输出“回退到入口”。

|  |  |
| --- | --- |
| {7CA6AD5D-A71F-41AB-82A9-39816A3AE4C4} | {E8152FC4-86D5-45D5-ADA7-411789FD7AB7} |
| 图 8 成功走出迷宫 | 图 9 回退到入口 |

5.下一次迷宫游戏：在输出结果以后会询问是否进行下一次迷宫游戏，1为继续，0为退出，输入其他内容会提示输入非法，要求继续输入。

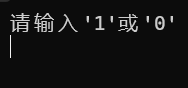
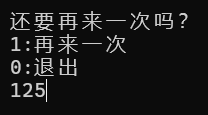


图 10 下一次迷宫游戏的询问

从流程和结果来看，这个课程设计程序可以完成题目所要求的所有功能，并加上了一些改进，如在走完迷宫以后，通过询问用户是否需要重新生成迷宫来防止在需要生成多次迷宫时反复打开程序造成不便。总的来看该程序设计迷宫生成模块能够根据用户输入的尺寸随机生成符合要求的迷宫，确保了入口和出口的存在，同时通道分布合理。在迷宫遍历方面，程序采用深度优先搜索策略，成功实现了从入口到出口的路径搜索，并在每一步实时输出迷宫状态，增强了用户体验。对于有出口的迷宫，程序能够找到正确的路径并输出“成功走出迷宫”；对于无出口的迷宫，程序能够正确回退到入口并提示用户。文件使用静态链接保证了可执行文件可以在缺少环境的情况下正常运行。在多次测试下没有发现bug。

1. **课程设计总结**

在这次的课程设计中，感觉进步最大的一方面是我在面对问题时独立思考独立寻找答案的能力。在这次课程设计中出现了许多尚未掌握的知识和许多尚不了解的问题，面对这些困难我学会了如何上网查找相关解决方案。比如多文件编写的方法，多文件编译连接生成可执行文件的方法，使用DFS生成规范迷宫的方法，遇到“由于找不到 libstdc++-6.dll，无法继续执行代码”的问题和srand使用时间作为随机数种子输出有误的问题，都是通过自己在网上搜索到解决方案进而得到解决。

**参考文献**

1. 阿里云. vscode安装+配置+使用+调试【保姆级教程】-阿里云开发者社区[J/OL].[2025-02-09]. [https://developer.aliyun.com/article/1475489](https://developer.aliyun.com/article/1475489" \t "https://kimi.moonshot.cn/chat/_blank).
2. 廖雪峰. 编译C程序 - Makefile教程 - 廖雪峰的官方网站[EB/OL]. 廖雪峰的官方网站.[2025-02-09]. <https://liaoxuefeng.com/books/makefile/compile-c/index.html.>
3. 新chenxf. gcc -static 命令[EB/OL]. CSDN博客, 2016[2025-02-09].https://blog.csdn.net/newchenxf/article/details/51743181.
4. 砥才人.【UE4 C++】迷宫生成——DFS、Prim、Kruskal算法实现[J/OL]. 博客园, 2021[2025-02-09]. [https://www.cnblogs.com/shiroe/p/15506909.html](https://www.cnblogs.com/shiroe/p/15506909.html" \t "https://kimi.moonshot.cn/chat/_blank).
5. 弥漫-飘渺.为什么rand()每次产生的随机数都一样\_rand每次产生的随机数都一样-CSDN博客[J/OL]. CSDN, 2014[2025-02-09]. [https://blog.csdn.net/buku2010/article/details/19404885?%3E](https://blog.csdn.net/buku2010/article/details/19404885?>" \t "https://kimi.moonshot.cn/chat/_blank).