项目说明文档

**数据结构课程设计**

**——勇闯迷宫游戏**

作 者 姓 名： 吴英豪

学 号： 1953608

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 项目简介

迷宫只有两个门，一个门叫入口，另一个门叫出口。一个骑士骑马从入口进入迷宫，迷宫设置很多障碍，骑士需要在迷宫中寻找通路以到达出口。

1.2功能分析

迷宫问题的求解过程可以采用回溯法即在一定的约束条件下试探地搜索前进，若前进中受阻，则及时回头纠正错误另择通路继续搜索的方法。从入口出发，按某一方向向前探索，若能走通，即某处可达，则到达新点，否则探索下一个方向；若所有的方向均没有通路，则沿原路返回前一点，换下一个方向再继续试探，直到所有可能的道路都探索到，或找到一条通路，或无路可走又返回入口点。在求解过程中，为了保证在达到某一个点后不能向前继续行走时，能正确返回前一个以便从下一个方向向前试探，则需要在试探过程中保存所能够达到的每个点的下标以及该点前进的方向，当找到出口时试探过程就结束了。

**由于课程要求，迷宫已经提前设置完成,迷宫的路径也在可执行文件中显示了出来,方便读者的阅读。此迷宫的大小为21\*21**

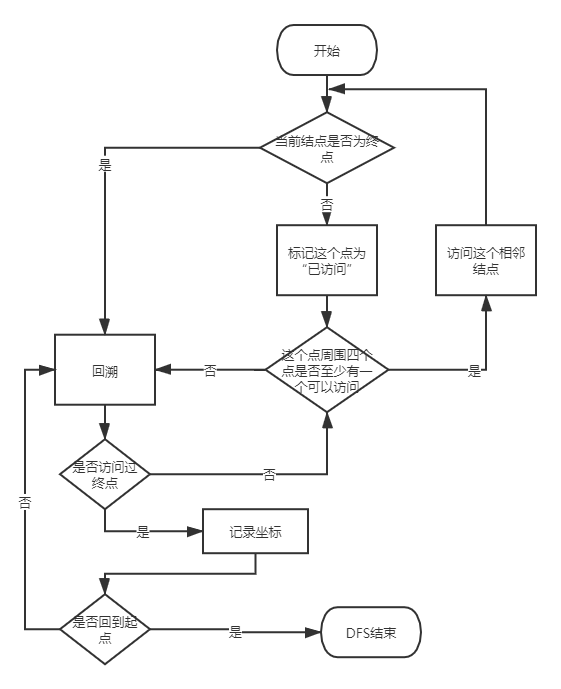
2.设计

2.1 算法设计

2.1.1 文字表述

此迷宫从入口到出口的路径是唯一的，并且考虑到效率，需要用户最快得到路径，所以笔者采用基于递归的深度搜索(dfs)算法。从入口开始,按顺序访问其四周的结点,若出界或是障碍物或者是已经被访问过，则访问其他的相邻结点；若其相邻的结点没有被访问，则标记这个点已被访问,再用DFS访问这个点。当访问的点为出口的时候，开始回溯，并把结点坐标保存起来形成路径。

2.2.2 算法流程图



2.2数据结构设计

该系统为一个平面迷宫系统，需要从入口进入通过一定的算法找到从入口到出口的路径。所以使用**(1)二维数组maze** 来存储迷宫,同时使用DFS的算法,为了保证地图上的每一个点不会被多次访问，故引入**(2)二维数组 maze\_map** 来判断迷宫maze上的点是否已经被访问过。除了走迷宫以外,还需要输出路径,在迷宫中路径唯一，且DFS算法是通过递归实现的,所以使用**(3)栈 \_stack** 来存储路径坐标，其后进先出的特点恰好和递归的回溯特点相符合。

2.3 类结构设计

(1)Maze类

用来表示迷宫以及与迷宫相关的信息

(2) StackNode 类

栈结点,用于存储路径坐标

(3) Stack类

栈类,通过栈结点来存储,增删栈结点

注：因为栈Stack需要访问栈结点StackNode中的成员(尤其是私有成员)，所以将栈Stack设置为了栈结点StackNode的友元

2.4 成员与操作设计

**(1)全局的常量（迷宫相关参数）**

//迷宫总边长

const int maze\_size = 21;

//围墙（方块）边长

const int block\_size = 25;

//初始位置

const int left\_side = 75;

const int right\_side = 100;

const int top\_side = 10;

const int bottom\_side = 35;

const int init\_left = 75;

const int init\_top = 10;

const int dir\_x[4] = { 0,0,1,-1 };

const int dir\_y[4] = { 1,-1,0,0 };

**(2)Maze类**

class Maze

{

private:

//建立迷宫

T\*\* maze;

T\*\* maze\_map;

Stack<int> \_stack; //用于存放最终的坐标

//迷宫坐标

int \_left = left\_side;

int \_top = top\_side;

int \_right = right\_side;

int \_bottom = bottom\_side;

//出口坐标

int out\_x = 20;

int out\_y = 19;

protected:

void setMaze();

bool isInRange(int & x,int & y); //判断搜索坐标是否在界内

public:

//入口坐标

const int in\_x=1;

const int in\_y = 0;

Maze();

~Maze();

void printPath(); //打印最终的路径

void runMaze(int x,int y); //DFS搜索路径

void printMaze(); //打印迷宫

};

**(3)StackNode类**

template<class T>

class StackNode //栈的结点

{

private:

T \_x;

T \_y;

StackNode<T>\* \_next;

public:

StackNode();

StackNode(T x, T y);

~StackNode();

friend class Stack<T>;

};

**(4)Stack类**

template<class T>

class Stack

{

private:

StackNode<T>\* \_top; //栈顶指针

bool \_flag; //目标位置是否入栈

public:

Stack();

~Stack();

void push(T x,T y); //入栈操作

void pop(T &x,T &y); //出栈操作

void setFlag(bool flag); //设置当前结点是否被访问过

bool getFlag()const; //当前结点没有被访问过

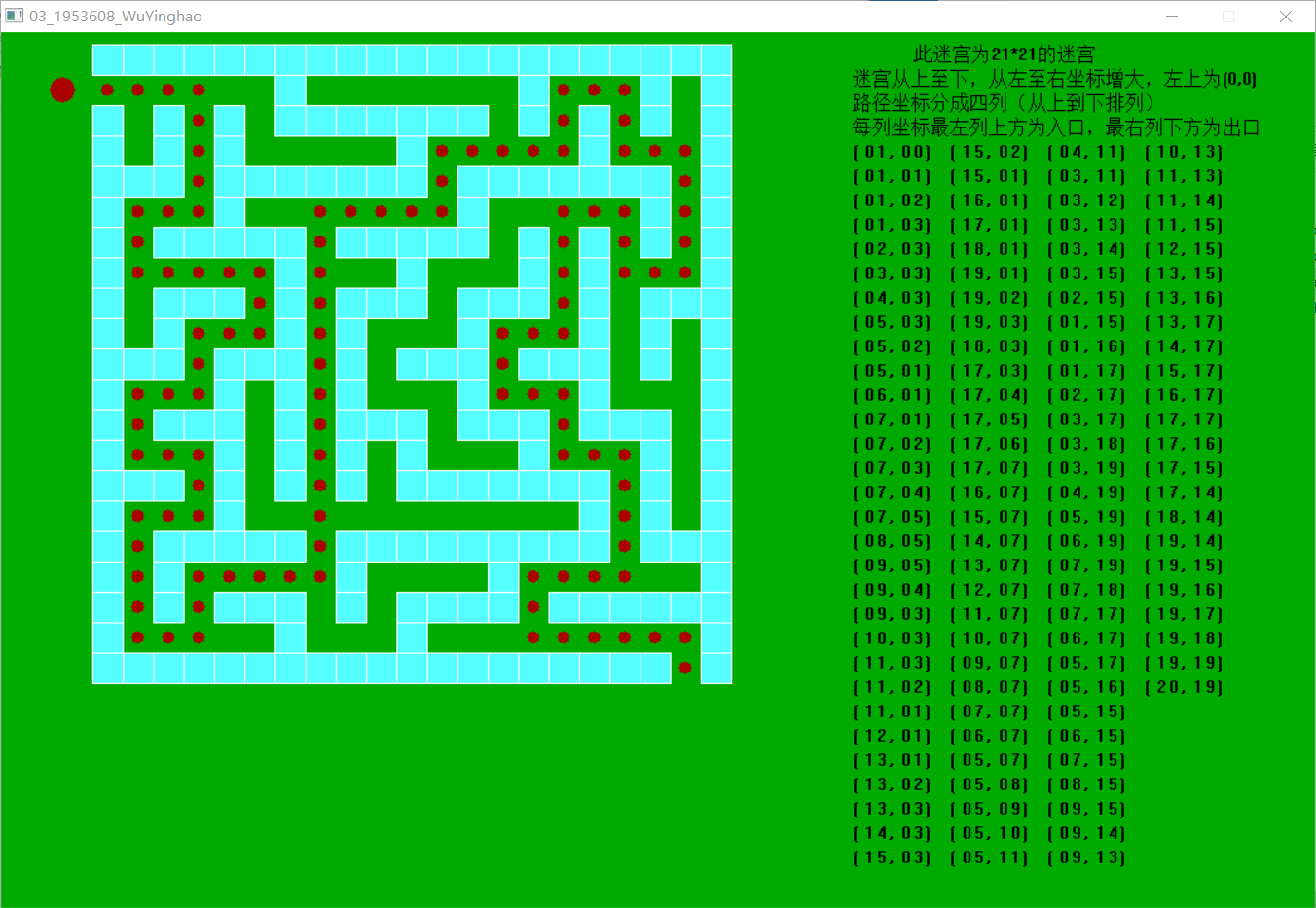
int getLength()const; //返回栈的长度

};

**2.5 特别说明**

在此次设计中,笔者设计的迷宫为21\*21的迷宫,出于方便用户查看迷宫，寻找路径的目的，笔者使用了开源的C++的库easyx.**easy库完全开源,其源代码在03\_1953608\_WuYinghao文件夹中的easyx.h和graphics.h两个头文件中.**

最终效果如下:



用户若想通过03\_1953608\_WuYinghao.cpp和03\_1953608\_WuYinghao.h编译运行生成可执行文件。可以

(1)在03\_1953608\_WuYinghao.h中修改文件路径



(2)将两个头文件graphics.h和easyx.h放在IDE对应的include文件中,在代码中直接引用(#include)即可

①Visual Studio 2019

将两个文件放在VS2019安装目录的\VC\Auxiliary\VS\include中

②Visual C++6.0

将两个文件放在Visual C++6.0安装目录的

\Microsoft Visual Studio\VC98\Include中

2.6 文件设计

**由于此系统使用了easyx库，此库调用了文件<windows.h>所以此系统仅支持Windows平台，不支持Linux平台。**

**本系统文件放在03\_1953608\_WuYinghao\_Windows文件夹中。**

本系统的源代码放在两个文件(03\_1953608\_WuYinghao.h和03\_1953608\_WuYinghao.cpp)中，其中类的定义以及类成员函数定义均放在了03\_1953608\_WuYinghao.h中,main函数则放在了03\_1953608\_WuYinghao.cpp中

两个文件的关系为03\_1953608\_WuYinghao.cpp

包含03\_1953608\_WuYinghao.h

**其中,03\_1953608\_WuYinghao.cpp和03\_1953608\_WuYinghao.h的编码格式均为简体中文(GB2312)-代码页 936**

3.实现

3.1深度搜索（DFS）功能的实现

**3.1.1 功能说明**

通过递归的方式来查找路径,再回溯的时候保存路径坐标(详细内容见2.1算法设计)

**3.1.2 DFS核心代码**

template<class T>

void Maze<T>::runMaze(int x,int y)

{

if (out\_x == x && out\_y == y) //找到出口

{

\_stack.push(x, y);

\_stack.setFlag(true);

return;

}

else

{

maze\_map[x][y] = '1'; //这个点已经被找到过

for (int i = 0; i < 4; i++)

{

if (\_stack.getFlag())

break;

int next\_x = x + dir\_x[i];

int next\_y = y + dir\_y[i];

if (isInRange(next\_x, next\_y) && (maze\_map[next\_x][next\_y]=='0'))//下一个点没有出界并且没有被访问过

{

runMaze(next\_x, next\_y);

if (\_stack.getFlag()) //已经找到出口

{

\_stack.push(x, y);

}

}

}

}

}

3.2判断边界功能的实现

**3.2.1 功能说明**

对于迷宫中的坐标(x,y)，分析其周围的四个点(x,y+1) (x,y-1) (x+1,y) (x-1,y).若这个点上为障碍物或这个点超出边界或这个点已经被访问过，返回false；若没有被访问过,返回true;

isInRange（x,y）函数在类 Maze的protected部分,所以用户不能直接调用,只有Maze类或者其子类才能够调用

3.2.2核心代码展示

template<class T>

bool Maze<T>::isInRange(int& x, int& y)

{

if ((x >= 0) && (x < maze\_size) && (y >= 0) && (y < maze\_size)&&(maze[x][y]=='\*')) //可以被访问

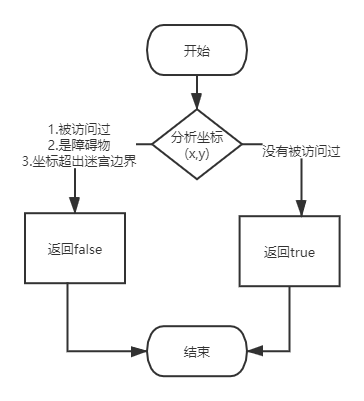
return true;

else

return false;

}

3.2.3 流程图展示



3.3画迷宫

**3.3.1 功能说明**

文件中已经设置了迷宫边界块的长度与宽度(全局const变量，不可修改)

const int block\_size = 25; //围墙（方块）边长

使用easyx库内置的画矩形函数,画矩形方块来实现迷宫。

3.3.2 核心代码展示

template<class T>

void Maze<T>::printMaze()

{

for (int i = 0; i < maze\_size; i++)

{

for (int j = 0; j < maze\_size; j++)

{

if (maze[i][j] == '#')

{

setfillcolor(LIGHTCYAN); //设置颜色

fillrectangle(\_left, \_top, \_right, \_bottom); // 画矩形

}

\_left += block\_size;

\_right += block\_size;

}

\_left = left\_side;

\_right = right\_side;

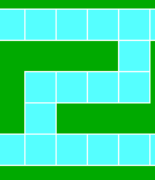
\_top += block\_size;

\_bottom += block\_size;

}

}

3.3.3 结果展示(截图)



3.4画路径

3.4.1 功能说明

由于在DFS搜索的是时候已经将路径结点存放在栈中，且由于栈后进先出的特点以及递归的特点，栈顶的元素(最后入栈的元素)为起点的坐标，栈底的元素(最先入栈的结点)为终点的坐标.

利用循环操作,对于有n个元素的栈,执行n-1次循环，当栈不会空的时候，弹出栈顶元素,在对应坐标上画一个圆点(便于用户观看)；同时在屏幕上展示当前点的坐标.

画圆的操作使用easyx库中的solidcircle(x,y)函数.输出坐标的操作使用了easyx库中的outtextxy()函数。

3.4.2 核心代码展示

template<class T>

void Maze<T>::printPath()

{

setcolor(BLACK);

outtextxy(750, 10, \_T("此迷宫为21\*21的迷宫"));

outtextxy(700, 30, \_T("迷宫从上至下，从左至右坐标增大，左上为(0,0)"));

outtextxy(700, 50, \_T("路径坐标分成四列（从上到下排列）"));

outtextxy(700,70,\_T("每列坐标最左列上方为入口，最右列下方为出口"));

int x = 0, y = 0, loc\_x=0, loc\_y =0, length = \_stack.getLength();

const int interval\_x = 10;

const int interval\_y = 20;

char words[20]="(00,00)";

int locx = 700, locy = 90, tempx = 700;

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

{

\_stack.pop(x,y);

loc\_x = y \* block\_size + (block\_size / 2) + init\_left; //画路径

loc\_y = x \* block\_size + (block\_size / 2) + init\_top;

setfillcolor(RED); **//设置圆圈的颜色为红色**

solidcircle(loc\_x, loc\_y, 5); **//通过画圆画出坐标**

words[1] = x / 10 + '0';

words[2] = x % 10 + '0';

words[4] = y / 10 + '0';

words[5] = y % 10 + '0';

//写出路径

for (int i = 0; i<strlen(words); i++)

{

outtextxy(locx, locy, words[i]); //输出坐标

locx += interval\_x;

}

locy += interval\_y;

if (locy >= 680)

{

locy = 90;

tempx+=80;

}

locx = tempx;

}

}

3.4.3 结果展示(截图)

