



同濟大學  
TONGJI UNIVERSITY

## 数据结构课程设计

### 项目说明文档

## 勇闯迷宫游戏

姓 名： 马小龙

学 号： 2353814

学 院： 计算机科学与技术学院（软件学院）

专 业： 软件工程

指导教师： 张颖

二〇二三年十月三十日

# 目录

第 1 章 项目分析.....	1
1.1 项目背景分析.....	1
1.2 项目需求分析.....	1
1.3 项目功能分析 .....	1
1.3.1 迷宫地图生成功能 .....	2
1.3.2 迷宫寻路功能 .....	2
1.3.3 异常处理功能 .....	2
第 2 章 项目设计.....	3
2.1 数据结构设计.....	3
2.2 结构体与类设计.....	3
2.2.1 链式栈结点 (Node) 设计 .....	3
2.2.1.1 概述.....	3
2.2.1.2 结构体定义.....	3
2.2.1.3 数据成员.....	4
2.2.1.4 构造函数.....	4
2.2.2 链式栈 (Stack) 设计 .....	4
2.2.2.1 概述.....	4
2.2.2.2 Stack 类定义.....	4
2.2.2.3 私有数据成员.....	5
2.2.2.4 构造函数与析构函数.....	5
2.2.2.5 公有成员函数.....	5
2.2.3 Maze 类设计.....	5
2.2.3.1 概述.....	5
2.2.3.2 Maze 类定义.....	6
2.2.3.3 私有数据成员.....	7
2.2.3.4 私有成员函数.....	7
2.2.3.5 构造函数和析构函数.....	8
2.2.3.6 公有成员函数.....	8
2.3 项目框架设计.....	8
2.3.1 项目框架流程图 .....	9
2.3.2 项目框架流程 .....	9
第 3 章 项目功能实现.....	10
3.1 项目主体架构.....	10
3.1.1 实现思路.....	10
3.1.2 流程图.....	10

---

3.1.3 核心代码.....	11
3.1.4 示例.....	11
3.2 迷宫地图生成功能.....	12
3.2.1 实现思路.....	12
3.2.2 核心代码 .....	12
3.2.3 示例 .....	13
3.3 迷宫寻路功能.....	14
3.3.1 实现思路 .....	14
3.3.2 核心代码 .....	15
3.3.3 示例 .....	16
3.4 异常处理功能.....	17
3.4.1 输入非法的异常处理 .....	17
3.4.2 动态内存申请失败的异常处理 .....	17
3.4.3 栈为空的异常处理 .....	18
第4章 项目测试.....	19
4.1 输入迷宫地图大小功能测试.....	19
4.2 生成迷宫地图功能测试.....	19
4.3 迷宫寻路功能测试.....	20
第5章 相关说明.....	22
5.1 编程语言.....	22
5.2 Windows 环境 .....	22
5.3 Linux 环境.....	22

## 第 1 章 项目分析

### 1.1 项目背景分析

迷宫问题自古以来就吸引了人们的注意。作为一种结构复杂且富有挑战性的益智游戏，迷宫不仅在现实生活中用于娱乐和教育，还在虚拟世界中被广泛应用于游戏设计和路径规划。迷宫的概念可以追溯到古希腊神话中的米诺斯迷宫，而随着技术的发展，迷宫被纳入现代计算领域，成为研究路径探索和图形表示的重要对象。

迷宫问题是经典的计算机科学和算法问题之一，它被广泛用于计算机科学和算法学习。它模拟了在复杂环境中找到通路的问题，这可以应用于实际生活中的路径规划、导航系统、游戏开发等领域。

迷宫问题内容包括迷宫设计以及迷宫路径查找等功能，本项目将着重实现这两个功能。

### 1.2 项目需求分析

在一个完整的迷宫游戏中，首先需要关注的是迷宫地图的生成。迷宫由一个  $n \times n$  矩阵表示，其中重要的要素包括起点和终点、障碍物和墙壁，墙壁应随机排列，形成复杂的路径和死胡同，增加游戏的挑战性。有效的地图生成算法能够确保每次游戏都有独特的迷宫布局，从而增强重玩价值。

在迷宫的探索过程中，寻路算法的实现至关重要。玩家在迷宫中冒险的目标是找到从起点到终点的路径，而寻路算法正是帮助他们实现这一目标的关键工具。一个有效的寻路机制不仅能提供解决方案，还能够提升游戏的趣味性和挑战性。

除了迷宫的设计和寻路功能，系统的稳定性和安全性也同样重要。为此，游戏应当实现一个全面的异常处理机制。这一机制首先应包括对用户输入的验证，确保玩家的输入有效且符合预期，避免因输入错误导致游戏崩溃。

### 1.3 项目功能分析

本项目旨在通过迷宫地图生成、迷宫寻路算法、异常处理等功能，以实现迷宫游戏的核心逻辑。

### 1.3.1 迷宫地图生成功能

本程序采用递归的方法生成迷宫地图。其实现思路是：

首先，将迷宫中所有单元格初始化为墙壁。

然后，从起点开始探索，将当前单元格（起点）标记为路径，并随机确定探索的四个方向顺序。这种随机化确保了每次生成的迷宫都有不同的路径结构。

在探索过程中，检查随机方向第二个单元格是否在迷宫范围内且未被访问。如果满足条件，则将第一个单元格标记为路径，连接当前单元格和第二个单元格，并递归调用自身，从第二个单元格开始探索。

当四个方向都探索完毕后，即视为迷宫生成成功。

这种方法通过递归生成路径，可以在生成过程中避免形成环路，使得迷宫呈现出复杂而自然的分支结构，确保任意两个空白单元格之间都有唯一路径。

### 1.3.2 迷宫寻路功能

本程序采用递归的方法寻找迷宫路径，实现思路如下：

从起点出发，依次探索上下左右四个方向，查找是否有通路，当一个方向有通路时，将该方向的第一个单元格作为起点，依次探索除来时方向以外的所有方向，查找是否有通路；

当其余三个方向均没有通路时，则返回上一个起点探索其他方向。

若到达终点，则表明路径查找成功，储存所有路径点；若初始起点四个方向均无法到达终点，则表明迷宫没有到达终点的路径。

### 1.3.3 异常处理功能

实现异常处理机制，处理用户可能输入的非法信息，确保系统的稳定性和安全性。

## 第2章 项目设计

### 2.1 数据结构设计

（一）本项目使用动态分配的 `State` 枚举类型二维数组来存储迷宫地图，主要有以下几个原因：

1. 动态分配使得迷宫的大小可以在运行时根据具体需求进行调整，能够适应不同规模的迷宫设计，而不必固定在编译时定义数组的尺寸。

2. 与静态数组相比，动态分配可以更有效地利用系统内存，避免了过度分配或浪费。只分配所需大小的空间，减少了不必要的内存占用。

3. 动态数组使得项目在未来扩展时更容易处理，例如增加新的功能或调整迷宫的结构。二维数组的结构能够简单地进行扩展或缩减以满足不同的场景和需求。

3. 使用 `State` 枚举类型来标识迷宫的单元状态（`WALL`、`ROAD`、`PATH`），使得代码更具可读性和可维护性，便于理解迷宫中每个位置的状态。

（二）本项目使用栈来储存路径结点，原因如下：

1. 在路径搜索或回溯中，栈的后进先出（`LIFO`）特性非常适合存储路径结点。在本程序中，只有到达终点之后，才开始储存路径点，这使得终点最先被储存，起点最后被储存；输出时起点最先被输出，最后是终点，与栈先进后出的特性非常契合；

2. 栈只存储路径上的点，不会保存已访问但不在当前路径上的点，因此在迷宫探索中更加节省空间，相比于其他结构（如数组），它能够更高效地管理路径；

3. 栈结构简单，操作（如压栈和弹栈）效率高，适合用于路径探索的实现，代码逻辑清晰明了，有助于维护和调试。

### 2.2 结构体与类设计

为了使栈更加通用，本栈为链式栈，且为模板设计。

#### 2.2.1 链式栈结点（`Node`）设计

##### 2.2.1.1 概述

`Node` 用于储存信息，包括结点储存的具体信息内容和下一结点的地址。

##### 2.2.1.2 结构体定义

```
template <typename T>
struct Node {
    T data;
    Node<T>* next;
    Node(Node<T>* ptr = nullptr);
    Node(const T& item, Node<T>* ptr = nullptr);
};
```

### 2.2.1.3 数据成员

T data: 数据域, 存储结点的数据

Node<T>\* next: 指针域, 指向下一个结点的指针

### 2.2.1.4 构造函数

```
Node(Node<T>* ptr = nullptr);
```

构造函数, 初始化指针域。

```
Node(const T& item, Node<T>* ptr = nullptr);
```

构造函数, 初始化数据域和指针域。

## 2.2.2 链式栈 (Stack) 设计

### 2.2.2.1 概述

该通用模板类 Stack 用于表示链式栈。能够实现进栈、弹出、取栈顶元素、获得栈的大小等基本操作; 采用链式栈而不采用顺序栈, 可以动态扩展、缩小栈的大小, 提高内存的利用率。

### 2.2.2.2 Stack 类定义

```
template <typename T>
class Stack {
public:
    Stack();
    ~Stack();
    void Push(const T& x);
    T Pop();
    T getTop();
    bool IsEmpty();
    int getSize();
```

```

    void makeEmpty();
private:
    Node<T>* top;
    int size;
};

```

### 2.2.2.3 私有数据成员

Node<T>\* top: 指向栈顶结点的指针

int size: 栈的大小

### 2.2.2.4 构造函数与析构函数

```
Stack();
```

默认构造函数，将栈顶指针指向 nullptr。

```
~Stack();
```

析构函数，释放链式栈的内存资源，包括所有结点的内存。

### 2.2.2.5 公有成员函数

```
void Push(const T& x);
```

向栈中添加元素。

```
T Pop();
```

弹出栈顶元素，并返回其值。

```
T getTop();
```

获取栈顶元素的值。

```
bool IsEmpty();
```

检查栈是否为空。

```
int getSize();
```

获取栈的大小。

```
void makeEmpty();
```

通过遍历堆栈的结点，释放其内存，并将 top 置为 nullptr 来实现将栈清空。

## 2.2.3 Maze 类设计

### 2.2.3.1 概述

Maze 类是一个用于生成迷宫并实现寻路算法的类。该类提供一个模块化的迷宫生成和寻路解决方案，使用户能够轻松地生成迷宫地图，进行路径搜索，以及获取找到的路径。



## 2.2.3.2 Maze 类定义

```
class Maze {
private:
    const int UP=1;
    const int DOWN=2;
    const int LEFT=3 ;
    const int RIGHT =4;
    const int min_size=5;
    int height;
    int width;
    enum State{WALL,ROAD,PATH};
    struct RouteDot {
        int x;
        int y;
    } Start{ 1,1 }, End{ height - 2,width - 2 };
    State**my_maze;
    Stack<RouteDot> route;
private:
    void Initial();
    void BreakWall(int x1, int y1, int x2, int y2) const;
    bool inBounds(int x, int y) const;
    void DrawMaze(int x,int y);
    bool GoThrough(int x,int y,int direction=0);
    void PrintMaze(bool is_print_route) const;
public:
    Maze();
    Maze (int height, int width);
    ~Maze();
    void setHeight(int height_);
    void setWidth(int width_);
    void MakeEmpty();
    void Draw();
    bool FindPath();
```

```
void PrintRoute();
};
```

### 2.2.3.3 私有数据成员

```
const int UP=1;
```

定义方向“上”

```
const int DOWN=2;
```

定义方向“下”

```
const int LEFT=3;
```

定义方向“左”

```
const int RIGHT =4;
```

定义方向“右”

```
const int min_size=5;
```

定义迷宫最小大小，为5

```
int height;
```

迷宫高度

```
int width;
```

迷宫宽度

```
enum State{WALL,ROAD,PATH};
```

State 枚举类型，用来表示迷宫单元格状态

```
struct RouteDot {
```

```
    int x;
```

```
    int y;
```

```
}Start{ 1,1 }, End{ height - 2,width - 2 };
```

定义结构体表示特定单元格，x 为高，y 为宽；Start{ 1,1 }, End{ height - 2,width - 2 }分别为起点和终点。

```
State**my_maze;
```

动态二维数组，用来表示迷宫；

```
Stack<RouteDot> route;
```

链式栈，用来储存路径点

### 2.2.3.4 私有成员函数

```
void Initial();
```

初始化函数，建立二位数组，并将其所有元素初始置为 WALL；

```
void BreakWall(int x1, int y1, int x2, int y2) const ;
```

打破当前单元格和目标单元格之间的墙，即将其标为 ROAD;

```
bool inBounds(int x, int y) const ;
```

检查当前点是否在迷宫范围之内;

```
void DrawMaze(int x,int y);
```

采用递归的方法生成迷宫;

```
bool GoThrough(int x,int y,int direction=0);
```

采用递归的方法查找可能存在的迷宫路径; 找到返回 true, 未找到返回 false;

```
void PrintMaze(bool is_print_route) const;
```

打印迷宫, 根据参数不同决定是否打印行走路径;

### 2.2.3.5 构造函数和析构函数

```
Maze();
```

构造函数, 将长和高初始化为设定最小值;

```
Maze (int height, int width);
```

含参构造函数, 将长和高设定为对应参数; 若参数不为奇数, 则将其加 1; 若参数小于设定最小值, 则将其设为设定最小值。初始化迷宫。

```
~Maze();
```

析构函数, 释放动态迷宫数组所占内存空间;

### 2.2.3.6 公有成员函数

```
void setHeight(int height_);
```

设置迷宫长度;

```
void setWidth(int width_);
```

设置迷宫宽度;

```
void MakeEmpty();
```

清空动态迷宫数组, 释放其所占内存空间;

```
void Draw();
```

调用私有函数 void DrawMaze(int x,int y)实现迷宫生成;

```
bool FindPath();
```

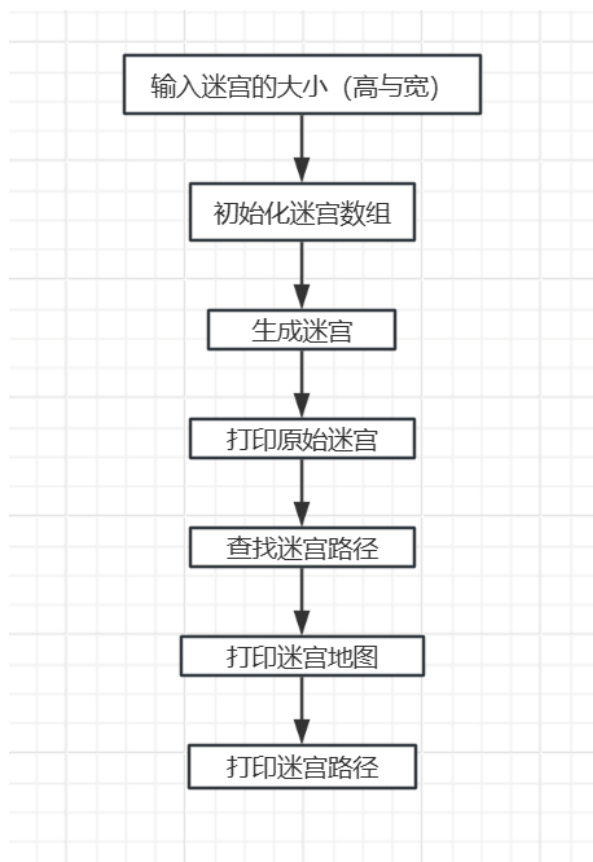
调用私有函数 bool GoThrough(int x,int y,int direction=0)实现路径查找;

```
void PrintRoute();
```

打印迷宫地图 (迷宫和行走路线) 以及路径点。

## 2.3 项目框架设计

### 2.3.1 项目框架流程图



### 2.3.2 项目框架流程

- 1.输入迷宫地图大小；用户被要求输入迷宫的行数和列数必须为奇数
- 2.初始化迷宫对象；使用用户提供的行数、列数，为迷宫数组分配空间，并初始化其元素；
- 3.生成迷宫：采用递归的方法，生成一个迷宫；
- 4.输出原始迷宫地图：输出初始生成的迷宫地图，显示迷宫中的墙壁和通道；
- 5.查找迷宫路径：采用递归的方法，查找迷宫中可能存在的路径
- 6.输出迷宫路径地图与路径：输出包含找到的路线的迷宫地图与找到的路径； 若未找到，输出提示信息。

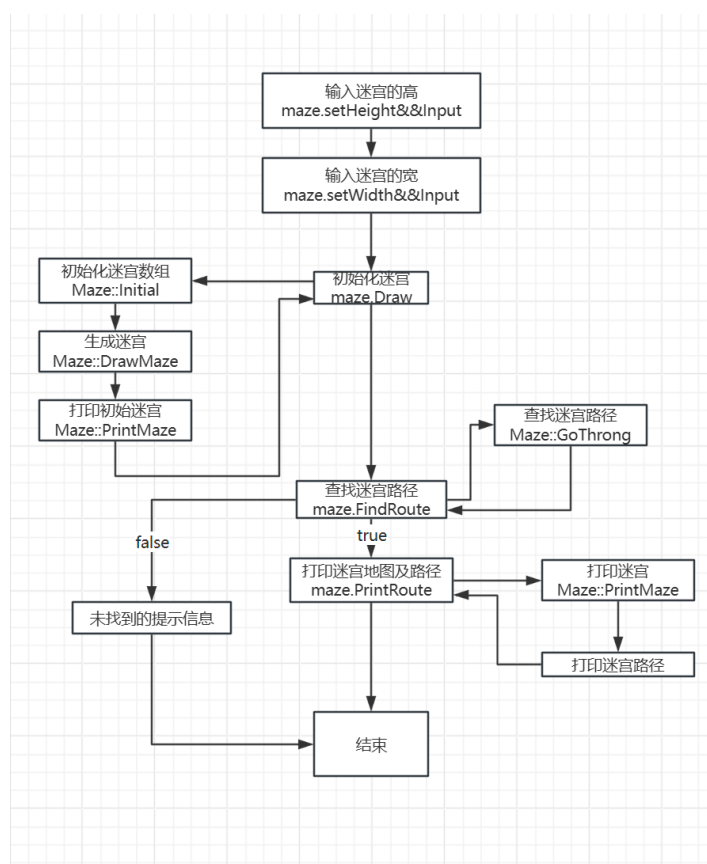
## 第3章 项目功能实现

### 3.1 项目主体架构

#### 3.1.1 实现思路

1. 进入 MazeAdventureGame 函数以进入勇闯迷宫游戏。
2. 调用 Input 函数输入迷宫的高和宽，并将其通过 maze.setHeight 与 maze.setWidth 函数设置迷宫的长和宽；
3. 调用 maze.Draw 函数实现初始化迷宫数组，生成迷宫，打印初始迷宫；
4. 调用 maze.FindRoute 函数查找迷宫路径；
5. 若查找成功，则调用 maze.PrintRoute 函数打印含路线的迷宫地图以及路径；查找失败则打印提示信息；
6. 退出游戏。

#### 3.1.2 流程图



### 3.1.4 示例



## 3.2 迷宫地图生成功能

### 3.2.1 实现思路

借助 DrawMaze 函数，采用递归的方法实现迷宫生成，具体思路如下：

- 1.调用 Initial 函数，初始化迷宫数组，将其所有元素赋值为 WALL；
- 2.然后，从起点开始探索，将当前单元格（起点）标记为路径，并随机确定探索的四个方向顺序；
- 3.探索一个方向时，检查该方向第二个单元格是否在迷宫范围内且为 WALL。如果满足条件，则将第一个单元格标记为 ROAD，连接当前单元格和该方向第二个单元格，并递归调用自身，从该方向第二个单元格开始探索。
- 4.依次随机探索四个方向，当四个都探索完毕后，返回上一层函数，探索其它方向；若该函数为最初的函数，即视为迷宫生成完成。

### 3.2.2 核心代码

```
void Maze::Initial()
{
    Start.x=1;
    Start.y=1;
    End.x=height-2;
    End.y=width-2;
    if(my_maze!=nullptr)
        return;
    my_maze=new State*[height];
    assert(my_maze!=nullptr);
    for(int i=0;i<height;i++) {
        my_maze[i]=new State[width];
        assert(my_maze[i]!=nullptr);
    }
    for(int i=0;i<height;i++)
        for(int j=0;j<width;j++) {
            my_maze[i][j]=WALL;
        }
}
```

---

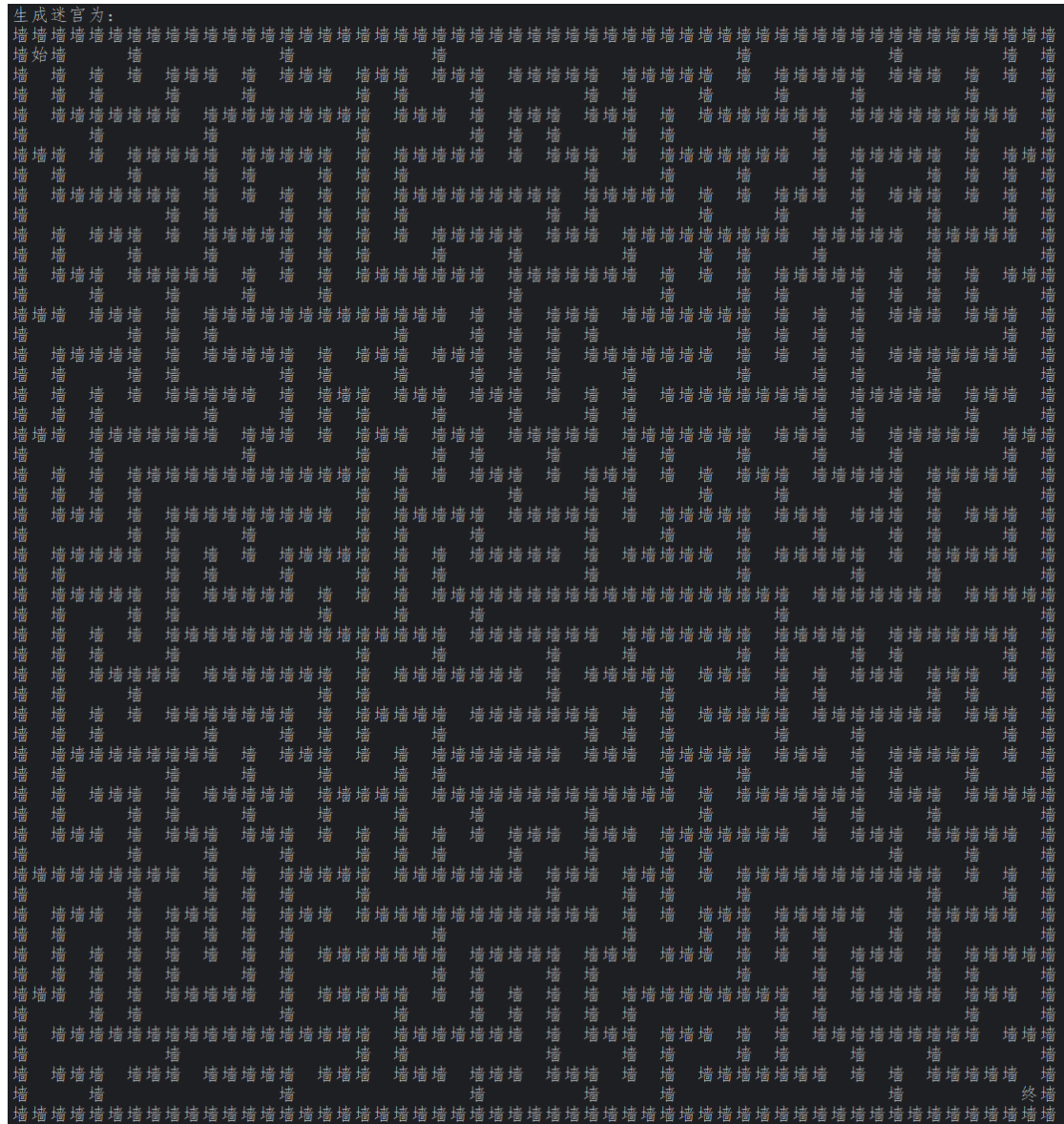
```

void Maze::DrawMaze(int x,int y)
{
    if(my_maze==nullptr)
        Initial();
    my_maze[x][y] = ROAD;
    constexpr int directions[4][2] = { {1, 0}, {-1, 0}, {0, 1}, {0, -1} };
    int dir_order[4] = { UP,DOWN,LEFT,RIGHT };
    dir_order[0] = rand() % 4 + 1;
    do {
        dir_order[1] = rand() % 4 + 1;
    } while (dir_order[1]== dir_order[0]);
    do {
        dir_order[2] = rand() % 4 + 1;
    } while (dir_order[2] == dir_order[1]||dir_order[2] == dir_order[0]);
    dir_order[3] = UP * DOWN * LEFT * RIGHT / dir_order[0] / dir_order[1] /
dir_order[2];
    for (int i : dir_order) {
        int dir = i - 1;
        int nx = x + directions[dir][0]*2;
        int ny = y + directions[dir][1]*2;
        if (inBounds(nx, ny)&&my_maze[nx][ny]==WALL) {
            BreakWall(x, y, nx, ny);
            DrawMaze(nx, ny);
        }
    }
}

```

### 3.2.3 示例





### 3.3 迷宫寻路功能

#### 3.3.1 实现思路

实现迷宫路径查找功能时，通过递归调用 `GoThrough` 函数探索迷宫中的路径。其基本实现思路如下：

1. 从迷宫的起点开始，依次尝试向上、下、左、右四个方向移动，检查这些方向是否存在可行的通路。对于每个探索方向，如果找到可通行的单元格，将该单元格走位新的起点，继续递归探索其除来时方向以外的三个新方向；

2.如果在新的探索中成功到达终点，则用栈储存终点，返回上一层并储存上一层起点，继续返回上一层并执行相同的操作，直到迷宫入口储存在栈中，此时 GoThrough 函数返回 true，表示路径查找成功。

3.如果当前起点的所有探索方向均已尝试但未能通往终点，则返回上一层递归，回溯到上一个起点，继续探索其他未探索的方向；若当前起点为迷宫入口，则返回 false，表明没有可通行路径。

### 3.3.2 核心代码

```
bool Maze::GoThrough(int x,int y,int direction)
{
    RouteDot dot{ x,y };
    if (!(inBounds(x, y)&& my_maze[x][y]==ROAD))
        return false;
    if (x == End.x && y == End.y) {
        route.Push(dot);
        my_maze[x][y] = PATH;
        return true;
    }
    if (direction != DOWN && GoThrough(x - 1, y, UP)) {
        route.Push(dot);
        my_maze[x][y] = PATH;
        return true;
    }
    if (direction != UP && GoThrough(x + 1, y, DOWN)) {
        route.Push(dot);
        my_maze[x][y] = PATH;
        return true;
    }
    if (direction != RIGHT && GoThrough(x, y - 1, LEFT)) {
        route.Push(dot);
        my_maze[x][y] = PATH;
        return true;
    }
    if (direction != LEFT && GoThrough(x, y + 1, RIGHT)) {
```

### 3.3.3 示例

拼音路径:

(1,1)	→ (2,1)	→ (3,1)	→ (3,2)	→ (3,3)	→ (2,3)	→ (1,3)	→ (1,4)	→ (1,5)	→ (2,5)	→ (2,6)	→ (3,6)	→ (4,6)	→ (5,6)	→ (6,6)	→ (7,6)	→ (8,6)	→ (9,6)	→ (9,2)	→ (8,2)	→ (7,2)	→ (6,2)	→ (5,2)	→ (4,2)	→ (3,2)	→ (2,2)	→ (1,2)	→ (1,1)		
(3,5)	→ (4,5)	→ (5,5)	→ (5,4)	→ (5,3)	→ (6,3)	→ (7,3)	→ (8,3)	→ (9,3)	→ (9,4)	→ (8,4)	→ (7,4)	→ (6,4)	→ (5,4)	→ (4,4)	→ (3,4)	→ (2,4)	→ (1,4)	→ (1,5)	→ (2,5)	→ (3,5)	→ (4,5)	→ (5,5)	→ (6,5)	→ (7,5)	→ (8,5)	→ (9,5)	→ (9,1)	→ (8,1)	
(9,1)	→ (10,1)	→ (11,1)	→ (12,1)	→ (13,1)	→ (13,2)	→ (13,3)	→ (14,3)	→ (15,3)	→ (16,3)	→ (17,3)	→ (18,3)	→ (19,3)	→ (20,3)	→ (21,3)	→ (22,3)	→ (23,3)	→ (24,3)	→ (25,3)	→ (26,3)	→ (27,3)	→ (28,3)	→ (29,3)	→ (30,3)	→ (31,3)	→ (32,3)	→ (33,3)	→ (34,3)	→ (35,3)	
(17,3)	→ (17,2)	→ (17,1)	→ (18,1)	→ (19,1)	→ (19,2)	→ (19,3)	→ (20,3)	→ (21,3)	→ (22,3)	→ (23,3)	→ (24,3)	→ (25,3)	→ (26,3)	→ (27,3)	→ (28,3)	→ (29,3)	→ (30,3)	→ (31,3)	→ (32,3)	→ (33,3)	→ (34,3)	→ (35,3)	→ (36,3)	→ (37,3)	→ (38,3)	→ (39,3)	→ (40,3)	→ (41,3)	
(25,1)	→ (25,12)	→ (25,13)	→ (24,13)	→ (23,13)	→ (22,13)	→ (21,13)	→ (21,12)	→ (21,11)	→ (20,11)	→ (19,11)	→ (18,11)	→ (17,11)	→ (16,11)	→ (15,11)	→ (14,11)	→ (13,11)	→ (13,10)	→ (13,9)	→ (12,9)	→ (11,9)	→ (10,9)	→ (9,9)	→ (8,9)	→ (7,9)	→ (7,8)	→ (7,7)	→ (6,7)	→ (5,7)	→ (4,7)
(3,7)	→ (3,8)	→ (3,9)	→ (3,10)	→ (3,11)	→ (3,12)	→ (3,13)	→ (3,14)	→ (3,15)	→ (3,16)	→ (3,17)	→ (3,18)	→ (3,19)	→ (3,20)	→ (3,21)	→ (3,22)	→ (3,23)	→ (3,24)	→ (3,25)	→ (3,26)	→ (3,27)	→ (3,28)	→ (3,29)	→ (3,30)	→ (3,31)	→ (3,32)	→ (3,33)	→ (3,34)	→ (3,35)	
(4,15)	→ (4,16)	→ (4,17)	→ (4,18)	→ (4,19)	→ (4,20)	→ (4,21)	→ (4,22)	→ (4,23)	→ (4,24)	→ (4,25)	→ (4,26)	→ (4,27)	→ (4,28)	→ (4,29)	→ (4,30)	→ (4,31)	→ (4,32)	→ (4,33)	→ (4,34)	→ (4,35)	→ (4,36)	→ (4,37)	→ (4,38)	→ (4,39)	→ (4,40)	→ (4,41)	→ (4,42)	→ (4,43)	
(1,25)	→ (2,25)	→ (3,25)	→ (4,25)	→ (5,25)	→ (6,25)	→ (7,25)	→ (8,25)	→ (9,25)	→ (10,25)	→ (11,25)	→ (12,25)	→ (13,25)	→ (14,25)	→ (15,25)	→ (16,25)	→ (17,25)	→ (18,25)	→ (19,25)	→ (20,25)	→ (21,25)	→ (22,25)	→ (23,25)	→ (24,25)	→ (25,25)	→ (26,25)	→ (27,25)	→ (28,25)	→ (29,25)	
(5,19)	→ (4,19)	→ (3,19)	→ (3,18)	→ (3,17)	→ (4,17)	→ (5,17)	→ (6,17)	→ (7,17)	→ (7,16)	→ (6,16)	→ (5,16)	→ (4,16)	→ (3,16)	→ (2,16)	→ (1,16)	→ (1,15)	→ (2,15)	→ (3,15)	→ (4,15)	→ (5,15)	→ (6,15)	→ (7,15)	→ (8,15)	→ (9,15)	→ (10,15)	→ (11,15)	→ (12,15)	→ (13,15)	
(7,15)	→ (8,15)	→ (9,15)	→ (9,16)	→ (9,17)	→ (10,17)	→ (11,17)	→ (12,17)	→ (13,17)	→ (13,18)	→ (14,18)	→ (15,18)	→ (16,18)	→ (17,18)	→ (18,18)	→ (19,18)	→ (20,18)	→ (21,18)	→ (22,18)	→ (23,18)	→ (24,18)	→ (25,18)	→ (26,18)	→ (27,18)	→ (28,18)	→ (29,18)	→ (30,18)	→ (31,18)	→ (32,18)	
(13,19)	→ (12,19)	→ (11,19)	→ (10,19)	→ (9,19)	→ (8,19)	→ (7,19)	→ (7,19)	→ (7,20)	→ (7,21)	→ (7,22)	→ (7,23)	→ (7,24)	→ (7,25)	→ (7,26)	→ (7,27)	→ (7,28)	→ (7,29)	→ (7,30)	→ (7,31)	→ (7,32)	→ (7,33)	→ (7,34)	→ (7,35)	→ (7,36)	→ (7,37)	→ (7,38)	→ (7,39)	→ (7,40)	
(9,21)	→ (10,21)	→ (11,21)	→ (11,22)	→ (11,23)	→ (12,23)	→ (13,23)	→ (13,23)	→ (13,22)	→ (13,21)	→ (14,21)	→ (15,21)	→ (16,21)	→ (17,21)	→ (18,21)	→ (19,21)	→ (20,21)	→ (21,21)	→ (22,21)	→ (23,21)	→ (24,21)	→ (25,21)	→ (26,21)	→ (27,21)	→ (28,21)	→ (29,21)	→ (30,21)	→ (31,21)	→ (32,21)	
(15,21)	→ (16,21)	→ (17,21)	→ (17,20)	→ (17,19)	→ (17,18)	→ (17,17)	→ (17,16)	→ (17,15)	→ (17,14)	→ (17,13)	→ (17,12)	→ (17,11)	→ (17,10)	→ (17,9)	→ (17,8)	→ (17,7)	→												

## 3.4 异常处理功能

### 3.4.1 输入非法的异常处理

在生成迷宫地图时，由于实现方法问题，使得生成的迷宫地图高和宽都必须为奇数，否则无法正常生成迷宫；因此，对用户输入的长和宽必须为奇数，否则提示输入错误，清除缓冲区，并重新输入，直到用户输入正确的值为止。

```
int Input(const char* str)
{
    int temp;
    double d_temp;
    while(true) {
        std::cout<<"请输入迷宫的"<<str<<"(应当为不小于 5 的正奇数): \n";
        std::cin>>d_temp;
        if(std::cin.fail()||d_temp !=static_cast<int>(d_temp)||d_temp<5||static_cast<int>(d
        _temp)%2==0) {
            std::cout<<"输入不符合要求! \n";
            std::cin.clear();
            std::cin.ignore(1000000000,'\n');
            continue;
        }
        temp=static_cast<int>(d_temp);
        std::cin.clear();
        std::cin.ignore(1000000000,'\n');
        break;
    }
    return temp;
}
```

### 3.4.2 动态内存申请失败的异常处理

在进行 Stack 类与动态迷宫数组 my\_maze 的动态内存申请时，程序使用 new(std::nothrow) 来尝试分配内存。new(std::nothrow)在分配内存失败时不会引发异常，而是 返回一个空指针(NULL 或 nullptr)，代码检查指针是否为空指针，

如果为空指针，意味着内存分配失败，对于内存分配失败，则输出异常信息，并退出程序。示例：

```
void Maze::Initial()
{
    Start.x=1;
    Start.y=1;
    End.x=height-2;
    End.y=width-2;
    if(my_maze!=nullptr)
        return;
    my_maze=new State*[height];
    assert(my_maze!=nullptr);
    for(int i=0;i<height;i++) {
        my_maze[i]=new State[width];
        assert(my_maze[i]!=nullptr);
    }
    for(int i=0;i<height;i++)
        for(int j=0;j<width;j++) {
            my_maze[i][j]=WALL;
        }
}
```

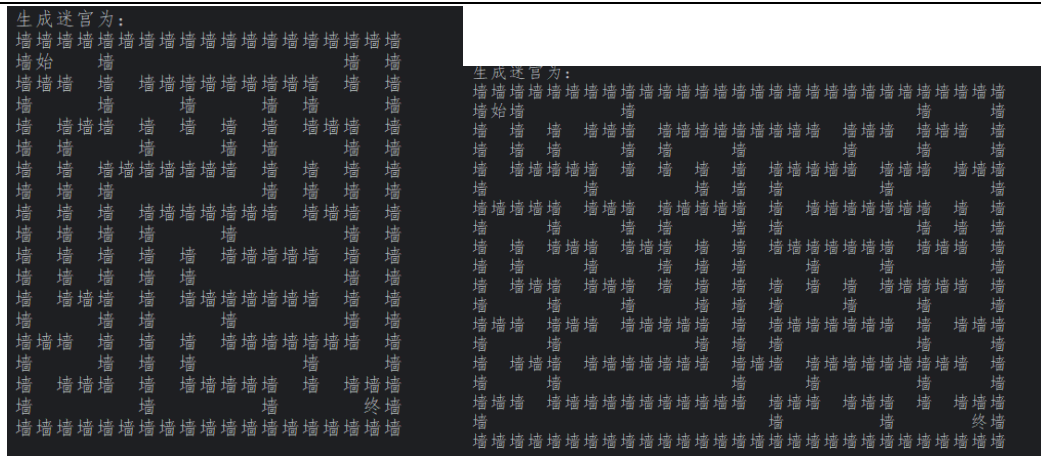
### 3.4.3 栈为空的异常处理

在 Stack 类中，会出现一种错误情况，即栈为空，却调用 Pop 函数试图删除栈顶结点，或调用 getTop 函数获取访问栈顶元素，对于这种情况，会直接输出异常，例如：

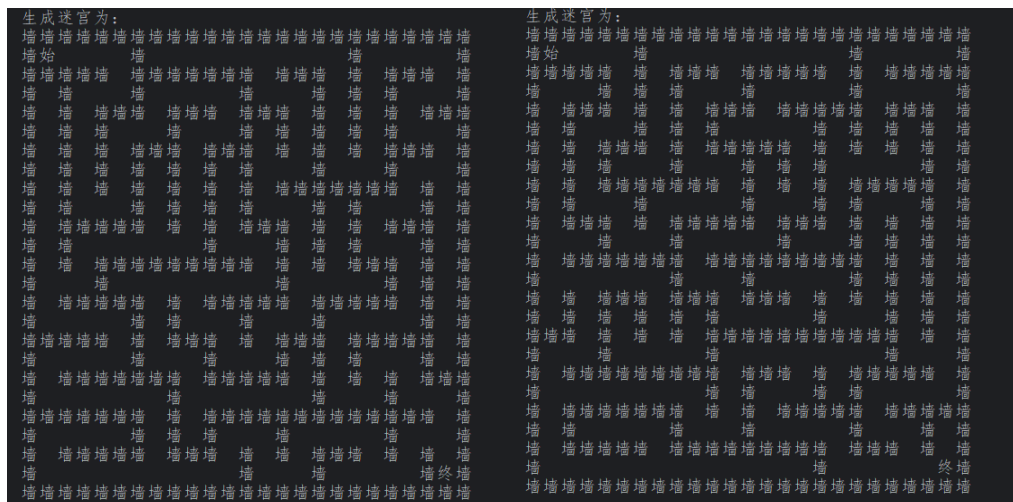
```
template <typename T>
T Stack<T>::getTop(){
    if (IsEmpty()) {
        std::cout << "栈为空,不能取出元素!\n";
        exit(STACK_EMPTY);
    }
    return top->data;
}
```

#### 4.1 输入迷宫地图大小功能测试

可以根据不同的高和宽生成不同的迷宫



## 同一长和宽也可以生成不同的迷宫



### 4.3 迷宫寻路功能测试

可以依据迷宫查找出迷宫路径







## 第 5 章 相关说明

### 5.1 编程语言

本项目全部.cpp 文件以及.h 文件均使用 C++编译完成，使用 UTF-8 编码。

### 5.2 Windows 环境

Windows 系统：Windows 11 x64

Windows 集成开发环境：CLion 2024

工具集：MinGW 11.0 w64

### 5.3 Linux 环境

基于 Linux 内核的操作系统发行版：Ubuntu 24.04.1

Linux 命令编译过程为：

1. 定位包含项目所在文件夹，包括.pp 与.h 文件；具体命令为：cd /home/bruce/programe/ maze\_adventure\_game

2. 编译项目，生成可执行文件；具体命令为：g++ -static -o maze\_adventure\_game\_linux maze\_adventure\_game\_system.cpp my\_stack.h;

其中指令含义分别为：

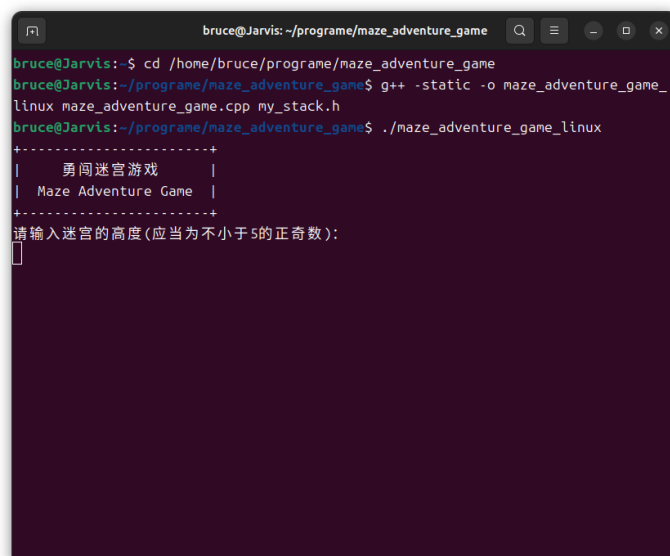
g++: 调用 GNU 的 C++编译器

-static: 使用静态链接而非动态链接，将所有依赖库直接嵌入到可执行文件，文件存储空间变大，但可以单独运行

-o maze\_adventure\_game\_linux: -o 表示输出文件选项，maze\_adventure\_game\_linux 为可执行文件名

maze\_adventure\_game.cpp my\_stack.h:编译所需要的文件。

- 3.运行可执行文件；具体命令为 ./ maze\_adventure\_game\_linux



```
bruce@Jarvis: ~/programe/maze_adventure_game
bruce@Jarvis:~/programe/maze_adventure_game$ g++ -static -o maze_adventure_game_
linux maze_adventure_game.cpp my_stack.h
bruce@Jarvis:~/programe/maze_adventure_game$ ./maze_adventure_game_linux
+-----+
|   勇闯迷宫游戏   |
| Maze Adventure Game |
+-----+
请输入迷宫的高度(应当为不小于5的正奇数):
█
```