1. 选题背景与应用意义

（描述选题的背景、所针对的具体实际问题及任务所体现的实用性价值和意义所在等）

作为益智游戏的经典，迷宫游戏曾风靡全球。玩家通过迷宫游戏可以锻炼判断力和记忆力，培养耐心，同时也要求玩家具备一定综合分析能力和对时空的感知能力。

在应用本身意义方面，本任务就是开发一款易玩有趣的迷宫游戏, 游戏中一般含有“墙”（障碍物）和“路”（通路）两种类型的块，可通过视野限制、时间限制、定义迷宫规模等机制加大难度。玩家通过键盘操纵角色在随机生成的地图中移动，通过探索寻找到出口。迷宫游戏主要考验玩家对全局地图（或视野内地图）的把握和试错的耐心，地图不规则程度更高、迷宫不可以固定方式解出、干扰路线的强干扰性等要素会提高玩家的试错成本，从而增加游戏的难度。通过本游戏可以最大限度地挑战与磨练人的好奇心和耐力, 致使游戏过程乐趣无穷。

在设计者能力提升方面，我们在了解并掌握《集合论与图论》、《数据结构与算法》、《高级语言程序设计》等基本原理和方法的基础上，已具备初步的独立分析和设计能力。通过学习软件开发的相关技术和方法，选择适当的数据结构、设计有效算法，开展问题求解和软件开发实践，能够很好锻炼我们分析问题和软件开发的能力。

2. 需求分析

（根据任务选题的要求，充分地分析和理解问题，明确用户要求做什么？完整性约束条件是什么？运行环境要求、图形操作界面要求等）

(1) 游戏功能需求：玩家操控角色在全局视角受限的地图中寻找迷宫出口，通过改变迷宫大小和算法实现，使难度随关卡推进而增加，玩家可在次数限制内查看路径提示，但过多可能导致游戏结束。游戏设置不同场景主题，游戏内部提供计时功能。

(2) 用户界面设计需求：在整个游戏的开发中以操作简便、界面美观、灵活使用为出发点，界面设计要求充分考虑玩家的感受，有丰富多彩的图片与场景，界面直观，互动性好，操作简单使得玩家容易上手。

(3) 可靠性需求：让玩家进行操作的过程中，要求游戏不能频繁发生错误导致不得不中止游戏的现象，对游戏的可靠性提出较高要求。

(4) 运行环境要求：Windows 系统

3. 系统主要功能设计

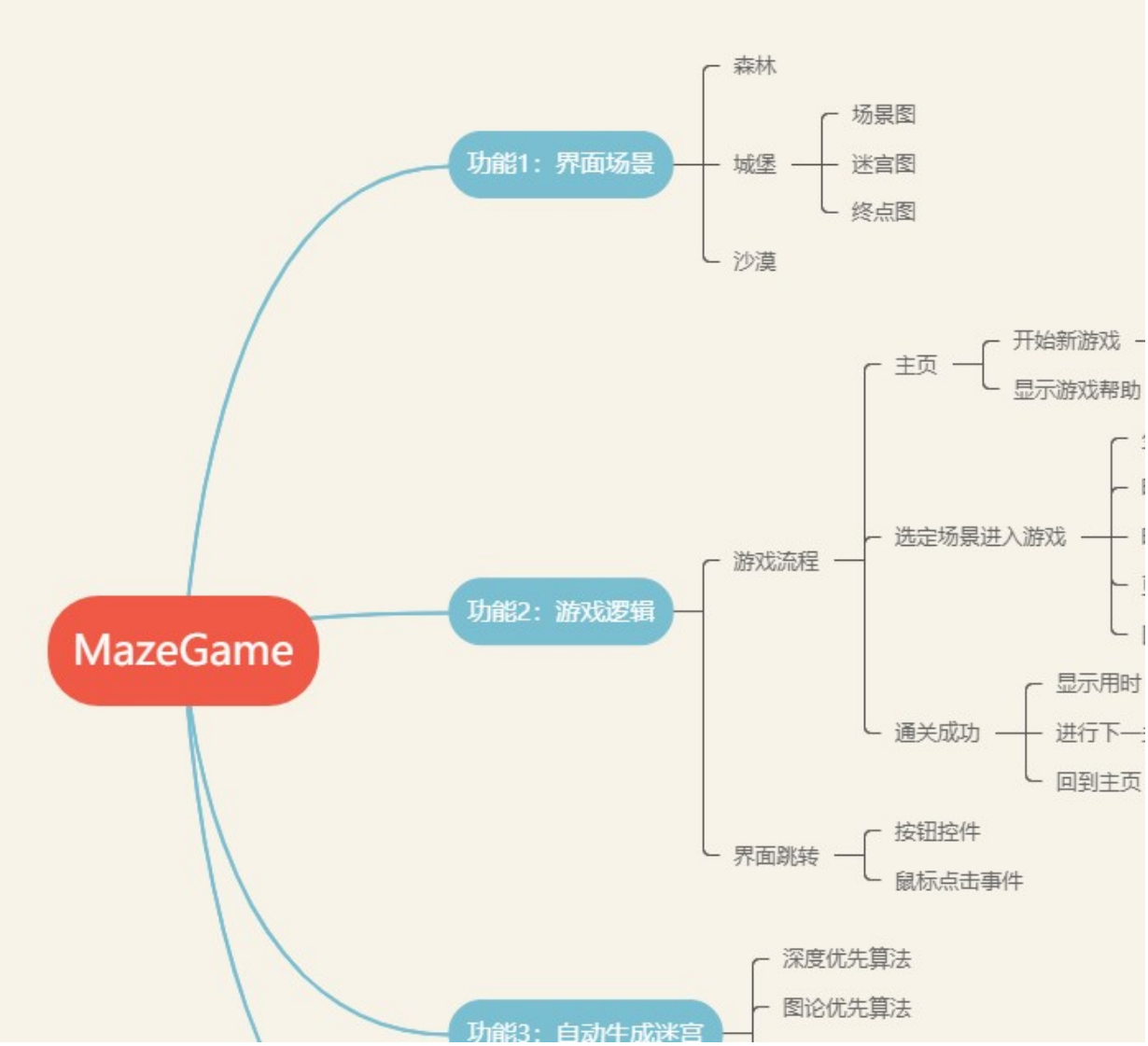
（根据需求分析来详细设计软件系统的主要功能模块，要求画出功能模块图，含子功能模块图，并给出详细文字描述）

(1) 游戏逻辑:利用图形库制作游戏主界面及各场景界面，实现界面内的按钮控件功能和界面间的逻辑跳转，控制游戏流程。

(2) 迷宫场景:迷宫游戏设计了局部的视野窗口，设定了森林、城堡、沙漠、星空四个主题，不同主题对应不同迷宫生成算法。

(3) 自动随机生成迷宫：利用随机 Prim 算法、深度优先搜索算法、递归分割算法等生成迷宫。

(4) 求解迷宫出路：当玩家感觉走到“死胡同”时可以选择路径提示功能。为增大难度只有在玩家停止时才会显示提示，且仅提示一次。



4. 核心算法设计与分析

（根据软件功能设计，概述所用到的主要数据结构；用伪代码或程序流程图的形式来详细描述核心算法的功能及过程，并定性分析其时间、空间复杂度）

(1) 数据结构：

a) 图：利用邻接矩阵的方式存储图的信息。定义含有规模大小、顶点数、边数、迷宫二维数组的结构体。

b) 栈：探索路径是在不断前进与后退的过程中完成，以栈 S 记录“当前路径”，则栈顶中存放的是“当前路径上最后一个通道块”。由此，“纳入路径”的操作即为“当前位置入栈”；“从当前路径上删除前一通道块”的操作即为“出栈”。

c) 队列：一层一层的往外拓展可走的点，有序地保存了所有可通的点。取出队头即可探索与其相通的位置。

(2) 核心算法：

a) 用栈的方法找路径：

从入口出发，顺某一方向向前探索，若能走通，则继续往前走；否则沿原路反回，换一个方向继续探索，直至到达出口或无路可退（即栈为空）。为了保证在任何位置上都能沿原路返回，显然需要用一个后进先出的栈的结构来保存从入口到当前位置的路径。

找到起点，将当前位置的初值设为入口位置，入栈标记为已访问；

```
While(栈不空)
{
    当前位置改为下一探索方向；
    If(当前位置可通)
    {   入栈并标志为已访问；
        If(该位置为出口)结束；
    }
    else
    {       if(栈不为空且栈顶位置尚有其他方向未经探索)
             则探索方向设置为下一个；
            else{
                    While（栈不为空但栈顶位置的四周均不可通）
                    { 删去栈顶位置；重新标志为未访问；}
                    if（栈不空）
                    { 得到新的栈顶元素；方向设置为下一个；}
                }
    }
}
```

b) 用队列的方法找路径：

从入口出发，利用队列的特点，一层一层的往外拓展可走的点，直到找到出口为止，后进行路径回溯输出。若队列为空且未达终点，则没有路径。

找到起点将其入队，将当前位置的初值设为入口位置；

While(队列非空且未找到终点）

{ 出队列取队头位置，在矩阵 mark[][]中标记访问过该位置；

For(遍历该位置周围四个方向)

{

If(位置可通行且未访问）

{入队列；

在 path[][]中记录此时的方向;

}

If(若已找到终点)

{则开始回溯路径退出;}

}

}

c) 深度优先搜索算法

1. 将起点作为当前迷宫单元并标记为已访问
2. 当还存在未标记的迷宫单元，进行循环
    1. 如果当前迷宫单元有未被访问过的的相邻的迷宫单元
        1. 随机选择一个未访问的相邻迷宫单元
        2. 将当前迷宫单元入栈
        3. 移除当前迷宫单元与相邻迷宫单元的墙
        4. 标记相邻迷宫单元并用它作为当前迷宫单元
    2. 如果当前迷宫单元不存在未访问的相邻迷宫单元，并且栈不空
        1. 栈顶的迷宫单元出栈
        2. 令其成为当前迷宫单元

d) 随机 Prim 算法

1. 让迷宫全是墙.
2. 随机选一个单元格作为迷宫的通路，然后把它的邻墙放入列表
3. 当列表里还有墙时
    1. 从列表里随机选一个墙，如果这面墙分隔的两个单元格只有一个单元格被访问过
        1. 那就从列表里移除这面墙，即把墙打通，让未访问的单元格成为迷宫的通路
        2. 把这个格子的墙加入列表
    2. 如果墙两面的单元格都被访问过，那就从列表里移除这面墙

e) 递归分割算法

1. 让迷宫全是迷宫单元
2. 随机选择一偶数行和一偶数列让其全部变为墙，通过这两堵墙将整个迷宫分为四个子迷宫

3. 在 3 面墙上各挖一个洞（为了确保连通）

4. 如果子迷宫仍可分割成四个子迷宫，返回 1. 继续分割子迷宫

5. 系统核心模块实现

（给出编程语言、开发环境及支撑软件、软件系统架构；给出主要数据结构的定义代码以及核心算法对应的关键函数定义代码，包括输入参数的含义、函数输出结果等；核心函数的实现代码段及注解等；主要功能界面截图及对应文字概述等）

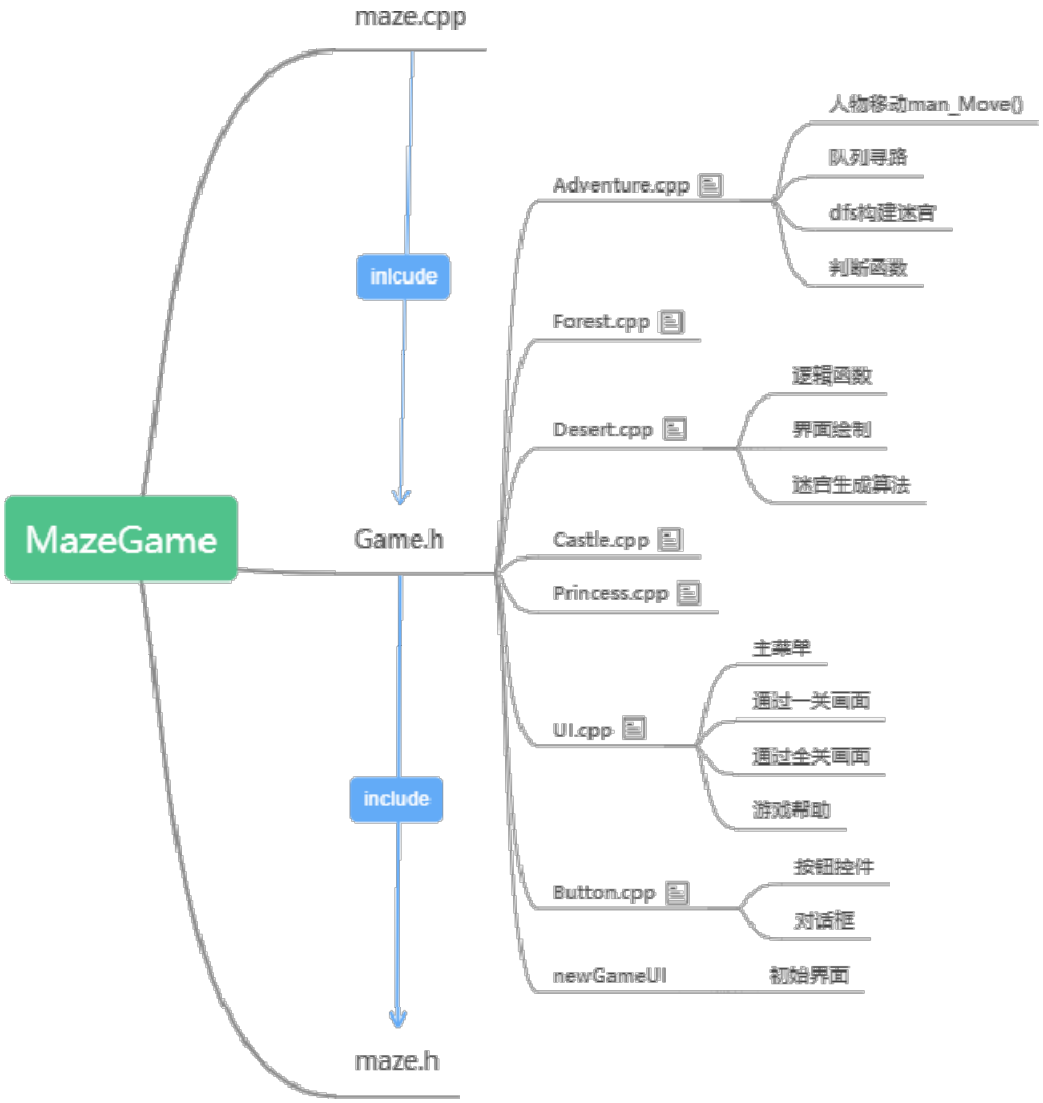
(3) 编程语言：C++/C

(4) 开发环境：Windows

(5) 支撑软件：Visual Studio 2019

(6) 简易图形库： Easy Graphics Engine (EGE)

(7) 开发文件系统架构：



(8) 数据结构：

a) 边：有向边起点与终点

```
/* 边的结构体定义 */
struct Edge
```

```
    {
        int head, tail;      /* 有向边<head,tail> */
    };
```

b) 图：利用邻接矩阵的方式存储图的信息。定义含有规模大小、顶点数、边数、迷宫二维数组的结构体。

```
typedef struct
    {
        int Nv;  /* 顶点数 */
        int Ne;  /* 边数   */
        int des_x, des_y;       //终点坐标
        int size_n, size_m;     //迷宫大小

        int map[MaxN][MaxN];    //迷宫地图存储
        int fmap[MaxN][MaxN];   /*辅助迷宫地图*/

        int reg[2600][2600];    /* 邻接矩阵 */
        int feg[2600][2600];    /*辅助邻接矩阵*/
    }Map;
```

c) 栈：

```
typedef struct StackNode {
        /* 栈中存储的节点 **/
        int x;
        int y;
        int dirCount;   /*表示上一步走到该步的方向*/
        }StackNode;
```

d) 队列：

```
struct QueueNode {
        /** 队列中存储的节点 */
        int x;
        int y;
    };
```

(9) 核心算法

a) 用栈的方法找路径关键代码：

```
void CAdventure::SolveByStack(int rows, int cols, int num) {
    /** 运用栈求解迷宫路径 */
    while (!s.empty()) /*栈不空*/
    {
        g = g + dir[dirCount][0];   /*下一步试探周围位置*/
        h = h + dir[dirCount][1];
        if (maze.map[g][h] == END && mark[g][h] == 0)
        {
            mark[g][h] = 1;
            temp.x = g;
            temp.y = h;
```

```
            temp.dirCount = dirCount;
            s.push(temp);
            find = 1;       /*到达终点，入栈并find=1,退出循环*/
            break;
        }
        if (maze.map[g][h] == ROAD && mark[g][h] == 0)/*当前位置可通且并未访问过*/
        {
            mark[g][h] = 1;
            temp.x = g;
            temp.y = h;
            temp.dirCount = dirCount;
            s.push(temp);
            dirCount = 0;       /*进行下一步之前改变dirCount = 0为初始方向*/
        }
        else if (dirCount < 3 && (!s.empty()))     /*还有其他方向尚未探索*/
        {
            temp = s.top();
            g = temp.x;
            h = temp.y;
            dirTemp = temp.dirCount; /*该步不走，g,h返回为上一步的值*/
            dirCount += 1;        /*更改为下一个方向再试探*/
        }
        else
        {
            /*若栈不空但栈顶位置四周均不可通*/
            while ((dirCount == 3) && (!s.empty()))
            {
                temp = s.top();
                g = temp.x;
                h = temp.y;
                dirCount = temp.dirCount; /*该步不走，g,h返回为上一步的值*/
                s.pop();  /*g,h传地址，弹出该步的位置和相对上一步方向*/
                mark[g][h] = 0;    /*弹出栈顶且将mark[][]重新标记为未访问*/
            }
            if (!s.empty())/*弹出几个不可通过的位置，栈不空*/
            {
                temp = s.top();
                g = temp.x;
                h = temp.y;
                dirTemp = temp.dirCount;
                dirCount += 1;       /*下一个方向重新试探*/
            }
        }
    }
    /** 输出栈求解迷宫经过的路径，略*/
}
```

b) 用队列的方法找路径关键代码：

```
void CAdventure::solveByQueue(int rows, int cols,int num) {
    while (!q.empty()&&!find)   //当队列非空时且未找到时继续执行，否则算法结束。
    {
        //出队列取队头位置，在矩阵mark[][]中标记访问过该位置。
        temp = q.front();
        q.pop();
        mark[temp.x][temp.y] = 1;
        //遍历该位置周围四个方向
        for (i = 0; i <= 3; i++)
        {
            tempG = temp.x + dir[i][0];
            tempH = temp.y + dir[i][1];
            //将可通行且未访问的位置入队列，在path[][]中记录此时的方向
            if (maze.map[tempG][tempH] == ROAD && mark[tempG][tempH] == 0)
            {
                temp2.x = tempG;
                temp2.y = tempH;
                q.push(temp2);
                path[temp2.x][temp2.y] = i;
                mark[temp2.x][temp2.y] = 1;
            }
            //若已找到终点，则开始回溯路径退出
            if (maze.map[tempG][tempH] == END)
            {
                //setbkcolor(RED);
                temp2.x = tempG;
                temp2.y = tempH;
                q.push(temp2);
                path[tempG][tempH] = i;
                find = 1;   /*到达出口*/
            }
        }
    }
    /**根据是否找到路径输出对应的内容，略*/
    }
```

c) 深度优先搜索算法

d) 随机 Prim 算法

```
//Prim 随机生成迷宫
void CForest::prim()
{
    maze.Nv = 0;            //节点数清空
    maze.Ne = 0;
    std::vector <Edge> e;       //存边的数组序列
```

```
    for (int i = 1; i <= maze.size_n; i++)      //初始化
    {
        for (int j = 1; j <= maze.size_m; j++)
        {
            if (i % 2 == 0 && j % 2 == 0 && i != maze.size_n && j != maze.size_m)
            {
                maze.map[i][j] = ROAD;
                flag[0][++maze.Nv] = maze.Nv;
                flag[1][maze.Nv] = i;
                flag[2][maze.Nv] = j;
            }
            else
            {
                maze.map[i][j] = WALL;
            }
        }
    }
    //初始化邻接矩阵均不可达
    for (int i = 1; i <= maze.Nv; i++)
    {
        for (int j = 1; j <= maze.Nv; j++)
        {
            maze.reg[i][j] = INF;
            maze.feg[i][j] = INF;
        }
    }

    for (int i = 1; i <= maze.Nv; i++)
    {
        visit[i] = 0;
        if (i % ((maze.size_n - 1) / 2) != 0)
        {
            maze.feg[i][i + 1] = 1;
            maze.feg[i + 1][i] = 1;
        }
        if (i <= maze.Nv - (maze.size_n - 1) / 2)
        {
            maze.feg[i][i + (maze.size_n - 1) / 2] = 1;
            maze.feg[i + (maze.size_n - 1) / 2][i] = 1;
        }
    }
    for (int i = 1; i <= maze.Nv; i++)
    {
        if (maze.feg[1][i] == 1)
        {
            Edge efo;
```

```
            efo.head = i;
            efo.tail = 1;
            e.push_back(efo);
        }
    }

    visit[1] = 1;
    //Prim 算法核心
    for (int i = 1; i <= maze.Nv - 1; i++)
    {
        std::random_shuffle(e.begin(), e.end());     //将所有元素随机打乱
        Edge arr;
        while (1) {
            arr = e.back();
            if (visit[arr.head] && visit[arr.tail])
            {
                e.pop_back();
            }
            else
            {
                break;
            }
        }
        e.pop_back();
        visit[arr.head] = 1;
        visit[arr.tail] = 1;
        maze.reg[arr.tail][arr.head] = 1;
        maze.reg[arr.head][arr.tail] = 1;    // 随机选边并标记

        for (int j = 1; j <= maze.Nv; j++)  // 加入侯选边
        {
            if (maze.feg[arr.head][j] == 1 && !visit[j])
            {
                Edge afo;
                afo.head = j;
                afo.tail = arr.head;
                e.push_back(afo);
            }
        }
    }

    for (int i = 1; i <= maze.Nv; i++)
    {
        for (int j = 1; j <= maze.Nv; j++)
        {
            if (maze.reg[i][j] == 1)
```

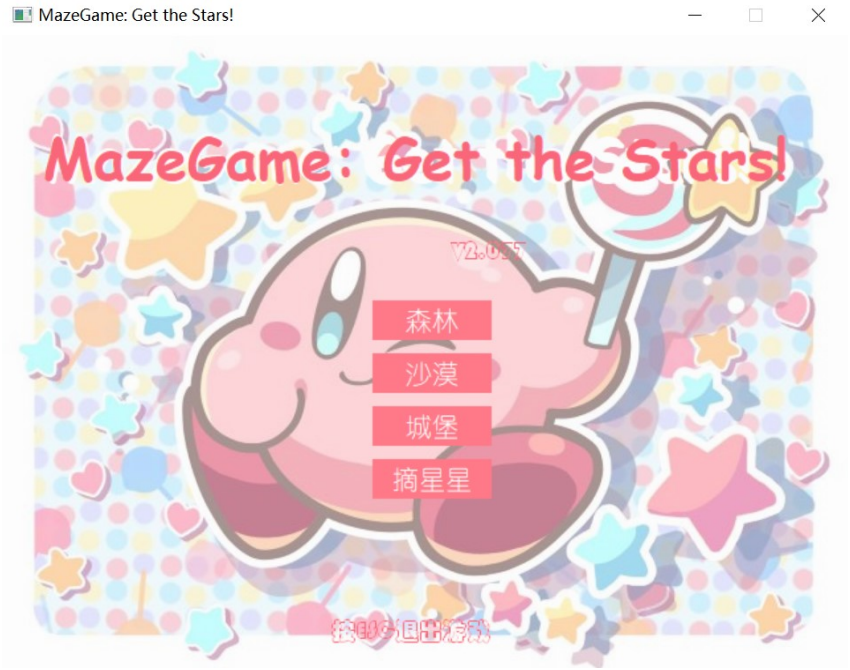
```
            {
                Connect(flag[1][i], flag[2][i], flag[1][j], flag[2][j]);
            }
        }
    }
    maze.map[2][2] = YOU;
    if (maze.size_n >= 1 && maze.size_m >= 0) {
        maze.map[maze.size_n - 1][maze.size_m] = END;    // 将 Prim 结果显示到迷宫中
    }
    }
```

(10)主要功能界面截图及概述

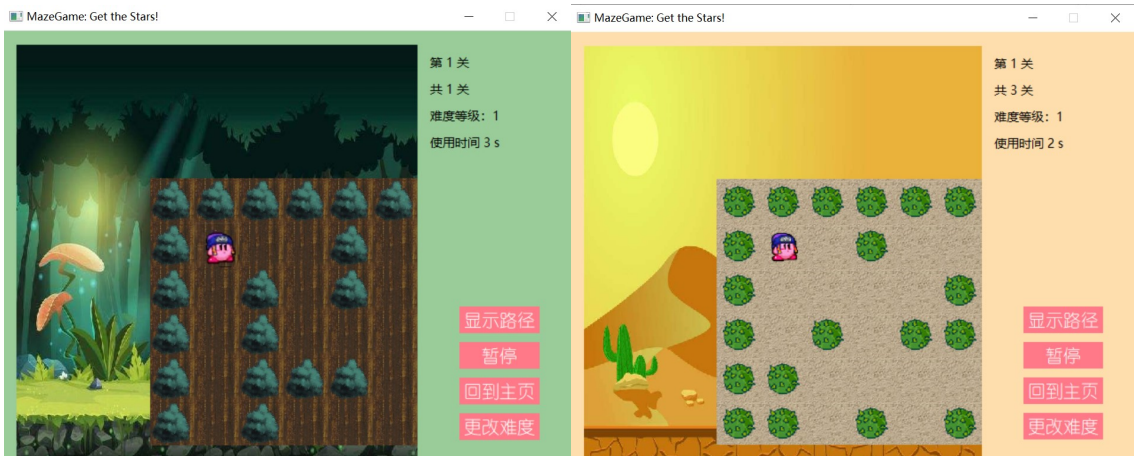
a) 初始界面

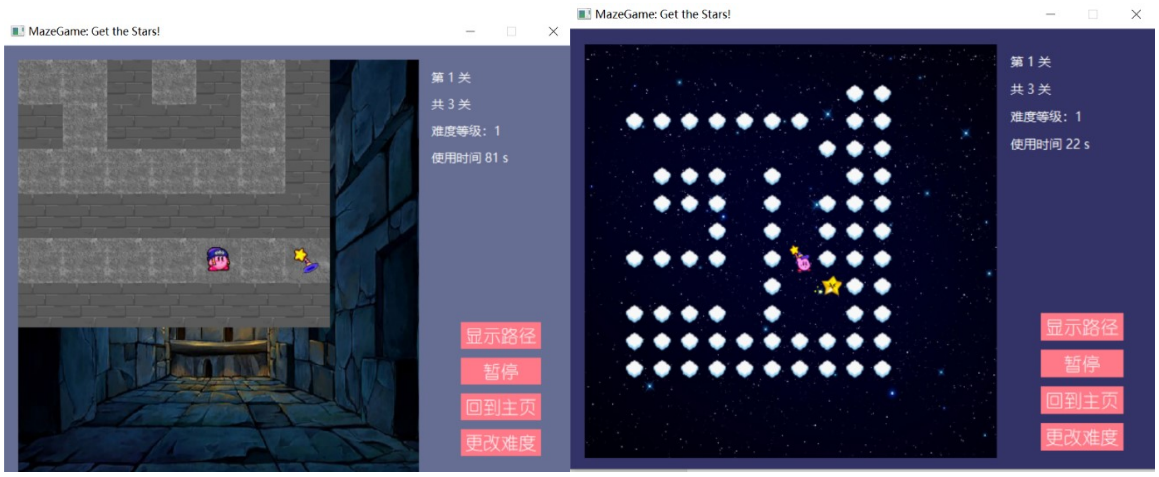
用户点击“开始新游戏”进入主界面，点击“游戏帮助”进入帮助界面。

b) 主界面：用户点击任意按钮进入对应场景。



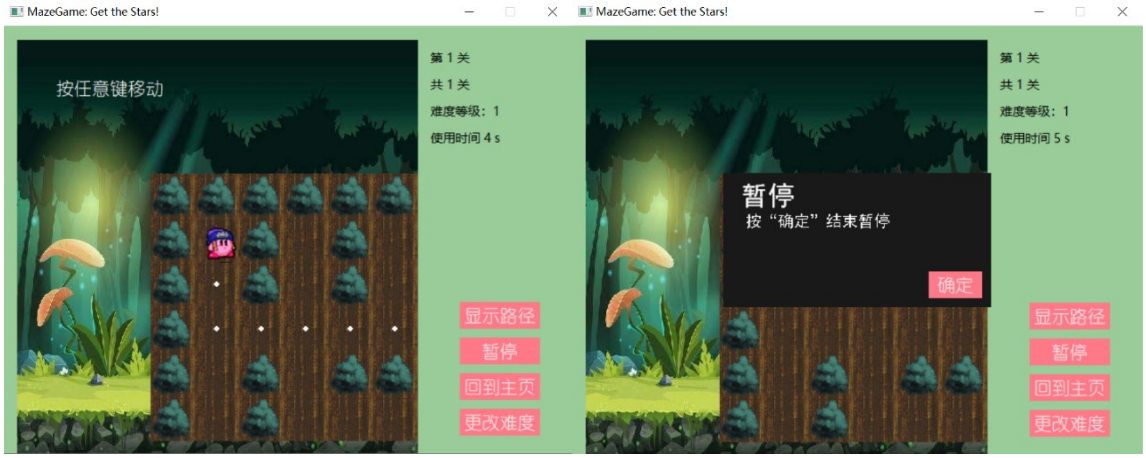
c) 场景界面：如图所示，左上、右上、左下、右下分别为森林、沙漠、城堡、星空的场景设置。





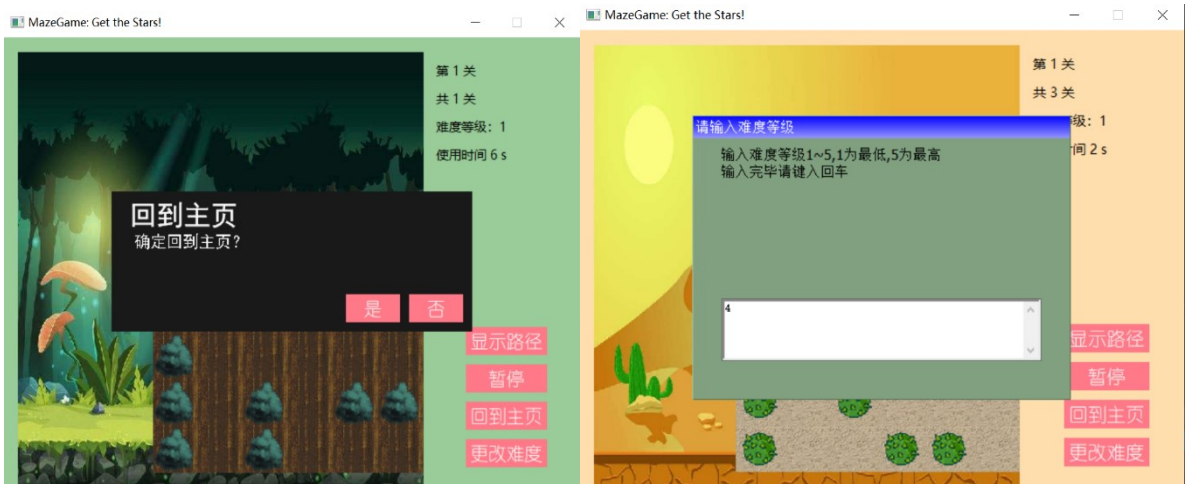
d) 路径显示功能

如图所示，单击显示路径，正确路径将以白色圆点的形式绘制在迷宫中。



e) 暂停：如图所示，点击确定按钮结束暂停。

f) 回到主页：如图所示，根据点击按钮选择实现逻辑跳转。



g) 更改难度：如图所示，在文本框中输入数字进行难度更改。

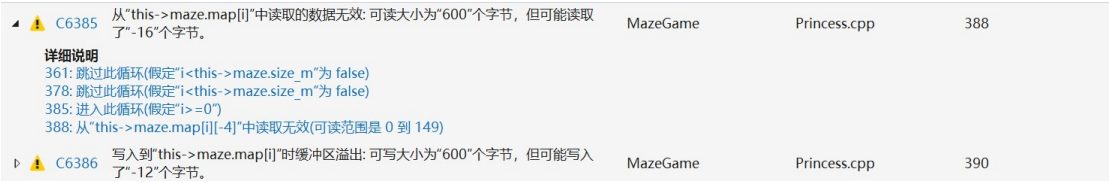
h) 通关界面：上左图、右图分别显示通过一关卡界面和通过该场景全部关卡界面。

6. 调试分析记录

（软件开发调试过程中遇到的问题及解决过程；核心算法的运行时间和所需内存空间的量化测定；符合实际情况的数据测试，算法及功能的改进设想等）

1.软件开发调试过程中遇到的问题及解决过程

（1）迷宫生成算法中存在数组边界溢出



如图所示，在已赋初值的情况下，编译器对不可能发生的数组越界情况进行告警，采取添加条件判断语句来解决，如下，则告警消失。

```
385     for (int i = maze.size_n - 2; i >= 0; i--)
386     {
387         if (maze.size_m >= 3) {
388             if (maze.map[i][maze.size_m - 3] == ROAD)
389             {
390                 maze.map[i][maze.size_m - 2] = ROAD;
391                 //返回出口所在的纵坐标
392                 return i;
393             }
394         }
395     }
```

（2）将鼠标点击事件与绘制按钮集成在一个函数中，在界面流程跳转中会出现界面停滞不动的情况，原因是在帧循环（窗口运行条件）的条件下，难以将鼠标信息清空并停留。

以下为错误代码示例：

```
bool CButton::putButton(int start_x, int start_y, char str_but[])
{
    static int x, y;
    int xClick, yClick;
    bool click_flag = false;
    //绘制边框
    setfillcolor(EGERGB(100, 100, 100));
    bar(start_x - 25, start_y, start_x + 7 * strlen(str_but) + 30, start_y + 30);

    //获取坐标
    for (; is_run(); delay_fps(60))
    {
        click_flag = false; //设置点击标志位
        while (mousemsg())  //获取鼠标信息
        {
            mouse_msg msg = getmouse();
```

```
            if (msg.is_left() && msg.is_down()) //存在左键点击事件
            {
                click_flag = true;
                xClick = msg.x;
                yClick = msg.y;
            }
        }
        if (click_flag)
        {
            //点击位置处于按钮内
        if (xClick > start_x - 25 && (size_t)xClick< start_x + 7 * strlen(str_but) + 30 &&
yClick>start_y && yClick < start_y + 30)
            {
                return 1;
            }
        }
        /*设置按钮内字体相关参数省略*/
    }
    return 0;
}
```

类似地，若是在分离函数中未将鼠标信息清空，也会出现一样的问题。下面为正确代码：

```
if (button->ifClick(xClick, yClick, 513, 350, 603, 380))
            {
                long long t1 = times;
                wchar_t* text[10];
                text[0] = L"按“确定”结束暂停\n";
                do {
                    getmouse();
                } while (!(button->putMessageBox(msg_Pause, L"暂停", text, 1, 0)));
                start_time = int(time(NULL)) - t1;
                times = t1;
                //IMPORTANT!重置标志值和已获取的鼠标位置信息，否则会导致界面停留不动
                click_flag = false;
                xClick = 0, yClick = 0;
        }
```

在上面的代码中，do-while 语句至关重要。由于 putMessageBox()绘制选框函数具有返回值，当未点选“确定”时应当将界面停留保持不变，故需要 getmouse()函数获取鼠标

信息，若未获取到则保持界面不动，否则将导致绘制的选框在 delay_fps(60)的帧循环条件下立刻被刷新。

（3）在路径选择按钮控件流程中，由于再采取鼠标点击事件作为路径结束，需要重复进行按钮绘制、点击标识符赋值和鼠标消息获取的语句，且会导致多重嵌套条件判断，故采取获取键盘操作以结束路径绘制，如下所示：

```
//仅有一次找路机会
    if (!ifhelped) {
        do {
            solveByQueue(maze.size_m, maze.size_n, Stage.num);
        } while (!(getch()));
        ifhelped = true;
    }
    else {
        do {
            outtextxy(60, 60, "机会仅有一次，请少侠继续努力~\(≧▽≦)/~");
        } while (!(getch()));
    }
    start_time = int(time(NULL)) - t;
    times = t;
    click_flag = false;
   xClick = 0, yClick = 0;
```

其中，do-while 语句和上述鼠标点击事件中 do-while 语句的作用类似。

2.内存空间的量化测定

在调试状态下，某次检测的内存空间使用情况如下：

进入初始界面，基线内存分配为 52.27MB，点击“开始新游戏”后，内存分配未 104.04MB，增加 51.78MB。进入森林场景，内存占用 259.49MB，大幅上涨。绘制路径内存仅增加 9.85KB。在回到主页二次进入森林场景后，内存涨幅较第一次进入有所下降，但仍达到 155.42MB。分别对沙漠、城堡、星空进行测定，占用内存涨幅降序排列为：星空类、沙漠类、城堡类。增长路径和更改难度都将小幅度提高内存占用率。内存峰值达 676MB，主要是由于尽管在帧循环外获取了图片，但仍然需要在帧循环中不断地绘制图像，而图形库的优化较为欠缺。

3.逻辑测试

主要对以下几种逻辑流程进行调试：

（1）各按钮控件执行流程的正确性；

（2）进入界面时进行路径绘制和移动后再进行路径绘制；

（3）同场景多次路径绘制时是否输出错误提示；

（4）难度更改的输入框内错误输入是否显示提示并能重新输入，且难度重置为 1；

（5）在难度更改后进行路径显示的逻辑保持不变；

（6）各按钮控件执行和取消执行对计时器的影响；

（7）界面暂停后对键盘信息和鼠标信息的接收检测；

（8）更改难度仅限于场景第一关的逻辑测试；

（9）生成迷宫联通测试。

4.算法及功能的改进设想

（1）迷宫算法多样性

生成迷宫的算法繁多，远不止本游戏中所应用的几种。在几种算法之外，可以增加利用并查集实现的 Kruskal 算法、Eller 算法等多种算法生成更多不同风格的迷宫。

（2）玩法扩展

迷宫在玩法多样性上仍有很大的改进空间。例如在城堡场景之中将人物的视野大小局限在一定半径的圆内以模拟烛灯，烛灯亮度随时间流逝而下降，最终归于黑暗，对玩家的时间把握提出更高的要求。此外，在迷宫中随机放置一定的道具要求玩家在规定时间内集齐方可通关，可以提供对玩家的短时正向激励。

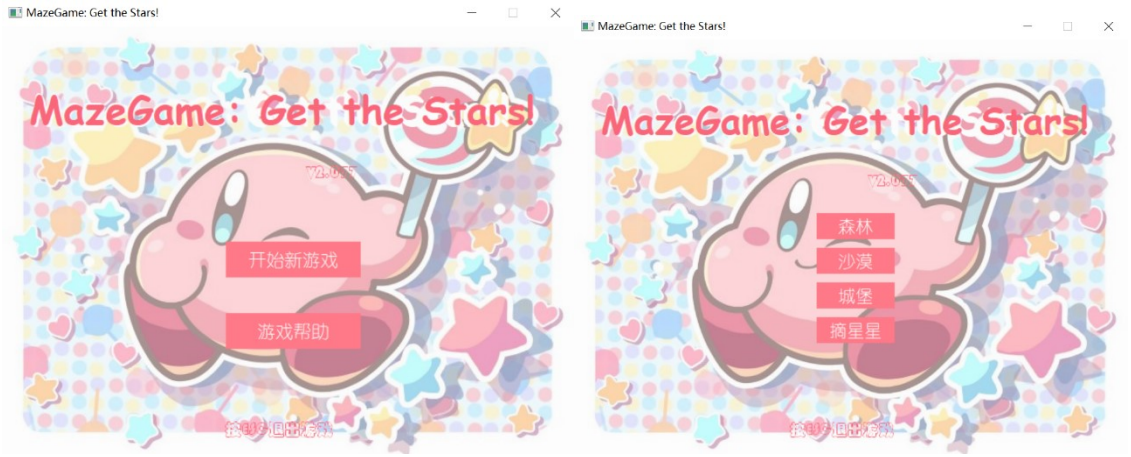
（3）迷宫形态多样性

当前迷宫仍局限于 2D 迷宫，且由于图形库绘制限制，无法采取线性风格绘制迷宫，如果对图形库有更多的了解，或是应用更高级的图形库，或许能对这种情况做出改进。

7. 运行结果与分析

（要有多组正确的实际测试数据，并给出相应运行结果截图，并对多组结果进行比较分析）

(1) 初始界面、主界面绘制



(2) 场景绘制

(3) 各按钮控件

(4) 流程控制

a) 更改难度后路径显示，路径显示次数控制

b) 通关画面

8. 教师指导建议及解决记录

（概述任课教师在开题指导、中期检查和软件验收环节中对课程任务的指导建议及各自所采取的相应解决措施和效果等）

8.1 开题指导及中期检查

指导建议：多使用数据结构，丰富玩法。

采取措施：在原定的三个场景森林、沙漠、城堡基础之上，新增星空场景，增加一种迷宫算法的使用。在玩法设定上，由原先各场景相互独立，改为游戏主角不变，以“get the stars”为主题设置贴图，使场景有一定的继承性。此外，增加对游戏难度的考虑，增加可更改难度和路径绘制的选项。

8.2 软件验收

指导建议：多使用数据结构。

采取措施：在迷宫寻路上由原定的使用队列改为使用栈和队列。

9. 总结（收获与体会）

（如实撰写课程任务完成过程的收获和体会以及遇到问题的思考、程序调试能力的培养提升等相关内容；要求不少于 500 字，严禁雷同）

本课程以项目为导向，我们选定了开发迷宫游戏，将理论所学与代码实践有机结合起来。经过根据兴趣选题、背景调研、用户需求分析、功能设计、系统架构、代码实现、修改 bug、进一步完善项目、接受检查评估等一系列软件开发的过程，我们积极参与到项目实践这一过程中来，充分发掘了我们学习、创造的潜能，提高了我们解决实际问题的综合能力。

从知识层面，首先，我学会了面向对象的语言——C++，这与我们之前学习的 C 语言在编程思想上有很大区别，它聚焦于总体架构以及类的设计。通过实践上手，我对软件开发有了更深刻的体会：1.要整体框架思路，把主要功能的细节思考清楚；2.设计类，大体分为数据类和操作类，想好成员有哪些；3.设计函数：构造、析构、重载功能函数等

等；4. 自底向上、循序渐进的方法对整个不断扩大，补充，升级。一步步进行测试，便于调试，找到错误。

从团队合作层面，学会与队员相互合作，探讨分享自己遇到的问题，一起对此进行讨论研究，才有助于项目问题的解决。同时，在项目推进过程中，只有队员之间相互督促，才能按时间轴一步步 push 进度。在遇到不能解决的问题之后，要善于去网上搜集资料，认真思考该知识点是怎么实现的，并处理变为自己掌握的东西，对能力的提升有非常大的帮助。

从思维层面，通过开发过程中的实践，我认识到程序设计需要用清晰且严谨的逻辑。既要从全局考量游戏整体的逻辑、页面跳转外，还要有对程序健壮性的考量，要对用户任意输入、任意点击情况进行特殊的处理，以免程序崩溃，这些都是十分重要的。

通过软件设计与开发实践报告 A 这门课程，不仅丰富我的编程知识，也培养了我更加缜密的思维方式。对于编程，亲手码代码才是最快的学习方式，我将通过更深入的学习，进一步培养自己的软件开发能力。