1. 选题背景与应用意义

迷宫游戏是一种经典益智类游戏。游戏中一般含有“墙”（障碍物）和“路”（通路）两种类型的方块，玩家可以在通路中移动，墙则是阻碍，一些迷宫中可能会加入可移动的障碍物（怪物）、视野限制、时间限制等特殊限制加大难度。玩家操纵角色在固定或随机的地图内移动，通过探索和尝试寻找到出口并走出迷宫来获得游戏胜利。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，较为复杂的迷宫往往对玩家的思考和记忆能力提出较高的要求，故一般而言，墙的分布趋于无序、游戏中起点到终点的连通路径在合适范围内数量增加、地图不规则程度更高等要素会提高玩家的试错成本，从而增加游戏的难度。

由于单纯的迷宫游戏一般对玩家的耐心和全局把握程度要求较高，尤其是有一定难度的迷宫更是如此，因而在设计迷宫游戏时往往会考虑在基础迷宫之上对迷宫进行一定的优化以增加可玩性。这种优化既可以体现在 UI 设计上，例如为传统的仅有单色方块的迷宫地图增加场景性元素，也可以体现在玩法设定上——尽管这些特殊设定往往进一步增加了迷宫的难度。比较常见的玩法扩展例如玩家的视野被局限在角色周边的圆内以模拟山洞中的黑暗场景，迷宫中放置可移动的怪物需要玩家在探索通路的同时躲避，设定计时模式并要求玩家在一定时间内收集到分散在迷宫各处的道具并找到出口，此外，多样的迷宫类型，传统的方形迷宫、线形迷宫、分形迷宫等都会提高游戏的可玩性。

在迷宫游戏中，图这一数据结构起到了至关重要的作用。从应用意义上讲，图可以以最直观的方式将迷宫抽象化，这一点为设计生成随机迷宫的算法、自动寻路的算法都提供了形象化思维的可能。此外，迷宫的设计将会给许多图论算法提供用武之地，这一点使得深入挖掘迷宫游戏的算法深度成为可能。而单纯以应用迷宫生成算法为目的开发的迷宫游戏往往较为简陋，或许在算法的应用上以控制台输出字符串或者单色方块的形式使迷宫显示化，但作为游戏而言显然是不够的。我们希望能在应用迷宫生成算法的同时，尝试以场景化的方式将各类算法置于一个能够串联游玩的游戏之中。考虑到迷宫游戏的核心玩法是玩家操纵角色走出迷宫，是一种较为纯粹和单调的玩法，故基于对游戏可玩性的考虑，本游戏尝试探索游戏难度的阶梯性变化和有限制的路径提示，使迷宫游戏在考验玩家耐心的同时也能给予玩家以游戏消遣。

在积累开发项目经验上，完整而独立地进行团队开发并最终产出一个成品对仅有 C 语言基础的我们是一个不小的挑战。从团队分工，软件设计，组织文件结构到编写代码，调试运行，不仅是对之前所学专业课《高级语言程序设计》《集合论与图论》《数据结构》《算法分析与设计》等综合知识的融汇掌握，也是提高我们工程能力和团队合作能力的绝佳途径。

需求分析

1.游戏的功能需求

迷宫游戏的核心功能为：玩家通过键盘按键“W” “A” “S” “D”控制游戏人物在地图中移动，游戏地图视野限制在一定大小的方框内，当玩家位于迷宫边沿时，背景以场景性元素覆盖。限定视野即通过将玩家坐标始终限定于视野中央，迷宫地图随人物坐标改变而更新来实现。

迷宫的场景性功能：玩家进入游戏后可以通过鼠标左键点击自行选择某个场景进行游玩，每个场景由不同的迷宫生成算法生成。每个场景中内置一定的关卡数，通过一关后，玩家能够选择进入下一关或是回到主页选择其它场景进行游玩。

迷宫的路径显示：当玩家点击“显示路径”按钮时，在玩家的视野范围内正确通路将以白色圆点的形式绘制在窗口上，当玩家存在任意键盘操作时，路径绘制消失。由于玩家视野有限，若是地图过大且玩家位置偏离视野过多，可能出现路径完全不显示在视野内的情况，该情况不改变玩家存在键盘操作时路径绘制的消失。出于对游戏性的考虑，玩家在各场景中有且仅有一次查看正确路径的机会，若是玩家多次点击，游戏将输出相应提示信息。

更改迷宫难度功能：当玩家点击“更改难度”按钮时，将出现输入框和提示信息，玩家按照提示信息输入难度标识并键入回车，迷宫将被重置为难度更改后的迷宫。难度更改主要通过更改迷宫大小实现。该功能仅在场景处于第一关时才有显现，即玩家仅能在各场景的第一关更改难度，且更改后的难度将延续到该场景的后续关卡。

计时器和暂停：游戏设置计时器，当玩家进入关卡即开始计时，当玩家点击暂停时计时停止，取消暂停时计时继续。玩家点击回到主页后若不继续进行选择，效果同点击暂停按钮。当玩家在观看路径提示时计时停止，停止观看路径时计时继续。更改难度后计时重新开始。

迷宫随机生成功能：在每一次重新进入某个场景或关卡，或者进行难度更改后，均通过算法进行迷宫的随机生成并重新绘制。

迷宫阶梯性难度设置：除更改难度外，在每个场景的特定难度中，游戏的难度随着关卡的推进而增加。

游戏帮助：在初始界面入口，需要设置按钮用于显示游戏帮助。

2.游戏编译环境

游戏采取 C/C++编写，在 Visual Studio 2019 中编码，需要安装 EGE(Easy Graphics Engine)图形库方能编译运行。由于图形库的限制（对.png 文件支持较弱），无法生成.exe 文件，仅能在调试模式下运行。

3.图形界面操作要求

游戏界面分为初始界面、主界面、四个场景界面、通过场景内一个关卡的过渡界面、通过一场景全部关卡的界面。

在初始界面中，绘制主界面背景图和游戏标题，玩家可以自行选择点击按钮“开始新游戏”或“游戏帮助”。若选择“开始新游戏”，则进入主界面。若选择游戏帮助，则将游戏帮助输出在窗口上，玩家可以通过键盘操作进入主界面，或使用“ESC”退出游戏。

主界面中有四个场景按钮，玩家单击任何一个按钮即可进入该场景进行游戏。

各场景界面中除游戏区贴图和场景配色外保持一致，在右侧边栏显示输出信息从上到下依次为：当前关卡数提示“第 x 关”、该场景总关卡数“共 x 关”、当前难度等级“难度等级：x”、当前使用时间“使用时间 x”（单位：s），右侧边栏下方从上到下依次设置四个按钮，分别为“显示路径”“暂停”“回到主页”“更改难度”，玩家单击任一按钮获得该事件响应。

在按钮显示中，由于玩家仅能在各个场景的第一关更改难度，故当某场景进入第一关之后，右侧边栏中将不再设置“更改难度”按钮。

当玩家点击“显示路径”时，正确路径自动在窗口中绘制，玩家作出任何键盘操作后路径消失。点击暂停时，窗口中弹出提示框，用户用鼠标点击“确定”后暂停取消。点击回到主页后，弹出复选提示框，若用户点击“是”，则游戏回到主界面，否则游戏继续。当用户点击更改难度时，窗口弹出更改难度文本框，玩家按照提示信息键入难度数值并键入回车，地图即被刷新重置。

在不同场景地图的游戏区中，迷宫的“路(ROAD)”“墙(WALL)”“出口(END)”“人物(YOU)”的贴图均有所改变，由于视野限制，当人物走到迷宫边沿时，不同场景将使用不同的场景图片进行背景填充。

当玩家通过某场景的某一个关卡时，通关过渡界面将会显示“下一关”和“回到主页”，若是玩家点击“下一关”，则将进入该场景的下一关。若玩家通过某场景的全部关卡，当玩家点击“下一关”时，将出现“恭喜通关”的提示信息，并显示“退出游戏”和“回到主页”两个按钮。当玩家点击“退出游戏”，则窗口关闭、游戏退出，点击“回到主页”则游戏回到主界面。

4.游戏总体风格

游戏名称为“MazeGame: Get the Stars!”，游戏主角采取“星之卡比”的主角贴图，在各个场景的设置中，侧边栏提示尽可能简洁，游戏主界面根据不同的场景选择贴图，贴图需综合现实感和卡通风格，整体配色选取饱和度低、明度低的颜色，并与主界面背景图颜色呼应。

系统主要功能设计

本迷宫游戏的主要功能设计模块图如下所示：

1. 界面模块

（1）初始界面及主界面

玩家进入初始界面后，可通过鼠标点击选择进入主界面或查看游戏帮助，亦可通过“ESC”按键退出游戏。进入主界面后，可以通过鼠标点击自由选择四个游戏场景之一进行游玩。

（2）场景界面

玩家进入场景界面，计时器开始计时，右侧边栏输出关卡、难度、用时提示信息。此时，玩家可以通过键盘按键“W”“S”“A”“D”控制游戏主角在地图中移动。右侧边栏放置从上至下放置“显示路径”“暂停”“回到主页”“更改难度”四个按钮，点击任意按钮触发相应事件。

（3）通过过渡界面

在通过一关卡界面中，界面显示玩家通过该关卡所用时间，玩家可自行选择进行下一关或回到主页。若通过该场景全部关卡，则玩家可选择回到主页或退出游戏。

2.功能模块

（1）难度更改模块

在任一场景的第一关，玩家可以通过鼠标点击相应按钮来更改游戏难度。游戏难度由数值确定，从 1 到 5 递增，难度递增通过地图大小的增加来改变。默认地图难度置 1，地图大小（横向/纵向单位格数）依次按难度等级向上分别增加 6、10、14、28，若输入难度不在 1~5 之间，则游戏默认置难度为 1。玩家仅能在某一场景的第一关更改难度，更改后的难度将延续至该场景的后续关卡，且难度无法再发生改变。

（2）路径绘制模块。

在任意场景界面中，玩家可以通过点击“显示路径”的按钮来显示正确通关路径。在通关路径出现在屏幕上时，计时器暂停，玩家按下任意键盘按键则路径消失，计时继续。出于对游戏性的考虑，玩家在某一场景仅有一次机会能获得路径提示，若玩家二次点击，则会出现相关提示信息。

（3）计时器及暂停模块

在任意场景界面的右侧边栏均有使用时间显示，且玩家可以通过点击“暂停”按钮来暂停计时器。当玩家点击“回到主页”且在确认框内选择“否”时，其效果等同于点击暂停。

（4）人物移动模块

玩家通过键盘按键“W”“S”“A”“D”控制游戏主角在地图中进行“上”“下”“左”

“右”方向移动。

3.算法模块

（1）迷宫生成算法模块

对应于四个场景，游戏采取四种不同的迷宫生成算法，分别是深度优先搜索算法、图论深度优先算法、随机 Prim 算法和递归分割算法。

（2）迷宫寻路算法模块

分别采用栈和队列的方式，实现在任意迷宫中寻找到从起始点到终点的路径。

核心算法设计与分析

1. 主要数据结构

（1）图

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

（2）栈

用入栈和出栈的操作来模拟寻路过程中的走入通道和回退。用栈 S 记录“当前路径”，栈顶表示“当前路径上最后遍历的路径块”，入栈操作代表“将路径块置入栈”，出栈操作代表“将前一通道块从路径中删去”。

（3）队列

用队列来实现寻路过程中的层级探索。通过一层一层地往外拓展可走的点，有序地保存了所有可达块，取出队头则可探索与之相通的位置。

2. 核心算法

（1）利用栈寻路

使用栈结构模拟的深度优先搜索是以深度为准则，先遍历一条路到底，若未能达到终点又无路可走，则退回到上一步的状态，即回溯。在每一步的试探中，都按照定义好的方向顺序依次向每一个方向前进，前进到每一个点时都判断一下该点是否为终点，并在试探的过程中，用一个辅助变量去记录路径上顶点的顺序。流程如下所示：

```
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；}
                }
    }
```

}

（2）利用队列寻路

以入口为中心，借助队列，一层一层地对该点进行周围的点进行探测，该方法将遍历所迷宫中所有可以访问的点，直到找到终点为止。在每一次对某点周围的点进行展开探测的过程中，所有被探测的子节点都会进入到一个先进先出的队列中，在子节点进入队列的同时，记录其前驱以便后续输出迷宫的通路。整个过程用队列实现：

```
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;}
     }
}
```

（3）深度优先搜索算法

基于深度优先搜索算法的原则，利用“拆墙”的方法来构建迷宫。算法使用栈作为辅助数据结构。

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

（4）随机 Prim 算法

基于在迷宫中构建最小生成树来生成迷宫地图的算法。

1. 初始化迷宫所有方块为墙.
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

（5）递归分割算法

基于构造墙壁来生成迷宫的算法。在初始化为路径的迷宫内随机生成十字墙壁，将空间分为四个子空间，然后在三面墙上各自选择一个随机点挖洞，保证四个子空间的联通。之后继续对子空间进行分割，直至空间不足以继续分割为止。

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. 在三面墙上各挖一个洞（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

系统核心模块实现

1.编程语言：C/C++

2.开发环境及支撑软件：Visual Studio 2019，配置 EGE 图形库

3.软件系统架构

如下图所示：

4.主要数据结构定义代码

（1）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>
};
```

（2）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;
```

（3）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;
```

（4）队列

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

5.核心算法定义代码

（1）利用栈寻路

```
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*/
    }
```

（2）利用队列寻路

```
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*/
    }
```

（3）随机 Prim 算法生成迷宫

```
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
    }
    }
```

（4）深度优先搜索算法生成迷宫

```
void CAdventure::CreateMaze(int x, int y)
{
int dir[4][2] = { { 1,0 },{ -1,0 },{ 0,1 },{ 0,-1 } };
int random[4];          //随机方向数组
//四个方向随机
for (int i = 0; i < 4; i++)
{
    random[i] = rand() % 4;
    for (int j = 0; j < i; j++)
    {
        if (random[i] == random[j])
        {
            random[i] = rand() % 4;
```

```
            j = -1;
        }
    }
}

if (!Check(x, y))
{
    return;
}

int back = 0;
for (int i = 0; i <= 3; i++)
{
    if (maze.map[x + dir[i][0]][y + dir[i][1]] == ROAD)
    {
        back++;
    }
}
if (back >= 2)      //判断是否形成回路
{
    back = 0;
    return;
}
maze.map[x][y] = ROAD;      //该点打通
for (int i = 0; i <= 3; i++)
{
    maze.fmap[x][y] = 1;        //标记
    CreateMaze(x + dir[random[i]][0], y + dir[random[i]][1]);
    maze.fmap[x][y] = 0;        //回溯
}
return;
}
void CDesert::myKLS()
{
if (maze.size_m >= 1 && maze.size_n >= 1) {
for (int i = 1; i <= maze.size_n; i++)  // 初始化
{
    for (int j = 1; j <= maze.size_m; j++)
    {
        maze.map[i][j] = WALL;
    }
}
}
CreateMaze(maze.size_n - 1, maze.size_m - 1);
maze.map[2][2] = YOU;
if (maze.size_m >= 2 && maze.size_n >= 2) {
```

```
    maze.map[maze.size_n - 1][maze.size_m] = END;
    maze.map[2][3] = ROAD;
    maze.map[3][2] = ROAD;
    maze.map[maze.size_n - 1][maze.size_m - 1] = ROAD;
    maze.map[maze.size_n - 2][maze.size_m - 1] = ROAD;
    maze.map[maze.size_n - 1][maze.size_m - 2] = ROAD;      // 进行 DFS 生成
}
}
```

（5）图论深度优先算法生成迷宫

```
//图论DFS搜索生成迷宫核心
void CAdventure::dfs_search(int step)
{
int random[4];
//四个方向随机
for (int i = 0; i < 4; i++)
{
    random[i] = rand() % 4;
    for (int j = 0; j < i; j++)
    {
        if (random[i] == random[j])
        {
            random[i] = rand() % 4;
            j = -1;
        }
    }
}
//随机化处理
int temp = 0, cst[4];
temp = step + 1;
cst[random[0]] = step + 1;
if (temp > maze.Nv)
    cst[random[0]] = step;

temp = step - 1;
cst[random[1]] = step - 1;
if (temp < 1)
    cst[random[1]] = step;

temp = step + (maze.size_n - 1) / 2;
cst[random[2]] = step + (maze.size_n - 1) / 2;
if (temp > maze.Nv)
    cst[random[2]] = step;

temp = step - (maze.size_n - 1) / 2;
cst[random[3]] = step - (maze.size_n - 1) / 2;
if (temp < 1)
```

```
        cst[random[3]] = step;

    for (int i = 0; i <= 3; i++)
    {
        //是否打通以及是否走过
        if ((maze.reg[step][cst[i]] == 1 || maze.reg[cst[i]][step] == 1) && !visit[cst[i]])
        {
            maze.reg[step][cst[i]] = 2; //标记打通
            maze.reg[cst[i]][step] = 2; //移除当前迷宫单元与相邻迷宫单元的墙
            visit[cst[i]] = 1;          //标记已经走过
            visit[step] = 1;        //标记已经走过
            dfs_search(cst[i]);
        }
    }
    }
    void CCastle::deepFS()
    {
    maze.Nv = 0;                    // 节点数清空
    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)
            {
                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;
            }
        }
    }
    for (int i = 1; i <= maze.Nv; i++)
    {
        visit[i] = 0;
        if (i % ((maze.size_n - 1) / 2) != 0)
        {
            maze.reg[i][i + 1] = 1;
            maze.reg[i + 1][i] = 1;
        }
        if (i <= maze.Nv - (maze.size_n - 1) / 2)
        {
            maze.reg[i][i + (maze.size_n - 1) / 2] = 1;
```

```
            maze.reg[i + (maze.size_n - 1) / 2][i] = 1;
        }
    }

    dfs_search(maze.Nv);            // 进行图论 DFS 生成

    for (int i = 1; i <= maze.Nv; i++)
    {
        for (int j = 1; j <= maze.Nv; j++)
        {
            if (maze.reg[i][j] == 2 || maze.reg[j][i] == 2)
            {
                Connect(flag[1][i], flag[2][i], flag[1][j], flag[2][j]);
            }
        }
    }
    maze.map[2][2] = YOU;
    if ((maze.size_n - 1) > 0 && maze.size_m > 0) {
        maze.map[maze.size_n - 1][maze.size_m] = END;    // 将图论 DFS 结果显示到迷宫中
    }
    }
```

（6）递归分割生成迷宫

```
void CPrincess::Create(int x, int y)
{
maze.map[x][y] = ROAD;
int dir[4][2] = { { 1,0 },{ -1,0 },{ 0,1 },{ 0,-1 } };
int i, j;
//确保四个方向随机，不是固定的上下左右顺序
for (i = 0; i < 4; i++)
{
    int r = rand() % 4;
    int temp = dir[0][0];
    dir[0][0] = dir[r][0];
    dir[r][0] = temp;
    temp = dir[0][1];
    dir[0][1] = dir[r][1];
    dir[r][1] = temp;
}
//向四个方向开挖
for (int i = 0; i < 4; i++)
{
    int dx = x;
    int dy = y;
    //控制挖的距离，由rank来调整大小
    int range = 1 + (Rank == 0 ? 0 : rand() % Rank);
    while (range > 0)
```

```
        {
            //计算出将要访问到的坐标
            dx += dir[i][0];
            dy += dir[i][1];
            //排除掉回头路
            if (maze.map[dx][dy] == ROAD)
                break;
            //判断是否挖穿路径
            int count = 0, k;
            for (j = dx - 1; j < dx + 2; j++)
            {
                for (k = dy - 1; k < dy + 2; k++)
                {   //确保是九宫格的四个位置
                    if (abs(j - dx) + abs(k -dy) == 1 && maze.map[j][k] == ROAD)
                        count++;
                }
            }
            //count大于1表明墙体会被挖穿，停止
            if (count > 1)
                break;
            //确保不会挖穿时，前进
            range -= 1;
            maze.map[dx][dy] = ROAD;
        }
        //没有挖穿危险，以此为节点递归
        if (range <= 0) {
            Create(dx, dy);
        }
    }
    }
    //初始化迷宫
    int CPrincess::init()
    {
    int i,j;
    //maze.size_m = 25;
    //maze.size_n = 25;
    for (i = 0; i <= maze.size_n; i++)
    {
        for (j = 0; j <= maze.size_m; j++)
        {
            maze.map[i][j] = WALL;
        }
    }
    //最外围设为路径，为防止挖路时挖出边界，为保护迷宫主体外的一圈墙体被挖穿
    for (i = 0; i < maze.size_n; i++)
    {
```

```
        maze.map[i][0] = ROAD;
        if (maze.size_m >= 0) {
            maze.map[i][maze.size_m] = ROAD;
        }
    }
    for (i = 0; i < maze.size_m; i++)
    {
        maze.map[0][i] = ROAD;
        if (maze.size_n >= 0) {
            maze.map[maze.size_n][i] = ROAD;
        }
    }
    //创建迷宫，（2,2）为起点
    Create(2, 2);
    //画迷宫的入口和出口，给出玩家初始位置
    maze.map[2][2] = YOU;
    for (i = 0; i < maze.size_n; i++)
    {
        if (maze.size_m >= 0) {
            maze.map[i][maze.size_m] = WALL;
        }
    }
    for (i = 0; i < maze.size_m; i++)
    {
        if (maze.size_n >= 0) {
            maze.map[maze.size_n][i] = WALL;
        }
    }
    //由于算法随机性，出口有一定概率不在（n-3,m-2)处，此时需要寻找出口
    for (int i = maze.size_n - 2; i >= 0; i--)
    {
        if (maze.size_m >= 3) {
            if (maze.map[i][maze.size_m - 3] == ROAD)
            {
                maze.map[i][maze.size_m - 2] = ROAD;
                //返回出口所在的纵坐标
                return i;
            }
        }
    }
    }
```

6.核心函数代码段

（1）主菜单界面及场景初始化

```
for(;is_run();delay_fps(60)) //检测窗口运行状态
{
```

```
    if (kbhit())//键盘消息获取
    {
        ch_msg = getch();
        if (ch_msg == 27)    //按ESC退出
        {
            closegraph();
            exit(0);
        }
    }

bool click_flag = false; //点击标志位初始置假
while (mousemsg())  //鼠标消息获取
{
    m_msg = getmouse();
    if (m_msg.is_left() && m_msg.is_down()) //存在左键点击事件
    {
        click_flag = true;  //点击标识符置真
        xClick = m_msg.x;
        yClick = m_msg.y;
    }
}
//绘制背景
button->drawBackground();
setbkcolor(WHITE);
putimage_alphablend(NULL, pimg_bg, 0, 0, 128, 0, 0, 640, 480);
//输出标题文字
for (int i = 0; i <= NUMOFSTAGE; i++)//阴影效果
{
    setfont(60, 0, "Comic Sans MS",0,0,500,0,0,0,NULL);
    setcolor(WHITE);
    outtextxy(30 + i, 60 + i, "MazeGame: Get the Stars!");
}
setfont(60, 0, "Comic Sans MS",0,0,1000,0,0,0,NULL);
setcolor(EGERGB(250,103,122));
outtextxy(30 + 1, 60 + 1, "MazeGame: Get the Stars!");
//绘制按钮
button->putButton_(280, 200, 370, 230, "森林");
button->putButton_(280, 240, 370, 270, "沙漠");
button->putButton_(280, 280, 370, 310, "城堡");
button->putButton_(280, 320, 370, 350, "摘星星");
//点击进入森林stage
if (click_flag && button->ifClick(xClick, yClick, 280, 200, 370, 230))
{
    click_flag = false; //清空鼠标消息
    xClick = 0, yClick = 0;
    Forest = new CForest();
```

```
            //设定迷宫参数
            Forest->all_pass = 1;
            Forest->single_pass = 1;
            Forest->maze.size_n = 15;
            Forest->maze.size_m = 15;
            times = 0;
            Forest->mainGame();
            delete Forest;
            Forest = NULL;
        }
        /*沙漠类、城堡类、星空类省略*/
        //字体参数设置
        LOGFONT f;
        getfont(&f);
        f.lfWidth = 8;
        f.lfWeight = 300;
        wcscpy_s(f.lfFaceName, L"华文彩云");
        f.lfQuality = ANTIALIASED_QUALITY;
        setfont(&f);
        setcolor(EGERGB(250, 103, 122));
        //主页菜单文字
        outtextxy(340, 153, "V2.077");
        outtextxy(249, 440, "按ESC退出游戏");
    }
    delimage(pimg_bg); //销毁背景图片指针
    delete button;
    button = NULL;
```

（2）“回到主页”点击事件

```
            if (button->ifClick(xClick, yClick, 513, 390, 603, 420)) //按钮参数
            {
                long long t = times;
                flushkey();//清空键盘消息
                wchar_t* text[10];
                text[0] = L"确定回到主页？\n";
                //点击确定
                if (button->putMessageBox(msg_back, L"回到主页", text, 1, MY_CHOICE))
                {
                    delete button;
                    button = NULL;
                    return;
                }
                else {
                    click_flag = false; //重置鼠标点击信息
                    xClick = 0, yClick = 0;
                    start_time = int(time(NULL)) - t; //更改计时器参数
                    times = t;
```

```
            }
        }
```

（3）“暂停”点击事件

```
        if (button->ifClick(xClick, yClick, 513, 350, 603, 380)) //按钮参数
        {
            long long t = times;
            wchar_t* text[10];
            text[0] = L"按“确定”结束暂停\n";
            do {
                getmouse();//若未获取“确定”鼠标消息，则保持等待
            } while (!(button->putMessageBox(msg_Pause, L"暂停", text, 1, 0)));
            start_time = int(time(NULL)) - t; //更改计时器参数
            times = t;
            //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;
            }
```

（4）迷宫绘制函数

```
//获取图像资源
PIMAGE pimg_castle_road = newimage();
getimage(pimg_castle_road, "JPG", MAKEINTRESOURCEA(CASTLE_ROAD));
PIMAGE pimg_castle_wall = newimage();
getimage(pimg_castle_wall, "JPG", MAKEINTRESOURCEA(CASTLE_WALL));
PIMAGE pimg_castle_player = newimage();
getimage(pimg_castle_player, "JPG", MAKEINTRESOURCEA(CASTLE_PLAYER));
PIMAGE pimg_castle_end = newimage();
getimage(pimg_castle_end, "JPG", MAKEINTRESOURCEA(CASTLE_END));
//以人物为中心绘制迷宫
for (int i = x - 4; i <= x + 4; i++) //限定迷宫视野大小
{
    for (int j = y - 4; j <= y + 4; j++)
    {
        if (i < 1 || j < 1) {
            continue;
        }
        if (maze.map[i][j] == WALL) //绘制砖墙
        {
            putimage((j - 1 - y) * 50 + 266, (i - 1 - x) * 50 + 266, 50, 50, pimg_castle_wall,
0, 0, 128, 128); //限定方格大小参数
        }
        else if (maze.map[i][j] == ROAD) //绘制路
        {
            putimage((j - 1 - y) * 50 + 266, (i - 1 - x) * 50 + 266, 50, 50, pimg_castle_road,
0, 0, 300, 300);
        }
```

```
        else if (maze.map[i][j] == END) //绘制终点
        {
            putimage((j - 1 - y) * 50 + 266, (i - 1 - x) * 50 + 266, 50, 50, pimg_castle_end,
0, 0, 100, 100);
            maze.des_x = i;
            maze.des_y = j;
        }
        else if (maze.map[i][j] == YOU) //绘制人物主角
        {
            putimage((j - 1 - y) * 50 + 266, (i - 1 - x) * 50 + 266, 50, 50, pimg_castle_player,
0, 0, 150, 150);
            x = i; //重置坐标
            y = j;
        }
    }
}
    delimage(pimg_castle_road); //销毁图像，避免内存溢出
    delimage(pimg_castle_wall);
    delimage(pimg_castle_end);
    delimage(pimg_castle_player);
```

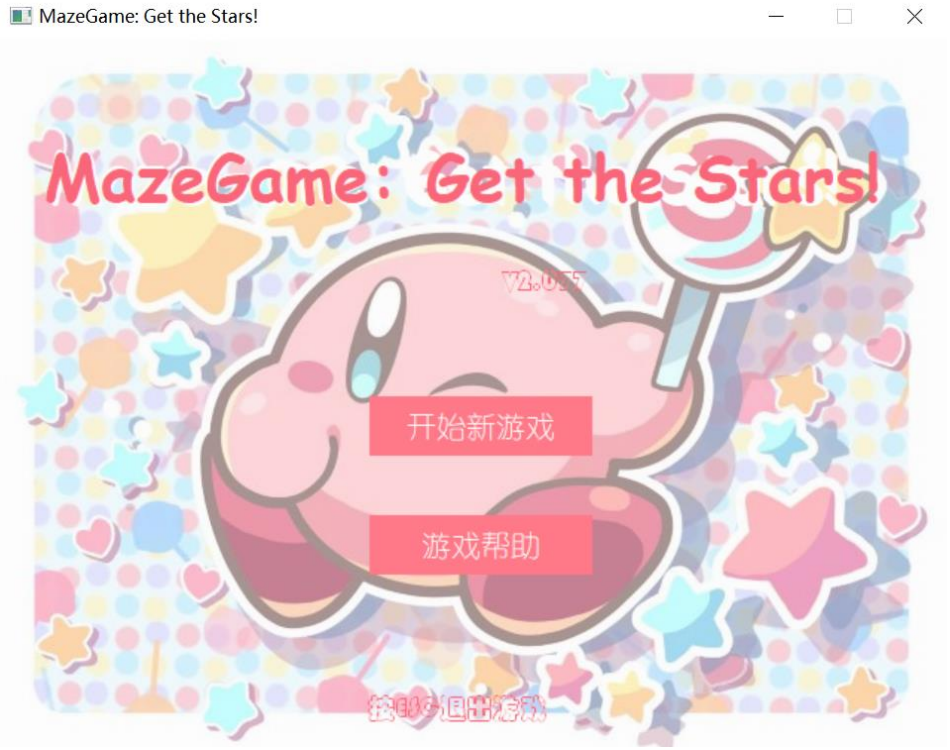
（5）“更改难度”点击事件

```
    if (single_pass == 1) //仅第一关能更改难度
        {
            button->putButton_(513, 430, 603, 460, "更改难度");
            if (button->ifClick(xClick, yClick, 513, 430, 603, 460))
            {
                click_flag = false;
                xClick = 0, yClick = 0;
                inputbox_getline("请输入难度等级", "输入难度等级1~5,1为最低,5为最高\n输
    入完毕请键入回车", strbuff, buffsize); //获取用户输入
                if (strbuff[0] == '\0') //未输入
                {
                    error = false;
                }
                else if (strlen(strbuff) > 5) {//输入字符过长
                    ifInput = true;
                    error = true;
                }
                else if (sscanf(strbuff, "%d", &GameLevel_forest) == 1) //输入正确
                {
                    error = false;
                    ifInput = true;
                }
                else {//其它输入
                    ifInput = true;
```

```
                error = true;
            }
            if (ifInput)
            {
                if (error) //输出错误提示
                {
                    setcolor(WHITE);
                    xyprintf(100, 60, "输入错误");
                    delay_ms(1000);                    }
                else
                {
                    mainGame();//重置迷宫
                }
            }
        }
    }
```

7. 主要功能界面截图及概述

（1）初始界面

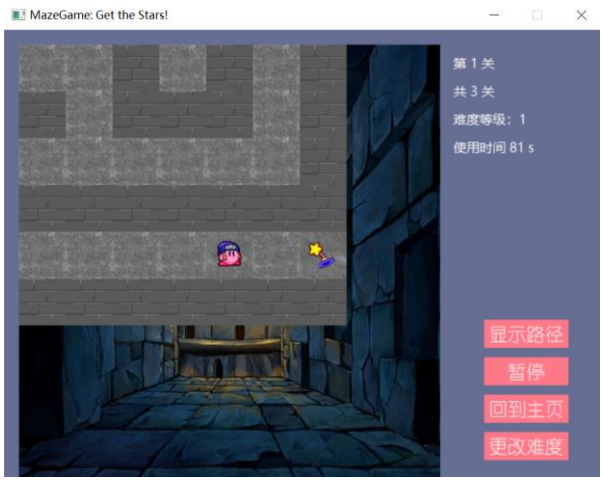


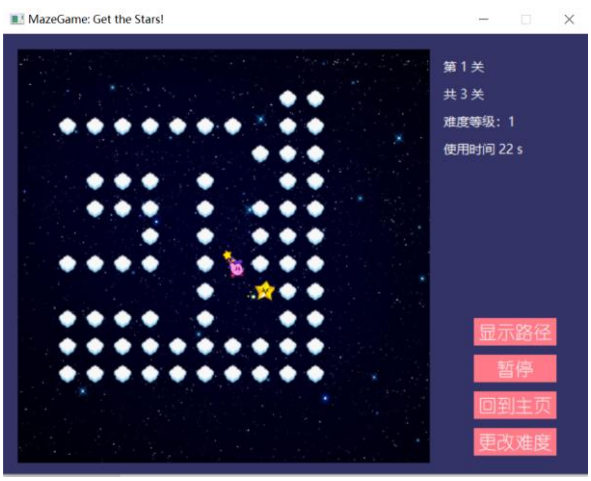
用户点击“开始新游戏”进入主界面，点击“游戏帮助”进入帮助界面。

（2）主界面

用户点击任意按钮进入对应场景。

（3）场景界面



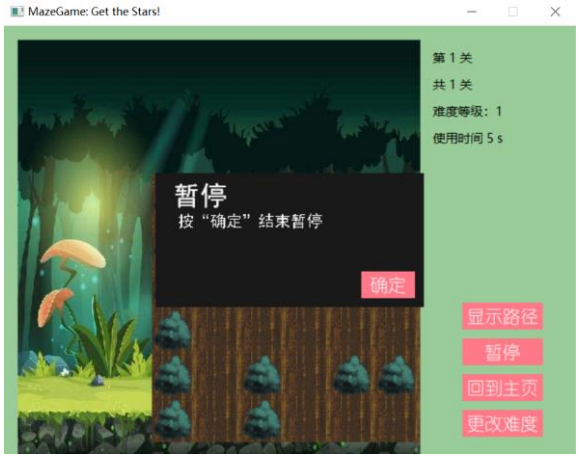


如图所示，左上、右上、左下、右下分别为森林、沙漠、城堡、星空的场景设置。

（4）路径显示功能

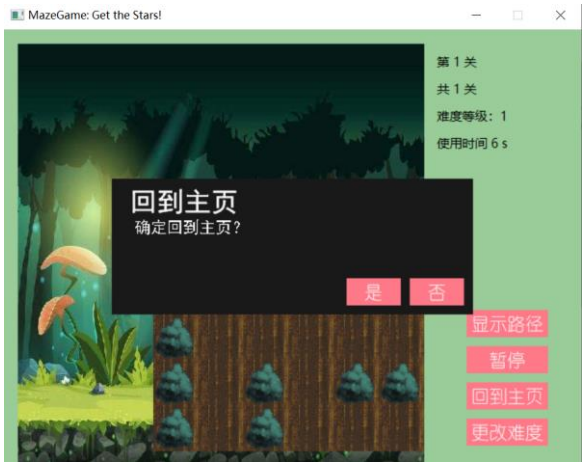
如图所示，单击显示路径，正确路径将以白色圆点的形式绘制在迷宫中。

（5）暂停



如图所示，点击确定按钮结束暂停。

（6）回到主页



如图所示，根据点击按钮选择实现逻辑跳转。

（7）更改难度

如图所示，在文本框中输入数字进行难度更改。当输入错误时，输出错误提示信息。

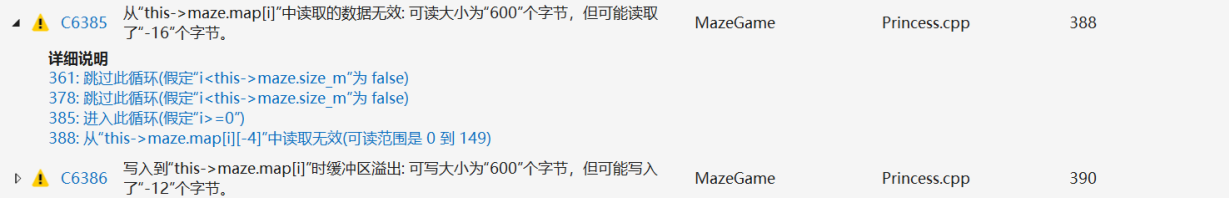
（8）通关界面

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。

调试分析记录

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：

```
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     }
```

则告警消失。

（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使界面停留。

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.核心算法的运行时间

3.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。

4.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）生成迷宫连通测试。

5.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

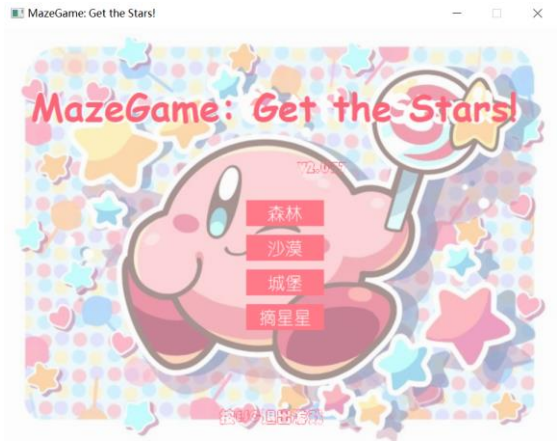
迷宫在玩法多样性上仍有很大的改进空间。例如在城堡场景之中将人物的视野大小局限在一定半径的圆内以模拟烛灯，烛灯亮度随时间流逝而下降，最终归于黑暗，对玩家的时间把握提出更高的要求。此外，在迷宫中随机放置一定的道具要求玩家在规定时间内集齐方可通关，可以提供对玩家的短时正向激励。

（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。

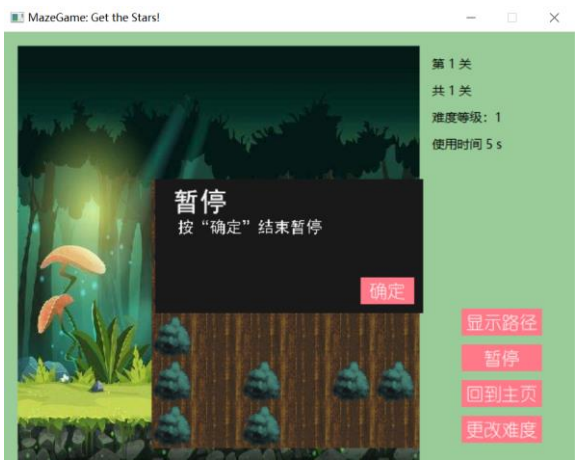
运行结果与分析

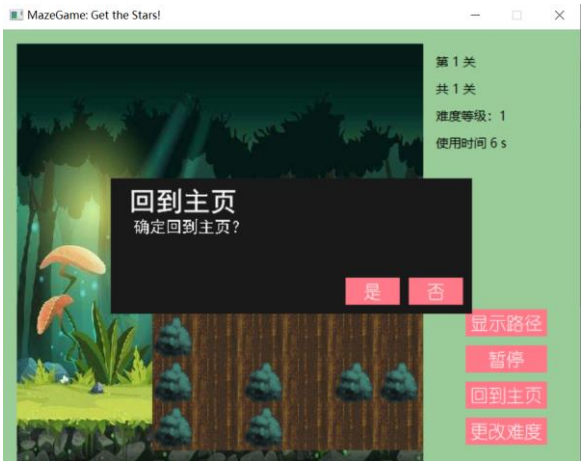
1.初始界面、主界面绘制



2.场景绘制

3.各按钮控件（暂停、回到主页、显示路径、更改难度）与通关画面

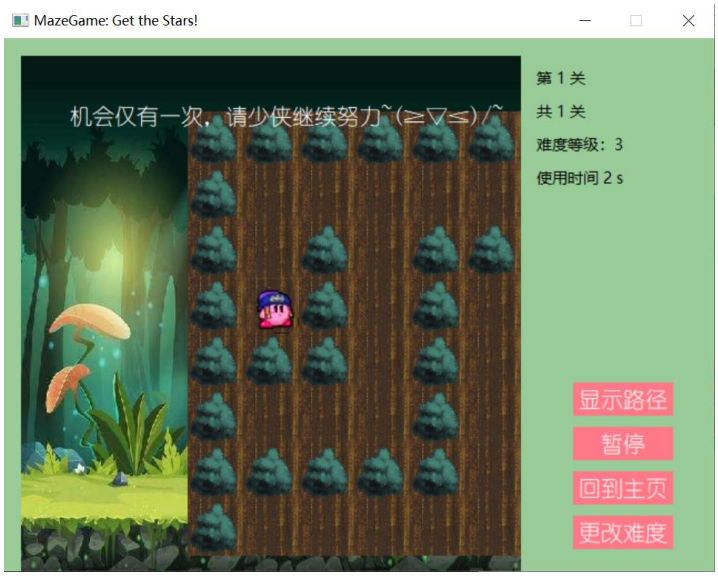




4.流程控制

（1）更改难度后路径显示

（2）路径显示次数控制



（3）文本框输入异常处理

教师指导建议及解决记录

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。

总结

软件设计与开发实践 A 这门课可以说是我进入大学学习编程以来第一次进行团队合作开发一个独立完整的软件作品。这件成品，一款延续传统又凝聚了个人风格特色的经典 2D 迷宫游戏，尽管仍旧是留有不少的遗憾，但在运行效果上可以说是基本满足设计时期我对于它的想象。从构思游戏场景、流程和玩法，设计有一定复杂度的文件依赖关系，到亲自编写代码实现控件和控制游戏流程，为游戏中的每一块砖和路和背景选定贴图，再到联调时 bug 频出，终于成功运行时发现按钮控件逻辑需要推翻重来，直至最后终于产出了逻辑较为完善、美术风格较为统一的游戏作品，这期间的每一步都使我所获颇多。

在初始的系统设计和需求分析阶段，我们最初设想的游戏功能不仅包括路径设置，还包括以人物为圆心模拟烛灯照明的山洞场景、在随机地图中生成一定的物品需要玩家在时限内收集并找到出口。但在实际编码的过程中发现，EGE 图形库的简陋程度实在是远超我的想象。仅仅在设置按钮、使鼠标左键点击能够跳转到对应流程这一个操作上，就需要分为“绘制按钮框”“在指定位置放置文字”“判断鼠标左键是否进行点击”“判断鼠标左键点击是否位于按钮框内”“跳转到对应逻辑”“重置点击标识符”这几步操作，并且由于绘图必须在帧循环中进行，将以上几步集成置于一个函数之中甚至也会导致诸多问题。这样令人无奈的问题造成了不少的重复代码。此外，在我尝试读取图片绘制时，我惊讶地发现 EGE 图形库对 png 格式的图片支持竟是如此之弱，以至于将之读取并缩放绘制在屏幕上都无法做到。在咨询了图形库开发者之后，我得知除了 PS 处理大小外其它可替代方法的效率均十分堪忧，这令我十分沮丧，也导致软件中看似是人物在移动，实则是粘贴在背景上的人物在进行移动。在如此简陋的图形库上开发一个文件依赖关系较为复杂的游戏，实在是令我深感任务艰巨，因而与队员商量后，我们暂时搁置了玩法上的拓展，转而先处理好游戏流程控制的编码。在流程控制较为完善了之后，我才尝试性地在游戏中增加了难度更改这一选项。这时，课程业已接近尾声。

在开发的过程中，由于队友在物理位置上与我相隔较近，我们能很方便地进行开发中问题的交流与探讨。在解决算法问题的过程中，我们也查阅了诸多资料，见识到了远超我们游戏中所用几种算法的迷宫生成算法，这不禁让我们感叹迷宫虽然只有“路”和“墙”两种元素，却仿佛二进制一般能构成无穷无尽的计算机世界。而当算法运行在程序中，墙与路组合成奇妙的排列，有些仿若自然生成，有些曲折复杂，有些擅长生成“一条长长的死胡同”，有些则具有美妙的分形特征，这些千奇百怪的迷宫构成了我们程序的核心，这让这个简单的、在外观上可爱而卡通的游戏有了向更深更难的方向探索的可能。越是纯粹就越是复杂，这是迷宫启发我的道理。

这次开发过程给我留下了非常深刻的印象，不仅仅是最终的成品，更在于过程中调试、解决问题时不断转换思路，尝试换个角度的思考方式，也让我们体会到几日未解决的 bug 在一两行代码的改动下突然之间消失不见时的欣喜与快乐，这种经历在激励我在开发道路上继续学习的同时，也值得珍藏和回味。