中 国 矿 业 大 学

2018 级《数据结构与算法分析》课程设计

学生姓名 王茂凯

学　　号 04181425

题　　目 迷宫通行

中国矿业大学信控学院

二 0 二 0 年十二月

目录

迷宫通行

摘要

迷宫通行输入任意大小的迷宫，控制小人设置起点、终点、障碍。按下演示按键后，在屏幕上显示出一条走出迷宫的路径。

使用到的图形库: EasyX 2020 版

开发环境: Windows 10

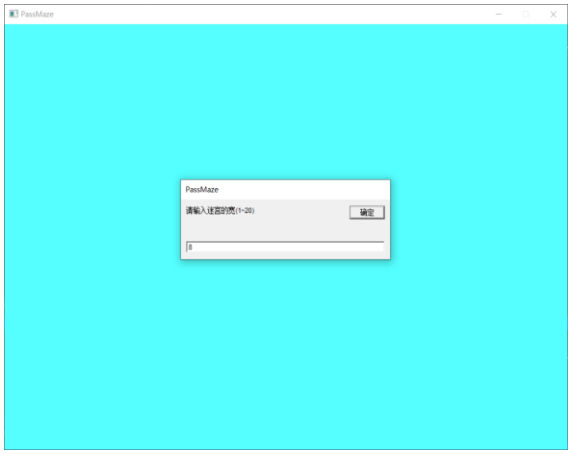
开发工具: Visual Studio 2019

源码地址: https://github.com/leomaokai/PassMaze

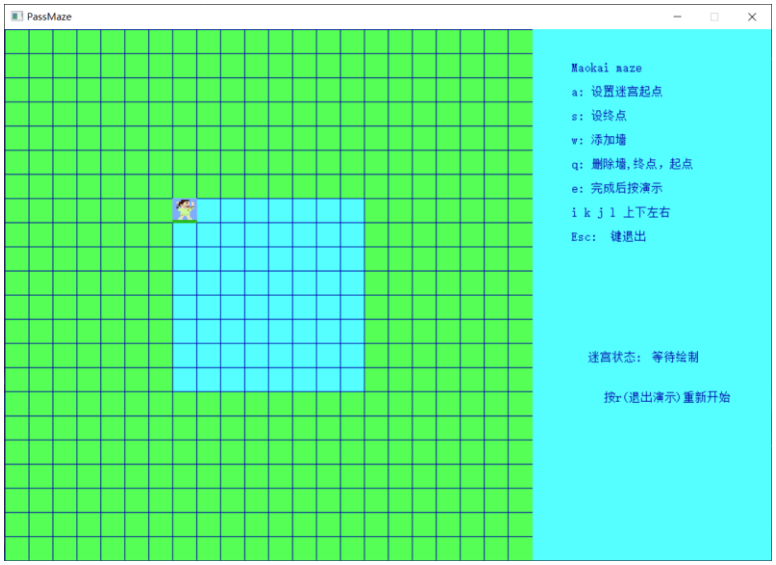
功能要求

- 输入任意大小的迷宫,任设起点,终点,障碍,用栈求出一条迷宫路径,并显示在屏幕上
- 根据用户界面提示，用键盘输入。home 键设置迷宫起点，end 键设终点，上下左右箭头键移动，enter 键添加墙，del 键删除墙，完成后按 F9 键演示，Esc 键退出。(为了方便操作与键盘兼容,a 键设置起点,s 键设终点,w 键添加墙,q 键删除墙,e 键开始演示)
- 橙色的实心小圆圈表示起点，绿色实心圆圈表示终点，空心圆圈表示足迹，红色方块表示墙。
- 未设起点或终点时，会显示“error”，找到路径时，屏幕显示足迹，并在消息框出现 Path found，否则消去足迹，显示 Path not found.

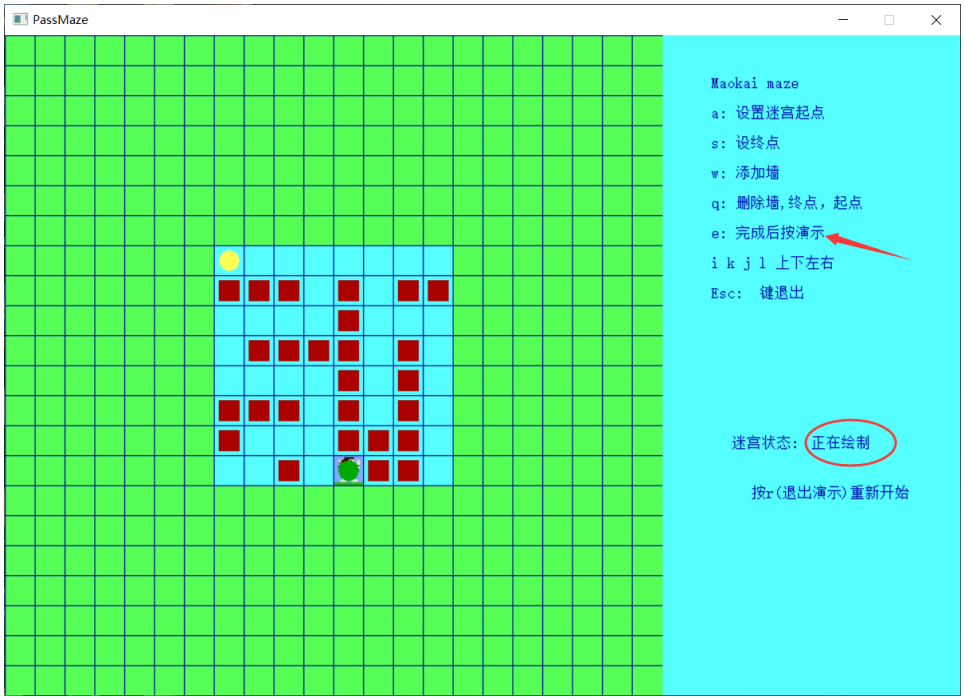
功能演示

- 输入一个 8*8 大小的迷宫
  - 

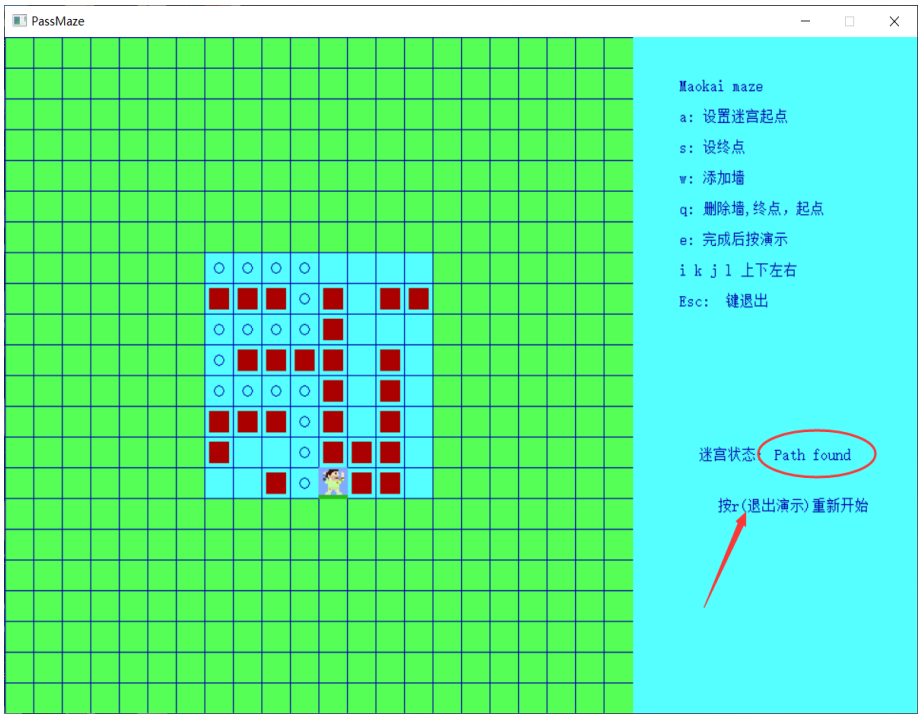
  -

- 

- 通过按 i k j l 控制小人设置迷宫,橙色的实心小圆圈表示起点，绿色实心圆圈表示终点，红色方块表示墙

- 

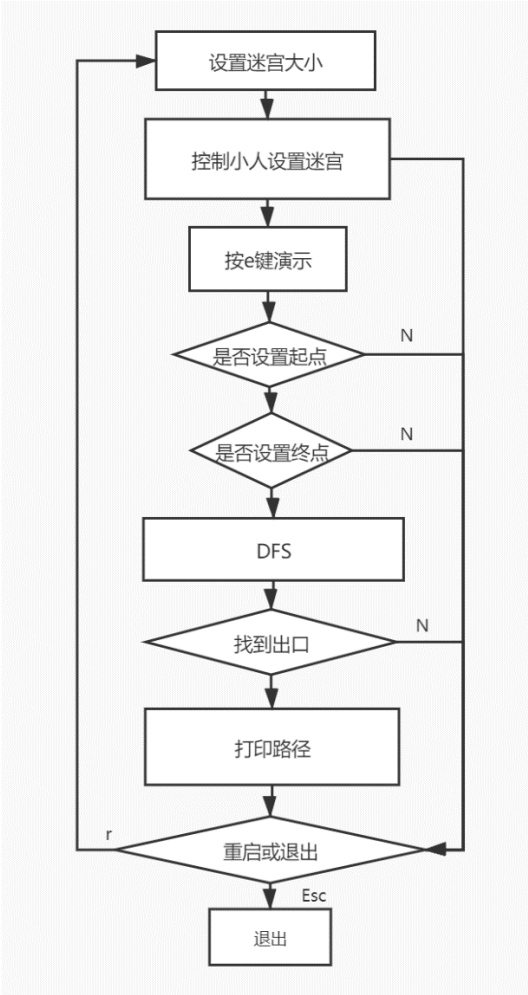
- 设置好迷宫后,按 e 键演示迷宫路径，空心圆圈表示足迹



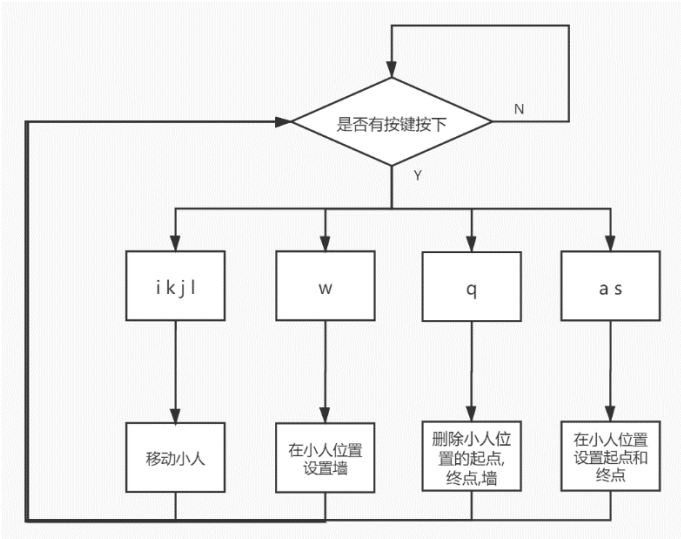
- 其余出错功能(未设起点或终点,无路径)等不一一演示(下载源码自行测试)

原理框图

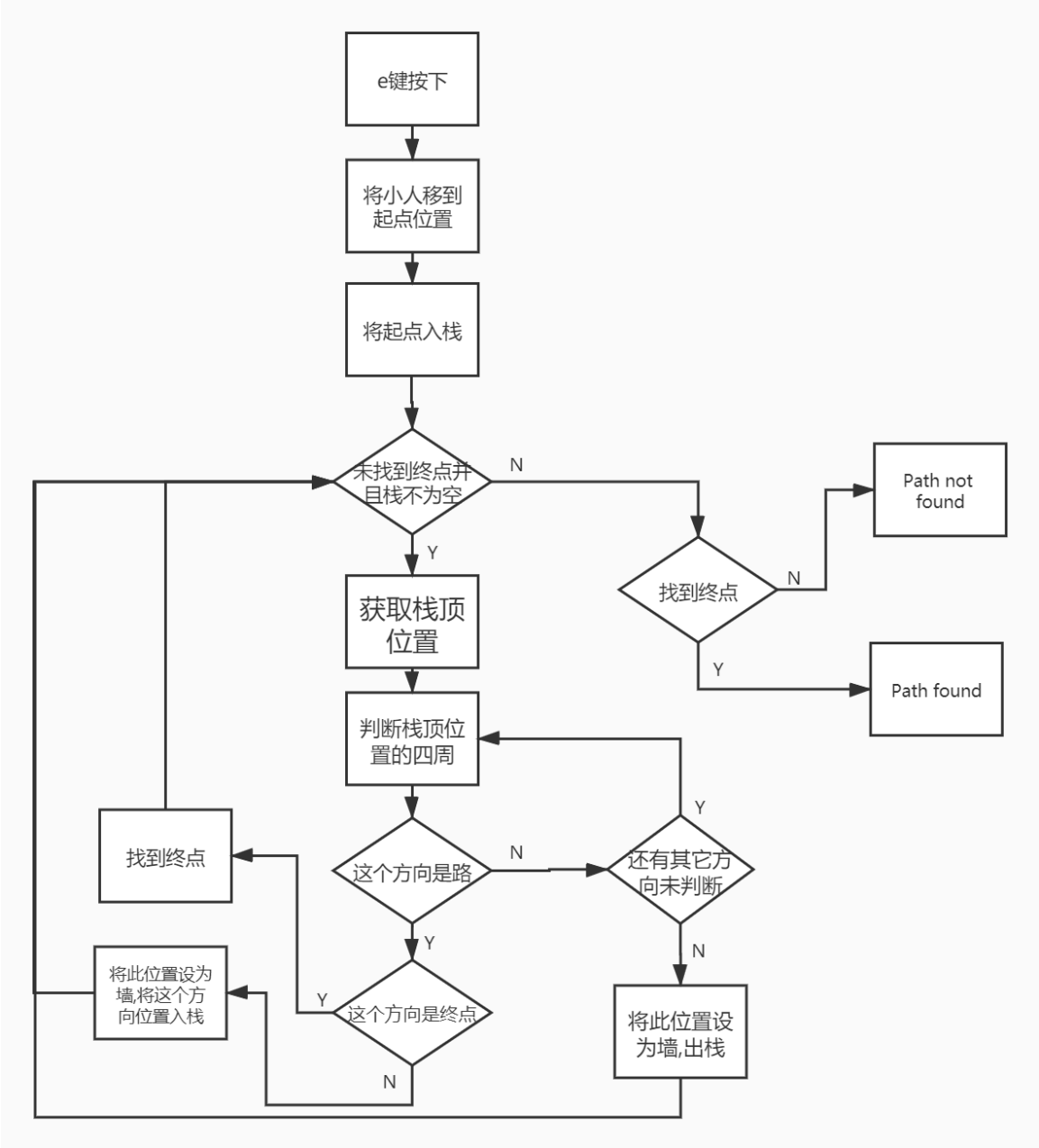
整体框图



设置迷宫框图



DFS 框图


e键按下
将小人移到起点位置
将起点入栈
未找到终点并且栈不为空
N
Y
Path not found
找到终点
N
Y
Path found
获取栈顶位置
判断栈顶位置的四周
找到终点
这个方向是路
N
还有其它方向未判断
Y
Y
N
将此位置设为墙,将这个方向位置入栈
Y
这个方向是终点
将此位置设为墙,出栈
N

代码模块

存储结构

```
#include<iostream>
#include<Windows.h>
#include<stack>
#include<queue>
#include<vector>
#include<conio.h>//键盘相关库
#include<graphics.h>//图形绘制相关库
using namespace std;
#pragma warning(disable : 4996) //关闭警告
struct node
{
    int x;
    int y;
    int flag;
};
```

```
class maze
{
public:
    maze();
    void run();
    void setmaze();//设置迷宫
    void initmaze();//初始化迷宫
    void movemessi();//移动小人
    void isrecover();
    void recover();
    void dfspass();//深度优先遍历
    void bfspass();//广度优先遍历
    void newsetout();
    void dfsprint();
    void bfsprint();//bfs打印
private:
    int key;//接受按键
    int mode;//运行模式
    IMAGE img;//图片对象
    int entryx;//入口x坐标
    int entryy;//入口y坐标
    int exitx;//出口x坐标
    int exity;//出口y坐标
    int entrycount;//入口个数
    int exitcount;//出口个数
    int wall[22][22]{};//墙数组 0表示墙,1表示通过
    int mazelong;//迷宫长
    int mazewide;//迷宫宽
    int messix;//小人坐标x
    int messiy;//小人坐标y
    node nodedirect[5]{};//0 center 1 2 3 4 东南西北
    stack<node> mystack;//深度优先遍历所用到的栈
    queue<node> myqueue;//广度优先遍历用到的队列
    vector<node> myvec;//广度优先遍历用到的数组
};
```

迷宫初始化

- 设置迷宫大小

```
cleardevice();

//设置迷宫的长和宽
wchar_t l[10];
wchar_t w[10];
InputBox(l, 10, L"请输入迷宫的长(1~20)");
InputBox(w, 10, L"请输入迷宫的宽(1~20)");
mazelong = _wtoi(l);
mazewide = _wtoi(w);
if (mazelong > 20 || mazelong < 1 || mazewide >20 || mazewide < 1)
{
    outtextxy(480, 330, L"long or wide error and 5s exit");
    Sleep(5000);
    exit(0);
}
```

- 绘制外墙与格子

```
//绘制外墙 左线x上线y右线x下线y
setfillcolor(LIGHTGREEN);//外围墙填充为绿色
solidrectangle(0, 0, (22 - mazelong) / 2 * 30, 660);//左墙
if (mazelong % 2 == 0)
    solidrectangle(660 - (22 - mazelong) / 2 * 30, 0, 660,660);//右墙
else
    solidrectangle(660 - (22 - mazelong) / 2 * 30 - 30, 0, 660, 660);//右墙 奇数右墙多刷一格
solidrectangle(0, 0, 660, (22 - mazewide) / 2 * 30);//上墙
if (mazewide % 2 == 0)
    solidrectangle(0, 660 - (22 - mazewide) / 2 * 30, 660, 660);//下墙
else
    solidrectangle(0, 660 - (22 - mazewide) / 2 * 30 - 30, 660, 660);//下墙 奇数下墙多刷一格
//绘制格子 (x1,y1)---(x2,y2)
setcolor(BLUE);
for (int i = 0; i < 22; ++i)
{
    line(i * 30, 0, i * 30, 660);
    line(0, i * 30, 660, i * 30);
}
```

- 绘制提示信息

```
//绘制提示信息
RECT r = { 710,40,900,300 };// 文字区域
setbkmode(TRANSPARENT);//文字背景为透明
//setcolor(BLUE);//文字为蓝色
setfont(15, 0, L"宋体");//文字格式
drawtext(L"Maokai maze \n\n\
a: 设置迷宫起点\n\n\
s: 设终点\n\n\
w: 添加墙\n\n\
q: 删除墙,终点，起点\n\n\
e: 完成后按演示\n\n\
i k j l 上下左右\n\n\
Esc:  键退出",
 &r,DT_WORDBREAK);//97 115 119 113 101
                  //105 107 106 108 27
setfillcolor(LIGHTGREEN);
newsetout();
outtextxy(730, 400, L"迷宫状态: ");
outtextxy(810, 400, L"等待绘制");
outtextxy(750, 450, L"按r(退出演示)重新开始");//114
```

- 初始化小人位置与墙数组

```
//初始化小人位置
messix = (22 - mazelong) / 2;
messiy = (22 - mazewide) / 2;
putimage(messix * 30, messiy * 30, &img);

//初始化墙数组
for (int i = 0; i < mazewide; ++i)
    for (int j = 0; j < mazelong; ++j)
        wall[i + (22 - mazewide) / 2 ][j + (22 - mazelong) / 2 ] = 1;
```

设置迷宫

- 小人上下左右移动

```
//messi移动但不能越界
if (key == 105)//i上
{
    isrecover();
    movemessi();
    messiy--;
    if (messiy <= (22 - mazewide) / 2) //防止越界
        messiy = (22 - mazewide) / 2;
    putimage(messix * 30, messiy * 30, &img);
    recover();
}
```

```
const COLORREF color[3] = { YELLOW,GREEN,RED };
void maze::isrecover()//判断移动messi是否要恢复格子状态
{
    nodedirect[0].x = messix * 30;
    nodedirect[0].y = messiy * 30;
    nodedirect[1].x = (messix + 1) * 30;
    nodedirect[1].y = messiy * 30;
    nodedirect[2].x = messix * 30;
    nodedirect[2].y = (messiy + 1) * 30;
    nodedirect[3].x = (messix - 1) * 30;
    nodedirect[3].y = messiy * 30;
    nodedirect[4].x = messix * 30;
    nodedirect[4].y = (messiy - 1) * 30;
    for (int i = 0; i < 5; ++i)
    {
        for(int j=0;j<3;++j)
            if (getpixel(nodedirect[i].x + 15, nodedirect[i].y + 15) == color[j])
            {
                nodedirect[i].flag = j+1;
            }
    }
}
```

- 此函数记录了人小移动前位置以及其四个方向的的颜色,黄色为起点,绿色为终点,红色为墙,小人利用图片覆盖的方式移动,若没有记录,会将其覆盖,所以得在移动前记录标记的,移动后进行恢复

```
void maze::recover()//根据标记恢复格子状态
{
    for (int i = 0; i < 5; ++i)
    {
        for(int j=0;j<3;++j)
        if (nodedirect[i].flag==j+1 )
        {
            setfillcolor(color[j]);
            if (j+1 == 3)
                solidrectangle(nodedirect[i].x+5, nodedirect[i].y+5, nodedirect[i].x + 25, nodedirect[i].y + 25);
            else
                solidcircle(nodedirect[i].x + 15, nodedirect[i].y + 15, 10);
        }
    }
    for (int i = 0; i < 22; ++i)
    {
        line(i * 30, 0, i * 30, 660);
        line(0, i * 30, 660, i * 30);
    }
    for (int i = 0; i < 5; ++i)
    {
        nodedirect[i].x = 0;
        nodedirect[i].y = 0;
        nodedirect[i].flag = 0;
    }
}
```

- 设置墙

```
//设置墙
//注意:绘图行x对应数组列y,y对应x
if (key == 119)
{
    setfillcolor(RED);
    solidrectangle(messix * 30+5, messiy * 30+5, messix * 30 + 25, messiy * 30 + 25);
    wall[messiy][messix] = 0;
    for (int i = 0; i < 22; ++i)
    {
        line(i * 30, 0, i * 30, 660);
        line(0, i * 30, 660, i * 30);
    }
}
```

- 删除墙或起点终点

```
//删除墙或起点终点
//注意:绘图行x对应数组列y,y对应x
if (key == 113)
{
    if (getpixel(messix * 30 + 15, messiy * 30 + 15) == RED)//删除墙
    {
        setfillcolor(LIGHTCYAN);
        solidrectangle(messix * 30, messiy * 30, (messix + 1) * 30, (messiy + 1) * 30);
        wall[messiy][messix] = 1;
    }
    else if (getpixel(messix * 30 + 15, messiy * 30 + 15) == GREEN)//删除终点
    {
        setfillcolor(LIGHTCYAN);
        solidrectangle(messix * 30, messiy * 30, (messix + 1) * 30, (messiy + 1) * 30);
        exitcount--;//终点数减一
        exitx = -1;
        exity = -1;
    }
    else if (getpixel(messix * 30 + 15, messiy * 30 + 15) == YELLOW)//删除起点
    {
        setfillcolor(LIGHTCYAN);
        solidrectangle(messix * 30, messiy * 30, (messix + 1) * 30, (messiy + 1) * 30);
        entrycount--;//起点数减一
        entryx = -1;
        entryy = -1;
    }
    for (int i = 0; i < 22; ++i)
    {
        line(i * 30, 0, i * 30, 660);
        line(0, i * 30, 660, i * 30);
    }
    putimage(messix * 30, messiy * 30, &img);
}
```

- 设置起点和终点

```
//设置入口和出口
if (key == 97 && entrycount==0)//入口数为0
{
    setfillcolor(YELLOW);
    solidcircle(messix * 30+15, messiy * 30+15, 10);//绘制黄色实心圆
    entrycount++;//入口个数加一
    entryx = messix;//入口x坐标为小人x坐标
    entryy = messiy;
}
if (key == 115 && exitcount==0)//出口数为0
{
    setfillcolor(GREEN);
    solidcircle(messix * 30+15, messiy * 30+15, 10);
    exitcount++;//出口个数加一
    exitx = messix;
    exity = messiy;
}
```

通行迷宫

- 移动小人到起点位置,将起点入栈

```
isrecover();//将小人移动到起点位置
movemessi();
recover();
putimage(entryx * 30, entryy * 30, &img);
bool success = false;//判断是否到达终点
int zx[4] = { 1,0,-1,0 };//4个方向 东南西北
int zy[4] = { 0,1,0,-1 };
node anode;//坐标节点
int x = 0, y = 0;//临时坐标
anode.x = entryx;
anode.y = entryy;
mystack.push(anode);//起点入栈
```

- 用栈进行深度优先遍历(核心代码)

```
while (!success && !mystack.empty())//没有找到终点并且栈不为空
{
    node head = mystack.top();      //得到栈顶坐标节点
    int i = 0;      //从第一个方向开始判断
    while (1)
    {
        x = head.x + zx[i]; //得到栈顶坐标的4个方向坐标
        y = head.y + zy[i];
        node newnode;       //定义临时节点记录坐标
        newnode.x = x;
        newnode.y = y;
        if (wall[y][x] == 1)//某个方向是路
        {
            if (x != exitx || y != exity)   //非终点
            {
                mystack.push(newnode);      //将此方向坐标节点坐标入栈
                wall[head.y][head.x] = 0;   //将栈顶节点设为墙
                break;                      //回到第一次while判断,取新的栈顶坐标
            }
            else if (x == exitx && y == exity)  //终点
            {
                mystack.push(newnode);          //将此方向坐标节点入栈
                success = true;                 //成功到达终点
                break;                          //回到第一次while判断会退出循环
            }
        }
        else if (wall[y][x] == 0 )  //方向是墙
        {
            if (i == 3)             //4个方向都判断过了且都是墙
            {
                wall[head.y][head.x] = 0;   //将栈顶节点设为墙
                mystack.pop();              //栈顶节点出栈,且节点每个方向都判断过且都走不通
                break;  //退出本次循环
            }
            ++i;        //继续判断其它方向
        }
    }
```

- 打印路径

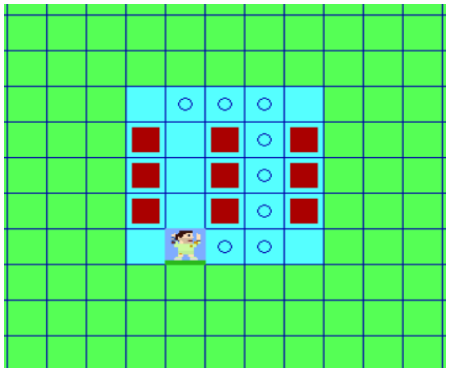
```
//打印路径
void maze::print()
{
    //用另一个栈保存DFS栈中的坐标节点(正确的从起点到终点的路径)
    stack<node> tempstack;
    while (!mystack.empty())
    {
        node tempnode = mystack.top();
        tempstack.push(tempnode);
        mystack.pop();
    }
    //打印
    while (!tempstack.empty())
    {
        node tempnode = tempstack.top();//得到栈顶节点坐标
        int x = tempnode.x;
        int y = tempnode.y;
        isrecover();//移动小人到栈顶节点
        movemessi();
        circle(messix * 30 + 15, messiy * 30 + 15, 5);//绘制空心圆圈
        recover();
        messix = x;
        messiy = y;

        putimage(x * 30, y * 30, &img);
        Sleep(500);//延时0.5秒
        tempstack.pop();//栈顶出栈
    }
    putimage(exitx * 30, exity * 30, &img);//最后将小人移动到出口位置
}
```

算法改进

DFS 转 BFS

DFS 的缺点:有多条路径时,无法得到最短路径



使用 BFS 可以得到最短路径,但 BFS 需要使用额外的一个数组记录路径节点

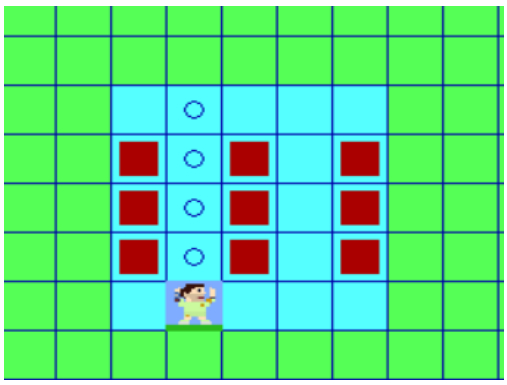
- vector 数组记录了每一个节点在 BFS 过程中的前驱节点

```
myvec.resize(22 * 22);//初始化bfs中使用的辅助数组
//某一点(x,y)对应数组中的 [ x * row + y ],myvec保存每个点的前驱点
```

- BFS 核心代码

```
bool success = false;//判断是否到达终点
int zx[4] = { 1,0,-1,0 };//4个方向 东南西北
int zy[4] = { 0,1,0,-1 };
node anode;//坐标节点
int x = 0, y = 0;//临时坐标
anode.x = entryx;
anode.y = entryy;
myqueue.push(anode);//起点入队
while (!success && !myqueue.empty())//没有找到终点并且队列不为空
{
    node tempnode = myqueue.front();//取队头节点
    wall[tempnode.y][tempnode.x] = 0;//将节点位置设为墙
    for (int i = 0; i < 4; ++i)
    {
        node tempnode1;//此节点用来保存队头节点的四个方向
        tempnode1.x = tempnode.x + zx[i];
        tempnode1.y = tempnode.y + zy[i];
        if (wall[tempnode1.y][tempnode1.x] == 1)//如果是路
        {
            //将某个方向节点的前驱节点设为队头节点
            myvec[tempnode1.y * mazewide + tempnode1.x] = tempnode;
            myqueue.push(tempnode1);//将这个方向节点入队
            if (tempnode1.x == exitx && tempnode1.y == exity)//如果是终点
            {
                success = true;//成功找到终点,退出while循环
                break;
            }
        }
    }
    myqueue.pop();//队头出队,判断下一个方向节点的四个方向
}
```

- 测试结果,得到最短路径



遇到的问题及解决方案

在最开始定义数组节点时,没有记录节点的 4 个方向,那么如何在深度优先遍历中每个节点都是从同一个方向开始判断?

- 第一两个方向数组,通过加这两个数组的方式得到其四个方向的坐标

```
int zx[4] = { 1,0,-1,0 };//4个方向 东南西北
int zy[4] = { 0,1,0,-1 };
node anode;//坐标节点
int x = 0, y = 0;//临时坐标
anode.x = entryx;
anode.y = entryy;
//cout << entryx << entryy << endl;
mystack.push(anode);//起点入栈

while (!success && !mystack.empty())//没有找到终点并且栈不为空
{
    node head = mystack.top();       //得到栈顶坐标节点
    int i = 0;       //重第一个方向开始判断
    while (1)
    {
        x = head.x + zx[i]; //得到栈顶坐标的4个方向坐标
        y = head.y + zy[i];
        node newnode;       //定义临时节点记录坐标
        newnode.x = x;
        newnode.y = y;
```

写完 DFS 后无法测试成功,通过调试 DFS 部分发现死循环,后发现错误原因是墙数组 `wall[x][y]`与绘图过程中小人坐标的`(x,y)`并不对应,墙数组 `wall[y][x]`才与小人坐标`(x,y)`对应

- 修改所有墙数组与小人坐标的对应方式

```
//设置墙
//注意:绘图行x对应数组列y,y对应x
if (key == 119)
{
    setfillcolor(RED);
    solidrectangle(messix * 30+5, messiy * 30+5, messix * 30 + 25, messiy * 30 + 25);
    wall[messiy][messix] = 0;
    for (int i = 0; i < 22; ++i)
    {
        line(i * 30, 0, i * 30, 660);
        line(0, i * 30, 660, i * 30);
    }
}
```