《实验十三——迷宫问题》

（一）问题描述

迷宫实验是取自心理学的一个古典实验。在该实验中，把一只老鼠从一个无顶大盒子的门放入，在盒中设置了许多墙，对行进方向形成了多处阻挡。盒子仅有一个出口，在出口处放置一块奶酪，吸引老鼠在迷宫中寻找道路以到达出口。对同一只老鼠重复进行上述实验，一直到老鼠从入口到出口，而不走错一步。老鼠经多次试验终于得到它学习走迷宫的路线。

模型建立：迷宫由 m 行 n 列的二维数组设置，0 表示无障碍，1 表示有障碍。设入口为（1，1），出口为（m，n），每次只能从一个无障碍单元移到周围四个方向上任一无障碍单元。编程实现对任意设定的迷宫，求出一条从入口到出口的通路，或得出没有通路的结论。

拓展：根据用户输入的迷宫大小，随机生成对应的迷宫，并实现交互式操纵人物走迷宫，及自动寻找迷宫的通路。

（二）解决思路

核心问题共分为三个部分：迷宫生成、人物移动及自动寻路。

迷宫生成时先将二维数组全部初始化为墙，然后选定一个初始位置，将墙打通，再随机选择一个方向，若没有生成回路，则将该方向上的墙打通，以此类推，通过递归生成迷宫。

人物移动可利用控制台光标位置的改变，将上一个位置刷新为空白，并在下一个可行位置打印人物，从而实现移动操作。

自动寻路可通过回溯算法实现。即在每个路口试探地选择一个方向向前搜索，若能够走通，则继续向前走，直到无路可走，这时，需要查看是否存在其它方向的通道，如果该位置没有任何可以通过的路径，需要沿原路返回，在历史分叉路口中选择另外一个方向继续试探，直到找到了出路，或者所有的通路都试探过了还无法走到终点，那就说明该迷宫不存在从起点到终点的通道。

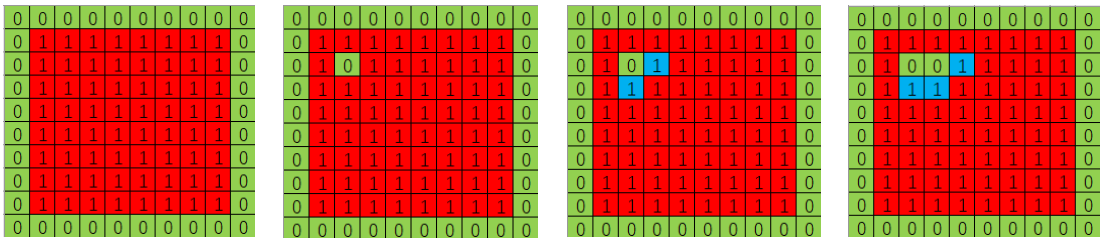
（三）数据结构

为了保证自动寻路时回溯的正常进行，需要用一个栈结构来保存从入口到当前位置的可行路径上所有位置的坐标和正在搜寻的方向。在需要回溯时，进行一次出栈操作，判断上一个位置是否有其它可通行的方向，如果所有方向都已遍历过，则继续出栈，直到找到某个有可通行方向的位置，或者栈为空（说明迷宫中不存在从起点到终点的路径）。

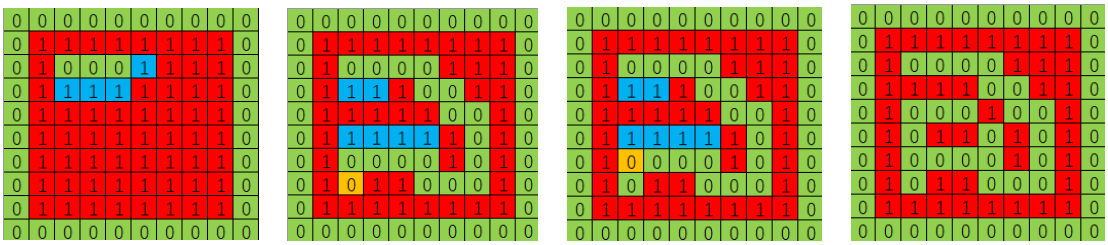
（四）算法分析

①**迷宫生成算法**：先用 malloc 函数分配内存，建立一个大小为(size+4)*(size+4)的二维数组，并将其全部初始化为墙，然后打通最外层形成一层保护，防止迷宫被挖穿。接着，选择一个初始位置（即入口），先判断该位置是否为墙，再判断其上下左右是否至少有 3 面墙（若没有，则打通墙后会生成回路），若条件均满足则打通墙。然后，随机选择一个方向，递归执行上述操作。注意递归退回上一层时，要在剩余的方向里再随机选择一个进行递归。

以 size = 6 的迷宫为例，具体生成的过程如下：



(a)　(b)　(c)　(d)



(e)　(f)　(g)　(h)

如图(a)所示为迷宫的初始状态，绿色表示路，红色表示墙，最外层为保护层，防止迷宫被挖穿。先将入口处的墙打通，如图(b)。此时，入口处的绿色方块有两个可行方向（蓝色标记），随机选择一个，如图(c)选择将右边的墙打通，此时又产生两个可行方向，如图(d)。依次类推，不断递归直至图(f)，此时递归到了黄色方块，由于该位置无可行方向，故退回到上一层递归，如图(g)。在该位置的可行方向里随机选择一个，继续重复上述操作，直至最终不存在蓝色的标记，即生成了一个迷宫，如图(h)所示。

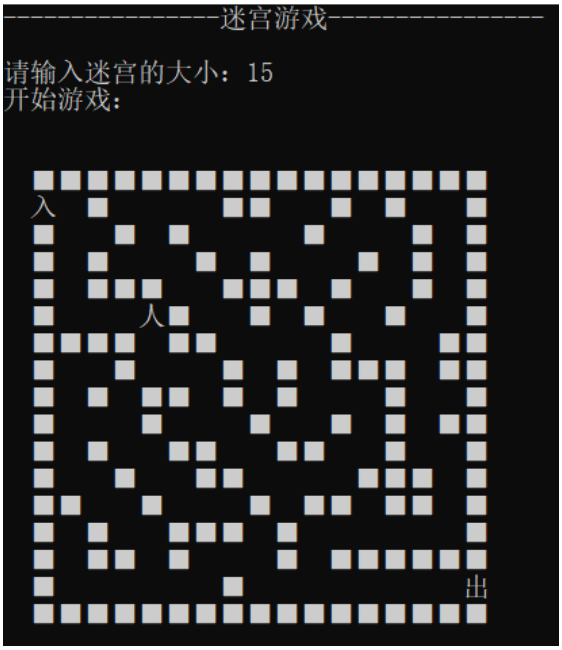
②**人物移动**：利用不带回显的 getch 函数（包含在 conio.h 头文件中）重复读取用户输入，若用户选择的方向上没有墙，则将控制台光标置于当前位置并输出空白（即空格），然后更新人物位置，将光标置于新的位置并打印人物，接着进入新一轮循环，直至人物到达迷宫出口或用户输入 Q 退出；若用户选择的方向上有墙，则直接进入下一轮循环，重复上述操作。

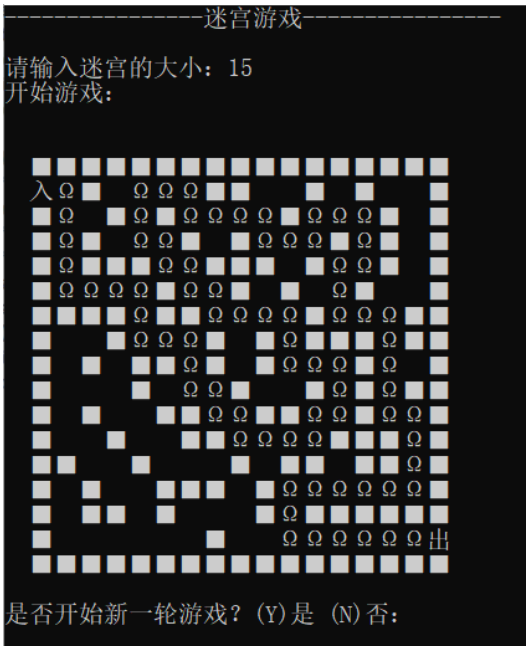
③**自动寻路算法**：首先构建一个栈，用于存储搜索过程中的坐标位置及方向。从迷宫入口开始，重复执行入栈及向下搜索操作，直至没有可行路径（即遇到墙），此时查看栈是否为空，若不为空，则从栈顶取出搜索过程中的结点进行回溯，查看其搜索方向，若已经搜索到最后一个方向（表明该位置没有可行路径），则继续出栈，否则就换个方向继续搜索。直到发现栈为空，没有找到可行路径，程序返回 false。若成功找到迷宫出口，程序返回 true。注意入栈操作时，需把二维数组中的相应位置元素由“ROUTE”改为“PATH”，出栈操作相反，以方便后续迷宫及其通路的打印输出。

（五）运行结果

测试数据：迷宫大小 size = 15

结果截图：





说明：输入迷宫大小后可自动生成一个迷宫，按键 W、S、A、D 分别操纵人物的上下左右移动，使人物从入口移动到出口，或者按下 Q 后可自动输出迷宫从入口到出口的路径，并选择是否开始新一轮游戏。

（六）反思总结

程序中存在的缺陷：

① 使用的迷宫生成算法无法保证存在一条从入口到出口的通路，但在实际运行中，生成的迷宫绝大多数都是存在通路的；

② 由于程序中使用了 Windows 控制台的系统库函数，因此程序不具有可移植性。

收获与感想：

通过本次实验，我进一步掌握了栈这种数据结构以及回溯算法的思想，学习了随机数函数的使用及随机数种子的设置，并了解了一部分 Windows 控制台的系统库函数，如清屏、程序暂停运行、光标位置设置与隐藏、改变代码页以输出不同的字符编码集等。更重要的是，在实验过程中，我学会了如何去调试程序以及遇见问题如何去查找资料自主学习。总之，通过本次实验，我的编程能力与解决问题的能力都得到了提升。

（七）源程序

```
//迷宫问题2.0
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <windows.h>
#include <conio.h>
#include <time.h>
#define WALL 1                  //墙
#define ROUTE 0                 //路
#define PATH 2                  //迷宫通道
#define OFFSET 5                //光标偏移量

//以左上角为原点，竖直向下为x轴正方向，水平向右为y轴正方向
typedef struct {
    int x;                  //位置坐标
    int y;
} position;

typedef struct {
    position pos;           //位置及搜索方向
    int direction;
} possibleWay;

typedef struct Node {
    possibleWay data;       //栈中元素
    struct Node* next;
} node;

node* top = NULL;           //栈顶指针

//函数原型
void ClearStack(void);
bool StackIsEmpty(void);
bool StackIsFull(void);
bool push(possibleWay data);
possibleWay pop(void);

void SetCCPos(int x, int y);
void HideCursor(void);

void dig(int** maze, int x, int y);
void createMaze(int** maze, int size);
```

```
position move(position player, int direction);
char playGame(int** maze, int size);
position nextPos(position cur, int direction);
bool mazePath(int** maze, int size);
void printMaze(int** maze, int size);

int main(void) {
    char choice;
    do {int size;              //迷宫大小
        do {system("cls");
            printf("----------------迷宫游戏----------------\n\n");
            printf("请输入迷宫的大小：");
            scanf("%d", &size);
            if (size < 3) { printf("迷宫大小必须大于等于3！"); Sleep(800);}
        } while (size < 3);

        int** maze = (int**)malloc((size + 4) * sizeof(int*));
        for (int i = 0; i < size + 4; i++)
            maze[i] = (int*)malloc((size + 4) * sizeof(int));
        createMaze(maze, size);

        printf("开始游戏：\n\n");
        printMaze(maze, size);
        char ch = playGame(maze, size);
        if (ch == 'q') {
            if (mazePath(maze, size)) {
                SetCCPos(OFFSET, 0);
                printMaze(maze, size);
            } else {
                SetCCPos(size + 4 + OFFSET, 0);
                puts("迷宫不存在通路！"); }
        }

        for (int i = 0; i < size + 4; i++) free(maze[i]);        //释放内存
        free(maze);

        printf("是否开始新一轮游戏？(Y)是 (N)否：");
        fflush(stdin);             //清空输入缓冲区
        choice = getchar();
        if (choice == 'y') choice += 'A' - 'a';
    } while (choice == 'Y');

    return 0;
}
```

```
//创建迷宫
void createMaze(int** maze, int size) {
    srand((unsigned)time(NULL));   //设置随机数种子
    //开始时，迷宫全为墙
    for (int i = 0; i < size + 4; i++)
        for (int j = 0; j < size + 4; j++)
            maze[i][j] = WALL;
    //边界保护，防止迷宫被挖穿
    for (int i = 0; i < size + 4; i++) {
        maze[0][i] = ROUTE;
        maze[i][0] = ROUTE;
        maze[size + 3][i] = ROUTE;
        maze[i][size + 3] = ROUTE;
    }
    dig(maze, 2, 2);
    maze[size + 1][size + 1] = ROUTE;
}

void dig(int** maze, int x, int y) {
    if (maze[x][y] == WALL) {
        //防止挖穿形成环
        if (maze[x + 1][y] + maze[x - 1][y] + maze[x][y + 1] + maze[x][y - 1] >= 3) {
            maze[x][y] = ROUTE;
            int directions[4] = { 0, 1, 2, 3 };
            for (int i = 4; i > 0; i--) {
                int r = rand() % i;
                int temp = directions[r];
                directions[r] = directions[i - 1];
                directions[i - 1] = temp;
                switch (directions[i - 1]) {
                case 0:
                    dig(maze, x - 1, y); break;
                case 1:
                    dig(maze, x, y + 1); break;
                case 2:
                    dig(maze, x + 1, y); break;
                case 3:
                    dig(maze, x, y - 1); break;
                default: break;
                }
            }
        }
    }
}
```

```
//搜索迷宫
position nextPos(position cur, int direction) {
    position next;
    switch (direction) {
    case 0:
        next.x = cur.x + 1;                //向下搜索
        next.y = cur.y;
        break;
    case 1:
        next.x = cur.x;                    //向右搜索
        next.y = cur.y + 1;
        break;
    case 2:
        next.x = cur.x - 1;                //向上搜索
        next.y = cur.y;
        break;
    case 3:
        next.x = cur.x;                    //向左搜索
        next.y = cur.y - 1;
        break;
    }
    return next;
}


//人物移动
position move(position player, int direction) {
    SetCCPos(player.x + OFFSET, player.y * 2);
    printf("  ");
    player = nextPos(player, direction);
    SetCCPos(player.x + OFFSET, player.y * 2);
    printf("人");
    return player;
}


//开始游戏
char playGame(int** maze, int size) {
    HideCursor();              //隐藏控制台光标
    position player = { 2, 2 };
    SetCCPos(player.x + OFFSET, player.y * 2);
    printf("人");
    char ch;
```

```
    while ((ch = getch()) != 'q') {
        switch (ch) {
        case 'a':               //左移
            if (maze[player.x][player.y - 1] != WALL)
                player = move(player, 3);
            break;
        case 's':               //下移
            if (maze[player.x + 1][player.y] != WALL)
                player = move(player, 0);
            break;
        case 'd':               //右移
            if (maze[player.x][player.y + 1] != WALL)
                player = move(player, 1);
            break;
        case 'w':               //上移
            if (maze[player.x - 1][player.y] != WALL)
                player = move(player, 2);
            break;
        default: continue;
        }
        if (player.x == size + 1 && player.y == size + 1) {
            SetCCPos(size + 4 + OFFSET, 0);
            puts("成功！");
            break;
        }
    }
    return ch;
}

//求解迷宫问题
bool mazePath(int** maze, int size) {
    position cur = { 2, 2 };
    possibleWay way;

    do { if (maze[cur.x][cur.y] == ROUTE) {
            maze[cur.x][cur.y] = PATH;
            way.pos = cur; way.direction = 0;
            push(way);
            if (cur.x == size + 1 && cur.y == size + 1) {
                ClearStack();
                return true;
            }
            cur = nextPos(cur, 0);
        }
```

```
        else {
            way = pop();
            while (way.direction == 3 && !StackIsEmpty()) {
                maze[way.pos.x][way.pos.y] = 0;
                way = pop();
            }
            if (way.direction < 3) {
                way.direction++;
                push(way);
                cur = nextPos(way.pos, way.direction);
            }
        }
    } while (!StackIsEmpty());
    return false;
}

//输出迷宫路径
void printMaze(int** maze, int size) {
    for (int i = 0; i < size + 4; i++) {
        for (int j = 0; j < size + 4; j++) {
            SetConsoleOutputCP(437);       //设置代码页，输出扩展ASCII码字符集
            if (i == 2 && j == 1) {
                SetConsoleOutputCP(936);   //设置代码页，输出简体中文字符集
                printf("入");
                continue;
            }
            if (i == size + 1 && j == size + 2) {
                SetConsoleOutputCP(936);
                printf("出");
                continue;
            }
            if (maze[i][j] == WALL)
                putchar(254);     //白色方块字符，2个字节
            else if (maze[i][j] == ROUTE)
                printf("  ");
            else
                putchar(234);     //Ω字符，2个字节
        }
        putchar('\n');
    }
    SetConsoleOutputCP(936);
}
```

```
//销毁栈
void ClearStack(void) {
    while (top) {
        node* temp = top;
        top = top->next;
        free(temp);
    }
}

//判断栈是否为空
bool StackIsEmpty(void) { return (top == NULL) ? true : false; }

//判断栈是否已满
bool StackIsFull(void) {
    node* temp = (node*)malloc(sizeof(node));
    if (temp) {
        free(temp);
        return false;
    }
    else
        return true;
}

//入栈
bool push(possibleWay data) {
    if (StackIsFull()) return false;
    node* temp = (node*)malloc(sizeof(node));
    temp->data = data;
    temp->next = top;
    top = temp;
    return true;
}

//出栈
possibleWay pop(void) {
    if (StackIsEmpty()) return;
    else {
        possibleWay data = top->data;
        node* temp = top;
        top = top->next;
        free(temp);
        return data;
    }
}
```

```
//设置光标位置
void SetCCPos(int x, int y) {
    HANDLE hOut;
    hOut = GetStdHandle(STD_OUTPUT_HANDLE);      //获取标注输出句柄
    COORD pos;
    pos.X = y;
    pos.Y = x;
    SetConsoleCursorPosition(hOut, pos);         //偏移光标位置
}

//隐藏光标
void HideCursor(void) {
    HANDLE handle = GetStdHandle(STD_OUTPUT_HANDLE);
    CONSOLE_CURSOR_INFO CursorInfo;
    GetConsoleCursorInfo(handle, &CursorInfo);  //获取控制台光标信息
    CursorInfo.bVisible = false;                //隐藏控制台光标
    SetConsoleCursorInfo(handle, &CursorInfo);  //设置控制台光标状态
}
```