项目说明文档

数据结构课程设计

——勇闯迷宫游戏

作 者 姓 名： 杨宇琨

学 号： 2252843

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

# 1 分析

## 1.1 背景分析

迷宫游戏作为一种传统的益智游戏，一直备受玩家喜爱。随着技术的发展，迷宫游戏也逐渐被引入到计算机和移动设备上，成为一种受欢迎的休闲娱乐方式。本项目旨在设计一个具有挑战性和趣味性的迷宫游戏，通过回溯法求解迷宫问题，为玩家提供一个寻找通路、克服障碍的游戏体验。

迷宫游戏的设计将考验玩家的逻辑思维能力和解决问题的能力，同时也为玩家提供了一个锻炼耐心和毅力的机会。通过设计一个精心构思的迷宫，可以让玩家在游戏中感受到探险和冒险的乐趣，增强游戏的互动性和趣味性。

## 1.2 功能分析

1. 迷宫设计： 设计具有一定复杂度和难度的迷宫地图，包括墙壁、通道、入口和出口等元素，确保每个迷宫都是独一无二的。

2. 回溯法求解： 采用回溯法对迷宫问题进行求解，通过试探地搜索前进，受阻时及时回头纠正错误另择通路继续搜索的方法，直到找到出口或无路可走。

# 2 设计

## 2.1 数据结构设计

1. 迷宫表示： 使用二维字符数组`char\*\* maze`表示迷宫，其中`'#'`表示墙壁，`'Y'`表示可探测的空地，空格表示已探索的路径，`'+'`表示起点，`'X'`表示终点。

2. 方向枚举： 使用枚举类型`enum Direction { UP, DOWN, LEFT, RIGHT };`表示四个方向。

3. 迷宫生成递归： 使用递归函数`Maze\_generate`来生成迷宫，其中通过递归调用不断连接可达的空地。

4. 迷宫解决标记： 在解决迷宫路径时，使用字符`'0'`表示未探索的路径，字符`'.'`表示正确路径，字符`'\*'`表示已经走过的路径。

5. 用户输入处理： 使用`input`模板函数处理用户输入，确保输入的行数和列数满足要求。

6. 迷宫内存管理： 使用动态分配的二维字符数组来表示迷宫，需要在适当的时候释放内存，避免内存泄漏。

## 2.2 算法设计

1. 递归迷宫生成： 通过递归调用`Maze\_generate`函数，使用回溯法生成迷宫路径，连接可达的空地，确保生成的迷宫有唯一通路。

2. 迷宫解决算法： 使用递归函数`solution`来解决迷宫路径，通过递归调用不断检查各个方向的可通行性，找到通往终点的正确路径。

3. 随机方向选择： 在迷宫生成和解决过程中，通过`shuffleArray`函数对四个方向进行随机打乱，确保迷宫生成和解决的随机性和多样性。

## 2.3 系统设计

1. 用户交互设计： 通过命令行界面与用户交互，提示用户输入迷宫的行数和列数，显示迷宫生成过程和解决结果，为用户提供友好的交互体验。

2. 迷宫生成模块： 包括迷宫初始化、迷宫绘制和迷宫生成算法，负责生成具有一定挑战性的迷宫地图。

3. 迷宫解决模块： 包括迷宫路径解决算法、路径显示和最佳路径输出，负责解决迷宫路径并展示最佳路径。

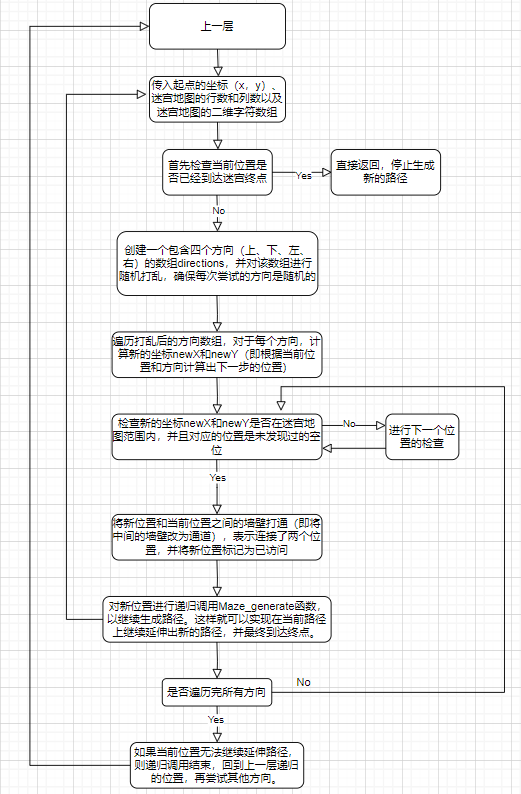
4. 内存管理模块： 负责迷宫内存的动态分配和释放，避免内存泄漏和内存溢出。

5. 系统交互模块： 负责用户输入处理、界面清空、暂停等交互操作，保证系统的交互性和友好性。

# 3 实现

## 3.1 迷宫生成功能的实现

### 3.1.1 迷宫生成功能流程图



### 3.1.2 迷宫生成功能核心代码

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name:Maze\_generate

Function:To generate a random maze from an original maze

Input Parameters:int x, int y, const int rows, const int columns,char \*\*maze

Return Value:void

Documentation：

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Maze\_generate(int x, int y, const int rows, const int columns,char \*\*maze)

{

//If we reached the terminal spot ,stop generating a new path.Because we only want one path that can lead to the end of maze

if (x == rows - 2 && y == columns - 2)

return;

int directions[4] = {UP, DOWN, LEFT, RIGHT}; //Create a vector to hold four directions

shuffleArray(directions, 4); //Disorder four elements in the array

//Handle different directions for each iteration.

for (auto dir : directions) {

int newX = x;

int newY = y;

switch (dir) {

case UP:

newX = x - 2;

break;

case DOWN:

newX = x + 2;

break;

case LEFT:

newY = y - 2;

break;

case RIGHT:

newY = y + 2;

break;

}

//Only if a spot is undiscovered before,can it be connected to the current spot

if (isInside(newX, newY, rows, columns) && maze[newX][newY] == 'Y') {

maze[(x + newX) / 2][(y + newY) / 2] = ' ';//Clear the new spot

maze[newX][newY] = ' ';//Connect the spot with the current spot

Maze\_generate(newX, newY, rows, columns, maze);//Entering into the new spot so we can connect more spot

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name:Maze\_generate

Function:To generate a random maze

Input Parameters:const int rows, const int columns

Return Value:char \*\*

Documentation：

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

char\*\* Maze\_init(const int rows, const int columns)

{

//Allocate space for the maze

char\*\* maze = new(std::nothrow) char\* [rows];

if (maze == NULL) {

cerr << "内存申请失败" << endl;

return NULL;

}

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

maze[i] = new(std::nothrow) char[columns];

if (maze[i] == NULL) {

cerr << "内存申请失败" << endl;

return NULL;

}

}

//Generate the original maze like this

//###############

//#Y#Y#Y#Y#Y#Y#Y#

//###############

//#Y#Y#Y#Y#Y#Y#Y#

//###############

//#Y#Y#Y#Y#Y#Y#Y#

//###############

//#Y#Y#Y#Y#Y#Y#Y#

//###############

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

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

if (!(i % 2))

maze[i][j] = '#';

else

if (j % 2)

maze[i][j] = 'Y';

else

maze[i][j] = '#';

}

}

maze[start\_x][start\_y] = '+';//Marks the start

Maze\_generate(start\_x, start\_y, rows, columns, maze);//Generate

maze[rows - 2][columns - 2] = 'X';//Marks the end

return maze;

}

//Define four directions

enum Direction { UP, DOWN, LEFT, RIGHT };

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name:shuffleArray

Function:Shuffle the array

Input Parameters:array and size

Return Value:void

Documentation：

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void shuffleArray(int arr[], int size)

{

//Using the random device

std::random\_device rd;

std::mt19937 gen(rd());

//from the last element

for (int i = size - 1; i > 0; --i) {

//Generate a random number ranging 0-i

std::uniform\_int\_distribution<> dis(0, i);

int j = dis(gen);

//Exchange the positon randomly

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name:isInside

Function:Check if the coordinate is inside

Input Parameters:const int x, const int y, const int rows, const int columns

Return Value:True for yes,false for no

Documentation：

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

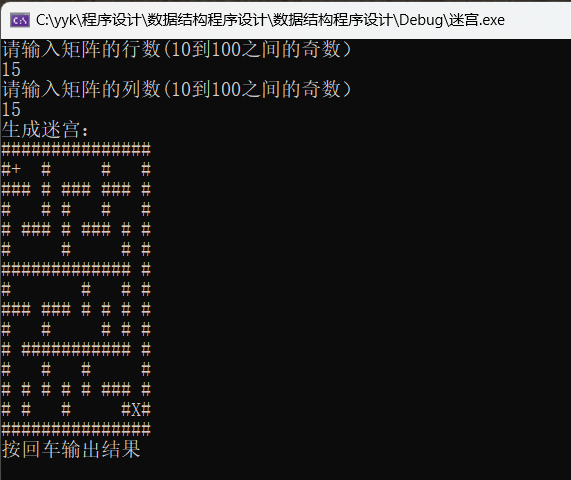
bool isInside(const int x, const int y, const int rows, const int columns)

{

return (x >= 0 && x < rows && y >= 0 && y < columns);

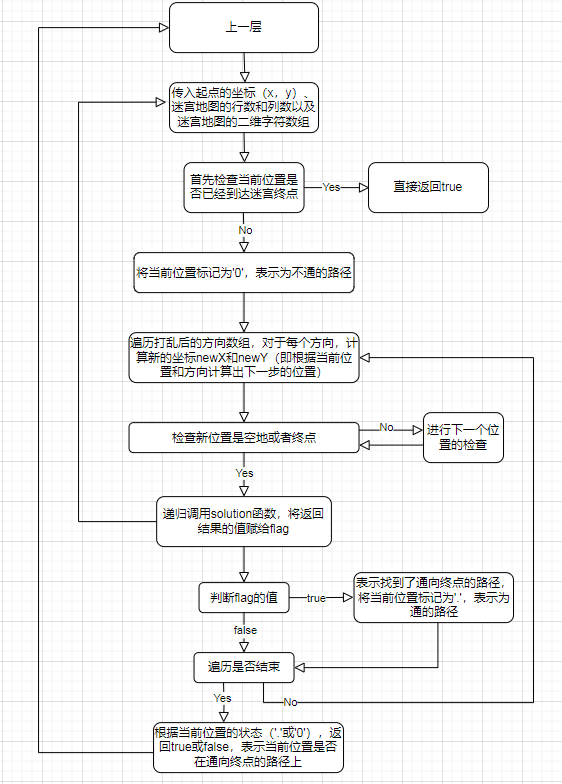
}

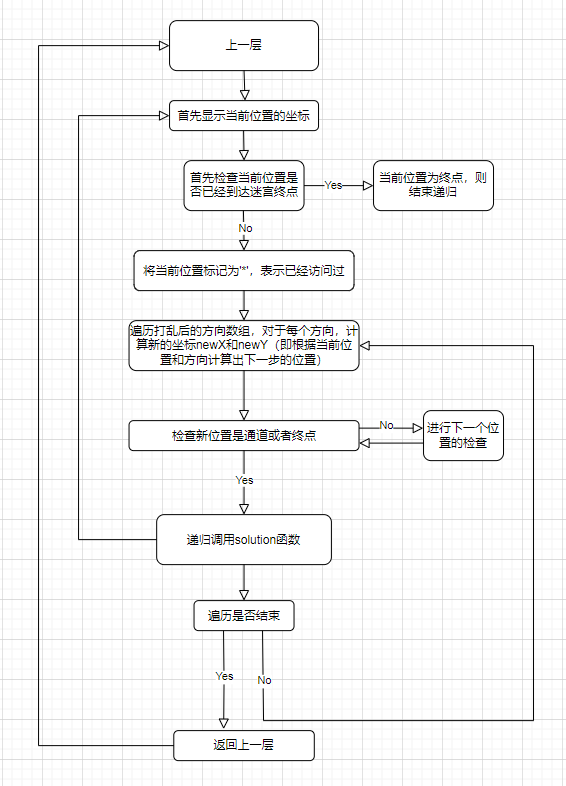
### 3.1.3 迷宫生成功能截屏示例



## 3.2 搜索出口功能的实现

### 3.2.1 搜索出口功能流程图





### 3.2.2 搜索出口功能核心代码

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name:solution

Function:To solve the maze by checking every path

Input Parameters:int x, int y,const int rows, const int columns, char\*\* maze,bool flag

Return Value:whether the spot we are checking is a right way

Documentation：When we are checking a new spot ,we defaultly regard it as a

'wrong' way(Because only we reached the real terminal can we go back to the

spot we passed and tell that is the right spot),so the flag is false and we

make the spot 0 defaultly.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool solution(int x, int y,const int rows, const int columns, char\*\* maze,bool flag)

{

//Reach the terminal and return true

if (x == rows - 2 && y == columns - 2) {

return true;

}

//Before checking the next spot ,we regard the current spot is the wrong way

maze[x][y] = '0';

int directions[4] = { UP, DOWN, LEFT, RIGHT }; //Create a vector to hold four directions

//Handle different directions for each iteration.

for (auto dir : directions) {

int newX = x;

int newY = y;

switch (dir) {

case UP:

newX--;

break;

case DOWN:

newX++;

break;

case LEFT:

newY--;

break;

case RIGHT:

newY++;

break;

}

if (maze[newX][newY] == ' '|| maze[newX][newY] == 'X') {

//This flag can be passed to a new solution so we cannot explore a new place after flaging

flag = solution(newX, newY, rows, columns, maze, flag);

}

else

continue;

//Why this continue makes a difference before we add'flag = false;'.

//Because if we passed the flag as True ,its okay when we reached a dead end

//And the dead end spot will not run the codes below(maze[x][y] = '.';)

//So after the loop is over ,the dead end spot returns false

//The former spot that connects to the dead end spot which should be '.' as we mistakenly passed the True flag

//will change its incorrect flag due to the 'flag = solution(newX, newY, rows, columns, maze, flag);'is run before

//the if (flag == true) ,so we changed the wrong flag and double mistakes make it right, it is really dangerous >\_<

if (flag == true) {

//Change the current spot if a adjoined spot is the

maze[x][y] = '.';

//!!Necessary!!If the flag is not changed to false,the TRUE could be passed to next solution,and thats not we want

flag = false;

}

}

if (maze[x][y] == '.')

return true; //If the current spot is '.',then tell the former spot that it is on the right path

else

return false; //Otherwise it is on the wrong path

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name:solution\_display

Function:To display the right path

Input Parameters:int x,int y,const int rows, const int columns, char\*\* maze

Return Value:void

Documentation：

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void solution\_display(int x,int y,const int rows, const int columns, char\*\* maze)

{

//Display the current spot

cout << '<' << x << ',' << y << '>';

if (maze[x][y] == 'X')

return;

else

cout << "--->";

//Mark the spot as read ,we only read it once

maze[x][y] = '\*';

int directions[4] = { UP, DOWN, LEFT, RIGHT }; //Create a vector to hold four directions

shuffleArray(directions,4); //Disorder four elements in the array

//Handle different directions for each iteration.

for (auto dir : directions) {

int newX = x;

int newY = y;

switch (dir) {

case UP:

newX--;

break;

case DOWN:

newX++;

break;

case LEFT:

newY--;

break;

case RIGHT:

newY++;

break;

}

//Find the next right spot and display it

if (maze[newX][newY] == '.'|| maze[newX][newY] == 'X') {

solution\_display(newX, newY, rows, columns, maze);

return;

}

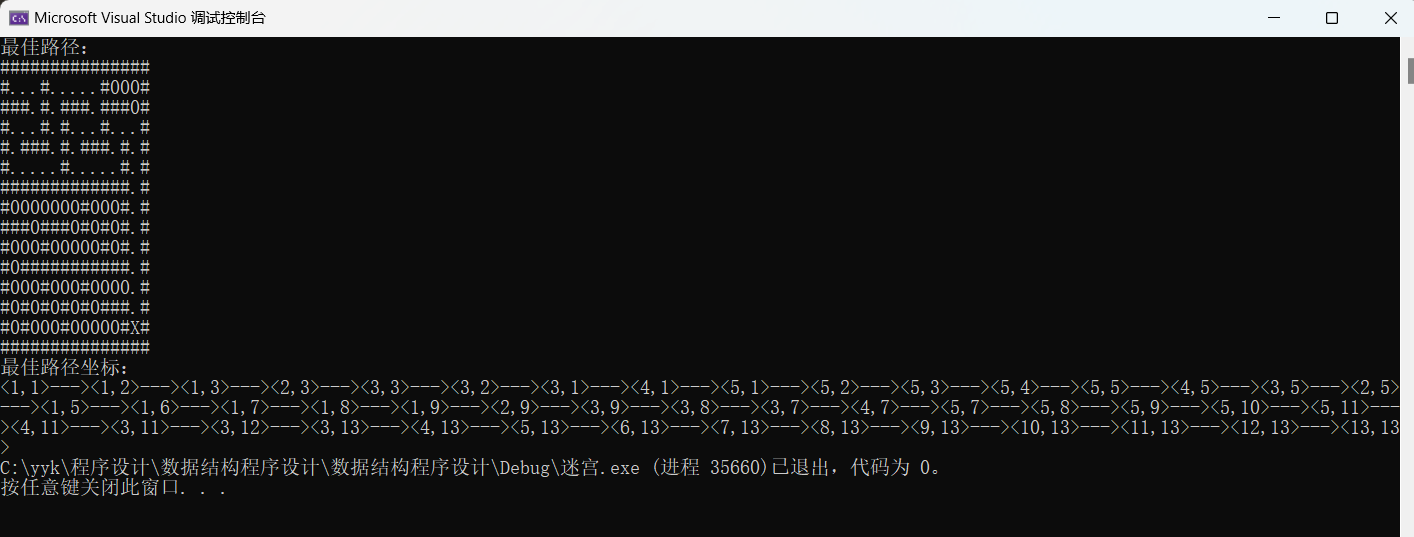
else

continue;

}

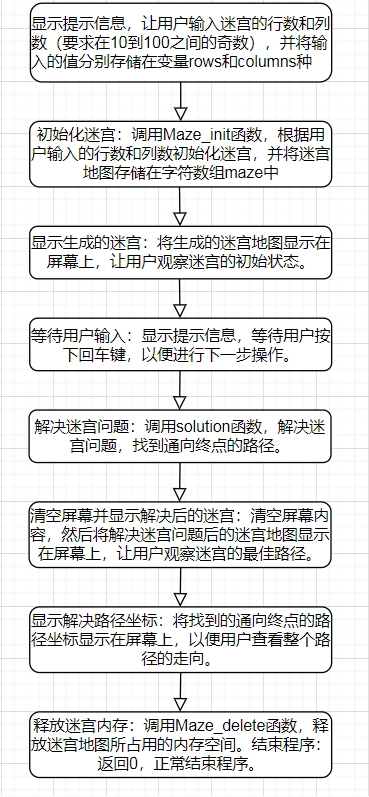
### }

### 3.2.3 搜索出口功能截屏示例



## 3.3 总体系统的实现

### 3.3.1 总体系统流程图



### 3.3.2 总体系统核心代码

int rows, columns;

//Display the prompt and let the client input

cout << "请输入矩阵的行数(10到100之间的奇数）" << endl;

rows = input<int>();

std::cout << "请输入矩阵的列数(10到100之间的奇数）" << endl;

columns = input<int>();

//Initialize the maze and display it on the screen

char\*\* maze = Maze\_init(rows, columns);

if (maze == NULL)

return 0;

cout << "生成迷宫：" << endl;

Maze\_display(rows, columns, maze);

//Wait for the enter

cout << "按回车输出结果" << endl;

pause();

//Solve the maze

solution(start\_x, start\_y, rows, columns, maze, false);

//Clear the screen and display the new maze

system("cls");

cout << "最佳路径：" << endl;

Maze\_display(rows, columns, maze);

//Display the solution path

cout << "最佳路径坐标：" << endl;

solution\_display(start\_x, start\_y, rows, columns, maze);

//Delete the maze

Maze\_delete(rows, maze);

### return 0;

### 3.6.3 总体系统截屏示例

