综合实训报告

题目：Go through the Maze（老鼠走迷宫）（选题 4）

一、人员和分工

姓名：蒋春鸿
学号：20222005089

二、问题描述和基本要求

程序开始运行时显示一个迷宫地图，迷宫中央有一只老鼠，迷宫的右下方有一个粮仓，粮仓内放置有老鼠最爱吃的奶酪。游戏的任务是使用键盘上的方向键（W/A/S/D）操纵老鼠在生命值耗尽之前走到粮仓处，吃到奶酪。

功能要求如下：

（1）老鼠形象可辨认，可用键盘操纵老鼠上下左右移动；

（2）迷宫的墙足够结实，老鼠不能穿墙而过；

（3）正确检测结果，若老鼠在生命值减为 0 之前走到粮仓处，提示成功，否则提示失败；

（4）添加编辑迷宫功能，可修改当前迷宫，修改内容：通路变墙，墙变通路（即增加或减少额外障碍）；

（5）增加闯关功能；

（6）找出走出迷宫的所有路径，以及最短路径；

最终呈现形式要求如下：

1、利用序列化功能实现迷宫地图文件的存盘和读出等功能；

2、要求人机界面友好，实现图形化界面；

3、撰写实验报告。

三、工具/准备工作

3.1 综合实验中涉及的数据结构知识：

栈（stack）是限定仅在表的一端进行插入和删除操作的线性表，允许插入和删除的一端称为栈顶（stack top），另一端称为栈底（stack bottom），不含任何数据元素的栈称为空栈。

任何时刻出栈的元素都只能是栈顶元素，即最后入栈者最先出栈，所以栈中元素除了具

有线性关系外，还具有**后进先出**（**LIFO**，Last in First out）的特性。

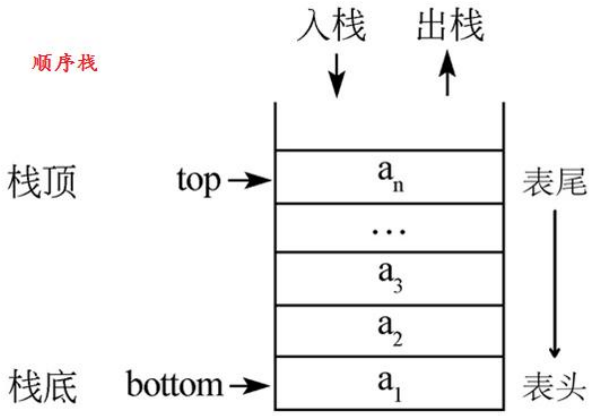

图 3.1 栈示例图

队列（queue）是只允许在一段进行插入操作，在另一端进行删除操作的线性表，允许插入（也称入队、进队）的一端称为队尾，允许删除（也称出队）的一端称为队头。队列中的元素除了具有线性关系外，还具有**先进先出**（**FIFO**，First in First Out）的特性。

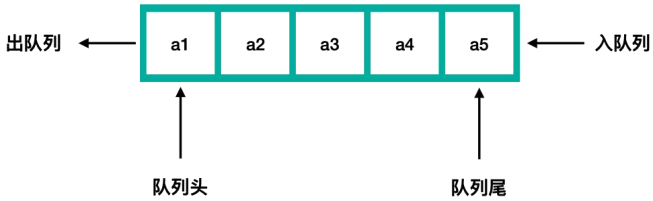

图 3.2 队列示意图

3.2 使用的 C++集成开发环境

Visual Studio 2019，EasyX 图形库等

（流程图的绘制采用 draw.io 软件绘制）

四、分析与实现

4.1 分析综合实验项目的实现方法（围绕基本要求）

要实现游戏项目的基本要求，**游戏的基本元素**有：老鼠、奶酪、关卡、通路、墙体（障碍墙）、边界墙路径标志、控制区（按钮）、提示窗口等。

考虑人机交互界面友好的基本要求，本项目选用 Visual Studio 作为平台开发，除了考虑到拥有强大的集成开发环境外，还有 Windows 平台优化的好处。该软件在 Windows 操作系统上的集成和性能通常更好，支持各种扩展和插件，允许开发者根据项目需求选择合适的工具和功能进行定制。这为不同类型的项目提供了更大的灵活性。

考虑图形化界面的基本要求，本项目选用 EasyX 图形库创建图形界面，可以导入#include<graphics.h>头文件。一开始，在游戏的全局变量处导入了老鼠、奶酪、墙体、背

景设定动画等一系列图片，并定义了图片大小、关卡数、方向、边界、得分、模式、迷宫数组等一系列全局变量，并按需赋初值。

考虑鼠标+键盘操控老鼠移动的基本要求，本项目代码中 ExMessage m;//定义一个消息变量，并利用该变量接收 EM_MOUSE 作为参数的 getmessage()的返回结果来获取鼠标消息，达到时刻监听鼠标动作的效果，鼠标动作按下后的对应游戏动作由项目代码块实现。鼠标动作包括中键弹起、中键按下、鼠标移动等。键盘动作同理。

考虑迷宫的表示与编辑的基本要求，项目选用了合适的数据结构——**二维数组**来表示迷宫地图。迷宫区域近似方形，可看成两个坐标维度，并在这两个维度上切分成一个个小网格，每个网格的状态近似成一个数值状态表示。当网格状态改变时，利用代码逻辑进行实现。

4.2 采用适当的数据结构与算法（Why、What、How）

What：本项目采用栈和队列两种数据结构，依托 **DFS**（Depth-Fist Search，深度优先搜索）和 **BFS**（Breadth-First Search，广度优先搜索）两种基础算法来进行实现。

Why：用栈的 DFS 和用队列的 BFS 解决迷宫问题，是广为适用的经典算法方法。

堆栈是一组元素的集合，类似于数组，不同之处在于，数组可以按下标随机访问，访问完 a[5]后可以访问 a[1]。但堆栈的访问规则被限制为 push 和 pop 两种操作，push（压栈或入栈）向栈顶添加元素，pop（出栈）则取出当前栈顶的元素，也就是说，只能访问栈顶元素而不能访问栈中其它元素。如果所有元素的类型相同，堆栈的存储也可以用数组来实现，访问操作可以通过函数接口来提供。

队列有两种操作，Enqueue（入队）是将元素添加到队尾，Dequeue（出队）是从队头取出元素并返回。由于是先来先服务的特性，所以先入队的元素也是先出队的。我们这里使用特殊的队列——环形队列，即把 queue 想象成一个圈，head 和 tail 指针仍然是一直增大的，当指到数组末尾时就自动回到数组开头。就像两个人围着操场赛跑，沿着它们跑的方向看，从 head 到 tail 之间是队列的有效元素，从 tail 到 head 之间是空的存储位置，如果 head 追上 tail 就表示队列空了，如果 tail 追上 head 就表示队列的存储空间满了。

How：DFS 是每次探索完各个方向相邻的点后，取其中一个相邻的点走下去，一直走到无路可走了再退回来，取另一个相邻的点再走下去。利用回溯（backtrack）的思想，堆栈中保存的应该是坐标（x, y），如果在探索问题的解时走进了死胡同，则需要退回来从另一条路继续探索。top 指针在 push 时增大而在 pop 时减小，因为栈空间是可以重复利用的。

BFS 是沿各个方向同时展开搜索，每个可以走通的方向轮流往前走一步，是一种步步为营的策略。head、tail 指针都在一直增大，虽然前面的元素已经出队了，但它所占的存储空间却不能重复利用。出队的元素仍然有用，保存着走过的路径和每个点的前趋。

4.3 给出各个函数之间调用关系

各个函数模块之间的调用关系如下：

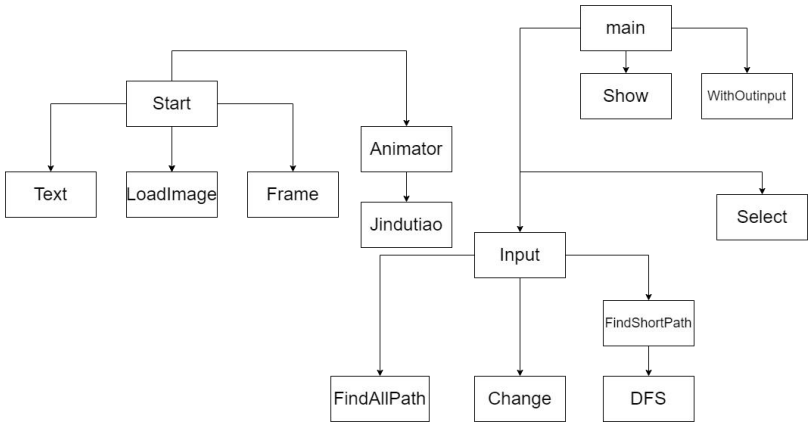

图 4.1 函数模块调用关系

Start()函数是游戏初始化函数，定义游戏的基本元素，如一系列带序号的 Texttip()函数定义一系列的提示窗口、LoadImage()执行图片加载（包括老鼠、奶酪、关卡、通路、障碍墙等基本元素）和 Frame()绘制整体框架/控制区框架函数定义控制区（按钮）的定位。

```
//-------------------------------游戏初始化函数-------------------------------
void Start(){...}

//-------------------------------游戏界面绘制函数-------------------------------
void Show(){...}

//-------------------------------鼠标交互函数-------------------------------
void Change(){...}

//-------------------------------关卡选择函数-------------------------------
void Select2(){...}

//-------------------------------登录界面函数-------------------------------
void Select(){...}

//-------------------------------播放开头动画函数-------------------------------
void Animator(){...}

//-------------------------------非手动更新函数-------------------------------
void WithOutInput(){...}

//-------------------------------玩家手动更新函数-------------------------------
void Input(){...}

//-------------------------------进度条加载函数-------------------------------
void Jindutiao(){...}

int main(){...}
```

图 4.2 部分函数缩略

执行 Start()后的 Animator()函数定义了播放动画函数，如定义每一张动画的停留时间还有跳过动画的执行方式。Jindutiao()是进度条加载函数，用于显示进度条加载画面，包括粮仓和小老鼠的图标。进度条加载完成后进入登录界面。

main()函数的主要 while 循环使得 Show()和 Input()函数不断执行，直到 key 为 3（到达最后一个关卡）时跳出循环。

在每一个 while 循环内部（每一个关卡）中，用 Show()函数定义游戏界面绘制，绘制边界墙、障碍墙、粮仓、路径高亮标记显示、四个方向通道生成、编辑模式下的绿色小网格等；用 WithOutInput()函数进行非手动更新，即编辑状态为 3 时，进入编辑模式，调用 Change()函数，通过鼠标交互进行迷宫编辑。

其中附加有 Select()函数和 Select2()函数，其中 Select()函数为选择定义了一个消息变量

用于获取登录界面的鼠标消息，等待用户选择：Play、Select、Exit。如果用户选择 Select，调用 Select2 进入关卡选择界面。Select2()函数为选关卡页面，定义 key 变量表示关卡，key 变量的值+1 即为关卡数字。

Input()函数中，用 input 记录玩家在键盘上的按键。当按键为“i”时调用 FindAllPath()函数，当按键为“c”时调用 Change()函数，当按键为“o”时调用 FindShortPath()等函数。其中，FindShortPath()函数调用 DFS()函数。

FindAllPath()和 FindShortPath()这两个函数在解决迷宫问题中非常重要。

FindAllPath()函数运用队列的数据结构来进行广度优先搜索，找出所有通路。创建一个二位数组 v[][]用于标记地图上的点是否被访问。从老鼠的起始位置开始，依次访问每个点，并试探其周围的四个方向。如果某个方向可走且未被访问，则将该点加入队列，并标记为已访问。使用特殊标记（数字 7）标记所有访问过的路径点。将标记后的路径存储在数组 v 中，并将存储路径信息的数组 v 赋值回原始地图。v[][]数组存放的是全部路径的点，所以接下来将 v 中的特殊标记赋值给 map 在到 show 函数中将数字 7 绘制出来，在用 copymap 返回 map 原貌即可。

FindShortPath()函数运用栈的数据结构来进行深度优先搜索 DFS()，找出最短路径。在复制的迷宫地图中，查找老鼠和粮仓的位置，并将老鼠的初始位置标记为 1（可通行）。根据关卡选择不同的起始点和目标点。将起始点标记为 8（表示路径），而后 DFS。DFS 中，在搜索过程中，将路径上的点标记为 8，并使用栈 sta 保存路径信息。以下用三线表形式，给出 DFS()的伪代码：

表 4.1 三线表表示 DFS()函数算法伪代码

```
DFS ( x, y, step ) //当前状态
    Begin
        if (  当前(x,y)是终点  )
            then  比较当前路径步数 step 是否小于当前最小步数 Min
            if yes
                then  更新最小步数
                将栈中的坐标按顺序出栈
                标记地图路径为 8
            else
                then  尝试四个方向
                if (  当前试探方向的下一步未超边界则是可行的  &&  未被访问过  )
                    then v[][]=7 表示已访问
                    更新坐标
                    当前坐标入栈
                    dfs(  新的 x,  新的 y,  新的 step) //递归调用
                    v[][]=0 表示未访问
                endif
            endif
        endif
    End
```

主程序流程如下图示。运行游戏项目后，抵达登录界面，用户可以选择关卡或开始游戏。开始游戏是按照关卡序号递增的顺序进入运行直至运行最后一关，而选择关卡可以选择任一序号关卡运行。当最后一关通过后，游戏正常结束，退出。玩家可以在任意环节任意时刻选择退出游戏。

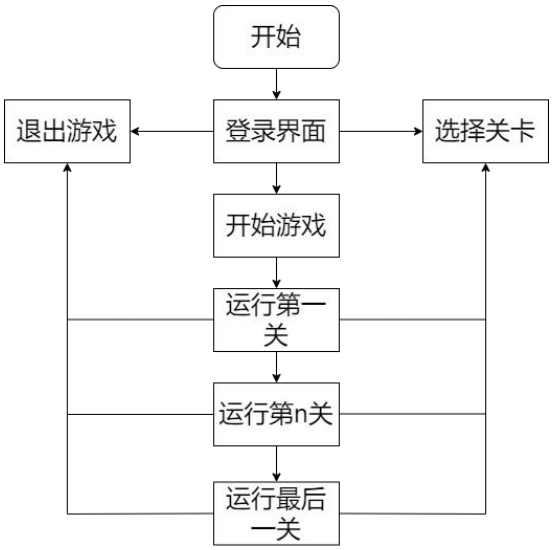

图 4.3 主程序流程图

五、测试与结论

运行程序后，等待进度条加载完毕。

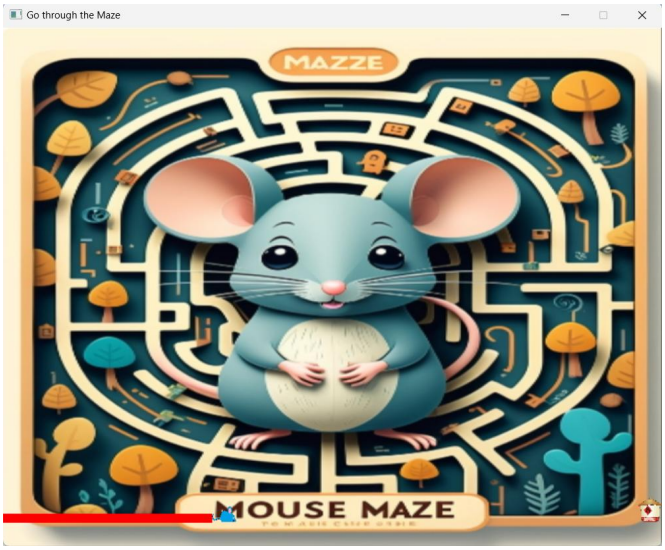

图 5.1 游戏加载窗口

加载完成后有三个选项，Play 表示开始游戏，Select 表示选择关卡，Exit 表示退出游戏。

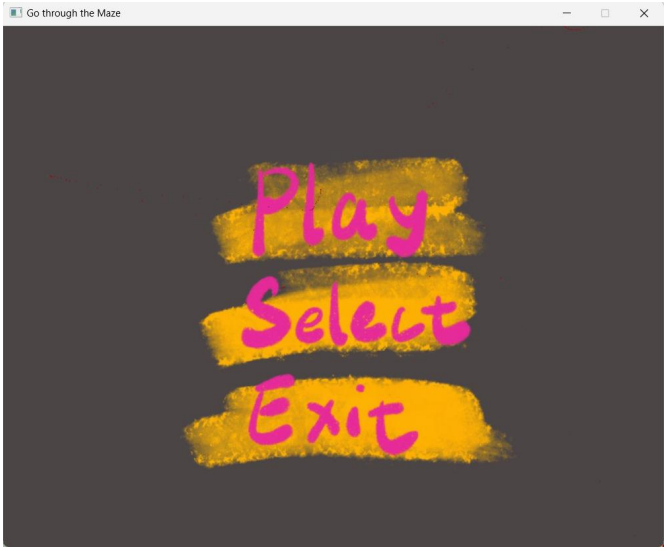

图 5.2 选择窗口

点击 Play，提示可以按下“v”键来播放动画，也可以按空格键跳过动画。

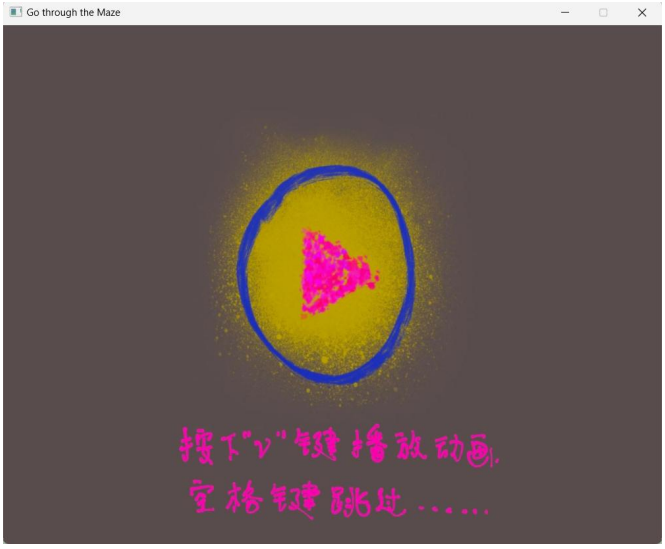

图 5.3 Play 提示窗口

按下“v”键，依次介绍的是游戏设置的背景信息。每张背景信息的停留时间为 5s。

图 5.4 背景动画窗口

背景动画放映结束后，进入游戏的主界面。在每一个关卡中，小老鼠在迷宫的中央，迷宫的右下角有一个奶酪。

现需玩家控制键盘上的 WSAD 键来控制老鼠的移动，W 表示向上走，S 表示向下走，A 表示向左走，D 表示向右走。

控制区的每一个关卡，都会有一个生命值的初始值（如第一关初始值为 200，第二关初始值为 300，第三关初始值为 300 等），表征小老鼠的剩余体力，代表小老鼠在该关卡的生命力。每移动一步（包括上下左右），控制器里的生命值都会减一；每撞墙一次，生命值也会减一。

图 5.5 游戏主界面

我们需要肉眼找到最短的路径，因为如果走的路径过长，或者一直撞墙，生命值就会减到 0，而后，小老鼠就会死亡，游戏被迫停止。

以下的环节中，鉴于前文所阐述的问题描述和功能要求，我们需要验证该游戏能实现以下功能：

表 5.1 验证内容总表

验证内容	说明
验证体力值消耗完了之后就会停止游戏	当玩家因操纵不当（一直撞墙或者找不到路而耗尽体力）导致小老鼠生命值减为 0 后提前结束当前关卡状态，游戏给出生命值耗尽提示，并终止游戏，游戏当前状态不作保存。
验证通关	当玩家通过第 N 关时，游戏给出通过当前关卡提示，并顺利过渡进入第 N+1 关卡。（当 N 最大即玩家通过最后一关时，游戏给出玩家通过所有关卡游戏结束提示。）
验证关卡选择	游戏可以提供玩家选择关卡的权力，玩家在进入游戏后，可以跳过前面的 N-1 个关卡而直接进入第 N 个关卡。
验证所有路径	当迷宫充分凌乱时，我们无法在短时间内肉眼找到所有路径，希望游戏能给出提示，提示老鼠从所在位置到粮仓奶酪处的所有路径。
验证最短路径	当迷宫充分凌乱时，我们无法在短时间内肉眼找到最短路径，希望游戏能给出提示，提示老鼠从所在位置到粮仓奶酪处的最短路径。
验证修改迷宫（路变墙）	游戏可以允许玩家设置等级难度，在每一个关卡的任意通路位置，允许将通路改变成障碍墙，从而改变迷宫属性。
验证修改迷宫（墙变路）	游戏允许将障碍墙改变成通路，从而改变迷宫属性。
验证保存并退出	游戏进行到一半但玩家想要离开的时候，需要提供一个记忆当前退出时迷宫元素属性的功能按键。

以下对表格内容进行一一验证：

当一直撞墙后，生命值减到 0，弹出提示框“小老鼠因为过度劳累饥饿而死了！”，即可验证体力值消耗完后，游戏停止。

图 5.6 验证体力值消耗完了之后就会停止游戏

当成功闯过第 1 关后，弹出提示框“恭喜通过本关！”，即可验证通关功能正常。

图 5.7 验证通关

进入游戏后，在【图 5.2】点击 Select 选项，即可由玩家自主选择点击关卡对应的数字，进入相应的关卡。

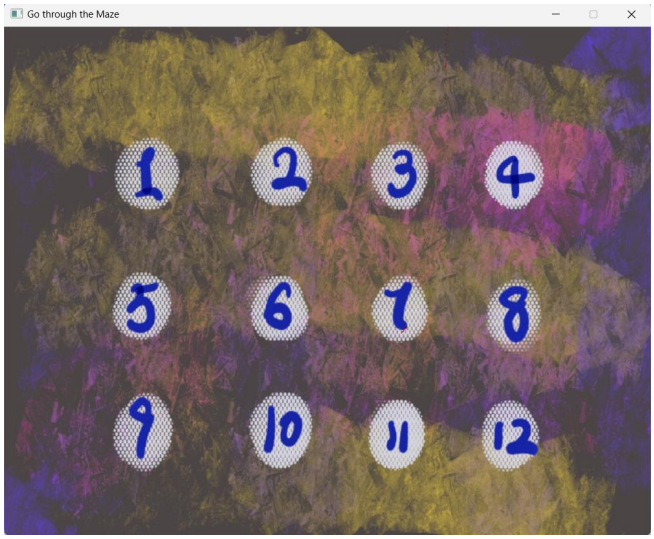

图 5.8 验证关卡选择

在关卡的游戏主界面，按下键盘上的“I”键，获取所有路径。按下后，可见最短的路径已经被用绿色标记出来了，绿色标记的路径即为老鼠在不穿透障碍墙的前提下能走到的所有位置。

有的时候我们希望取消这个标记，因为所有路径已经找到了然后又太显眼了。按下键盘上的“C”（Controller）键，可进入控制区。进入控制区后，点击“所有路径”按钮来取消高亮标记。取消标记之后，点击控制区内的“修改完成”。

图 5.9 验证所有路径

在关卡的游戏主界面，按下键盘上的“O”键，获取最短路径。按下后，可见最短的路径已经被用红色标记出来了，我们可以按照红色标记路径提示去操纵老鼠在生命值耗尽前走最短的路去拿到奶酪。

类似地，如果想取消红色的高亮标记，则按下键盘上的“C”键，进入控制区，点击“最

短路径”按钮来取消高亮标记。取消标记之后，点击控制区内的“修改完成”。

图 5.10 验证最短路径

在关卡的游戏主界面，按下键盘上的“C”键进入控制区，点击“修改迷宫”，可以看到迷宫变成了一个个绿色的小网格，意为修改迷宫的状态。可以在任意一个通路的地方点击鼠标左键，就可以把路变成墙，起到修改迷宫的作用；类似地，可以在任意一个障碍墙的地方点击鼠标右键，就可以把墙消失掉，变成路。这种情况可以强制缩短到达奶酪出的距离，强制以近似直线目标为导向通关。

修改完毕后，点击“修改完成”，绿色小网格消失，玩家可以正常进行游戏；此时，所有路径、最短路径等算法会进行重新部署，在新修改的迷宫状态上进行最短路径或所有路径的搜寻，规则仍然为障碍墙不可穿透。

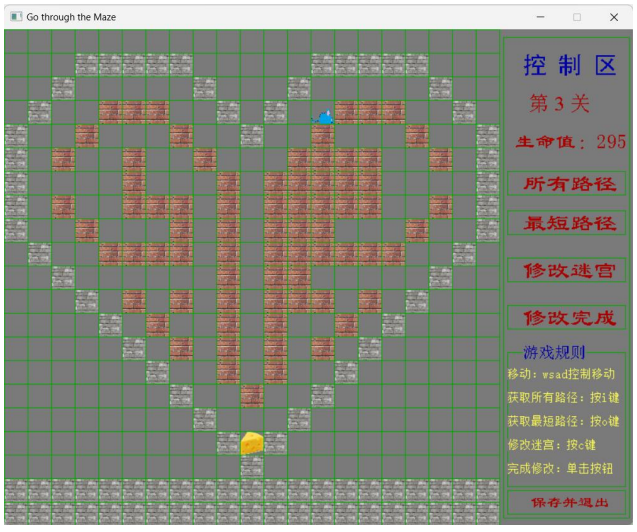

图 5.11 验证修改迷宫（路变墙）（墙变路）
（鼠标左键为路变墙，鼠标右键为墙变路）

游戏进行中还未结束时，游戏必须要记忆当前状态，等待下次玩家重启游戏进入时快速切换到记忆状态。点击“保存并退出”选项，即可实现该功能。

图 5.12 验证保存并退出

至此，【表 5.1】中的所有可能方面已被全部验证成功。
综上所述，本次提交的实验程序满足综合实验题目的要求。

六、综合实验总结

DFS 的**特点**是“全面扩散，逐层递进”，BFS 的**特点**是“一路到底，逐步回退”。DFS 不能用队列来实现，BFS 不能用栈来实现，因为 DFS 用递归实现，而递归和非递归的转换又要依靠栈进行；BFS 从起点开始，一层一层扩散，先处理完离起点近的，再处理它的下一层。根据这个特点可知它与队列先进先出的特点相吻合。使用队列保存未被检测的结点。

在项目中，我**创新性**地单独写了“EasyXPng.h”头文件，其中的 putimagePng()函数的作用是在当前设备上绘制出带透明通道的 png 图片。这个头文件很有价值，因为在主模块中的多个位置都要调用该函数，图形化界面的每个位置都需要有基本元素的图像的直观展示。其中，核心代码的作用是将图像缓冲区指针移动到绘制位置，遍历图像的每个像素，进行透明度的混合（即获取源图像的红色、蓝色、绿色分量，获取源图像的透明度，进行混合）。

本次实验项目的开发过程，使用了个人 Github 账号（主页：https://github.com/starsinhands）进行代码托管，最终成品的完整项目代码也同步上传到了个人的 Github 账号平台：https://github.com/starsinhands/Go-through-the-Maze/tree/master

图 6.1 部分代码托管日志

数据结构课程是软件工程专业所有课程中重要性排名前四的课程，是 408 专业组的核心专业课，甚至在计算机科学领域都占据有举足轻重的地位。我从大一入学即向教务出申请，做出在大一先修学习该门课程的打算。经过一个学期的修习，按照我的理解，用一句话来概括，**数据结构就是数据的存储方式**。因此，该门课程定义了数组、线性表、链表、栈、队列、堆、树、图等新结构，有了这些结构，就有了组织和存储数据的更有效有用的方式。

使用 C++来描述数据结构是有用的。C++语言内置了多种库函数方便使用者调用接口。但这次的实验，我们需要在不调用已经写好的数据结构的接口上完成，需要自己完成接口，则必须要自己弄懂底层原理，抓住最根源和最本质的知识点。在问题求解层面，以数据表示和数据处理为主，培养“问题到想法到算法到程序”的思维模式；在算法设计层面，通过伪代码描述算法，强调计算思维的培养；在算法分析层面，理解什么是“好”算法，能给出算法分析的基本方法；在存储结构层面，通过存储示意图理解数据表示，再给出存储结构定义；在程序实现层面，给出所有数据结构的 C++程序实现；在数据结构和算法的运用层面，通过项目实例理解如何为求解问题设计适当的数据结构，如何基于数据结构设计算法，从而将数据结构、算法和程序设计有机地融合在一起。可见，本项目达到了课程的教学目标。

我对这门课程对后继课程的重要性深有体会。数据结构课程所运用的思想，对后续修读的《离散数学》（已修）、《算法设计与分析》（在修）、《计算机操作系统》（在修）有紧密的联系，是众多课程的先导和基础课程。在学校的集训队进行算法训练和算法竞赛中，我们更常需要使用各种数据结构来解决问题。学习这些核心课程，不仅需要培养兴趣，更要在抽象问题上常作构思，常作想象，需要付出时间和精力。在未来的课程学习之路上，我也相信，数据结构课程给我带来的知识和阅历足以让我走得更深更远。

附项目制作参考文献如下：

EasyX 图形库下载、图形化界面制作教程，参考 csdn 博文：https://blog.csdn.net/Shun_Hua/article/details/128182566 、EasyX 官方在线文档：

https://docs.easyx.cn/zh-cn/intro

项目打包成 exe 可执行文件过程，参考 csdn 博文：https://blog.csdn.net/qingyining/article/details/128517378

（注：项目运行验证时，如果没有安装 Visual Studio，则打开“可执行文件 .exe”文件夹，打开 Setup.exe 安装项目，详细视频演示见“演示视频补充.mp4”；如果已安装好 Visual Studio，则直接打开“Go through the Maze.sln”文件即可编译运行，详细视频演示见“演示视频.mp4”。）

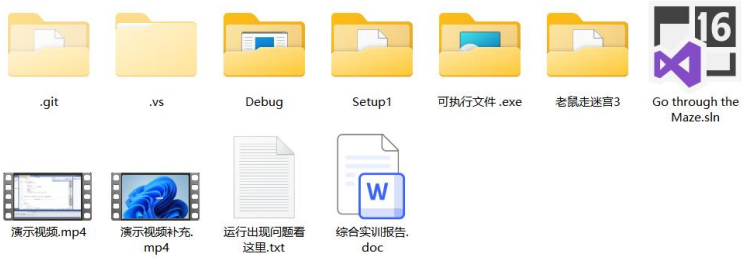

图 6.2 项目结构

七、附录 源代码（所有代码）

1.数据结构定义

```
……省略……
//3、定义坐标点结构体并申请一个队列
    struct point
    {
        int x;
        int y;
        int c;//步数
    };
    const int QUEUE_SIZE = 1000;
    point queueArray[QUEUE_SIZE];
    int front = 0, rear = 0;
    //以上三行相当于 queue<point> r;    用于申请队列
……省略……
    queueArray[rear++] = begin;//将起点入队，相当于 r.push(begin);
……省略……
while(front!=rear)//相当于 while (!r.empty())
    {
        int x = queueArray[front].x, y = queueArray[front].y;
        //取出队首元素下标，相当于 int x = r.front().x, y = r.front().y;
```

```
……省略……
                temp.c = queueArray[front].c + 1;
                //相当于 temp.c = r.front().c + 1;
                queueArray[(rear++)%QUEUE_SIZE] = temp;
                //相当于 r.push(temp);
……省略……
        front=(front+1)%QUEUE_SIZE;
        //访问完后要将队首元素出队，相当于 r.pop();
……省略……
```

以上是队列数据结构的定义。queue 是已经封装好的类，这里参考教材，把所有已经写好的类和接口以最原始的数组的形式改写。特别注意了循环队列中防止溢出的情况。

```
const int STACK_SIZE = 1000;
point sta[STACK_SIZE];
int top = -1;
//以上三行相当于 stack<point> sta;

……省略……
void push(point p) {//相当于 C++里 stack 类的 push()函数
    if (top < STACK_SIZE - 1) {
        sta[++top] = p;
    }
}
point pop() {//相当于 C++里 stack 类的 pop()函数
    point temp;
    if (top >= 0) {
        temp = sta[top--];
    }
    return temp;
}

void DFS(int x, int y, int step)//x，y 表示当前坐标点
{
……省略……
            for(int i=0;i<Min;i++){
                for(int j=0;j<Min;j++){
                    map[key][sta[top].i][sta[top].j] = 8;
```

```
                    //相当于 map[key][sta.top().i][sta.top().j] = 8;
                }
                pop();//相当于 sta.pop();
            }
        }
        while (top >= 0)pop();//相当于 for (int i = 0; i < sta.size(); i++)sta.pop();
        return;
……省略……
            push(temp);//相当于 sta.push(temp);
……省略……
        }
……省略……
```

以上是栈数据结构的定义。stack 是已经封装好的类，这里参考教材，把所有已经写好的类和接口以最原始的数组的形式改写。

2.给出实现的各个函数，要求每个函数注释实现功能。

所有函数命名和表示已经在【图 4.2】中表示。篇幅原因，以下只展示几个重要函数。

```
//------------------------------鼠标交互函数----------------------------------------
void Change()
{
    Frame2();//编辑格子
    Show();
    while (true) {
        ExMessage m;//定义一个消息变量
        m = getmessage(EM_MOUSE);//获取鼠标消息
        switch (m.message)
        {
        case WM_MOUSEMOVE:
            //  鼠标移动的时候画红色的小点
            putpixel(m.x, m.y, RED);
            break;

        case WM_LBUTTONDOWN:
        {
            if (m.x <= 790 && m.x >= 640 && m.y <= 210 && m.y >= 180)//显示所有路
径
            {
```

```
                item = 4;//取消显示所有路径
                for (int i = 0; i < bound; i++)//恢复原来的迷宫样貌
                {
                    for (int j = 0; j < bound; j++)
                    {
                        map[key][i][j] = copyMap[i][j];
                    }
                }
            }

            if (m.x <= 790 && m.x >= 640 && m.y <= 260 && m.y >= 230)//显示最短路
径
            {
                item = 4;//取消显示最短路径
                for (int i = 0; i < bound; i++)//恢复原来的迷宫样貌
                {
                    for (int j = 0; j < bound; j++)
                    {
                        map[key][i][j] = copyMap[i][j];
                    }
                }
            }

            if (m.x <= 790 && m.x >= 640 && m.y <= 380 && m.y >= 350)
            {
                item = 4;//取消编辑，完成修改
                return;
            }
            if (m.x <= 790 && m.x >= 640 && m.y <= 615 && m.y >= 585)//保存并退出
            {
                Texttip5();
                key = 3;
                return;
            }
            //通过鼠标位置计算出点击的小方块在二维数组中的下标
            int clicked_i = int(m.y) / imSize;
            int clicked_j = int(m.x) / imSize;
            //点击放墙
```

```
            if (map[key][clicked_i][clicked_j] == 1)
                map[key][clicked_i][clicked_j] = 0;
            Show();
        }
        break;
        case WM_RBUTTONUP:
        {//通过鼠标位置计算出点击的小方块在二维数组中的下标
            int clicked_i = int(m.y) / imSize;
            int clicked_j = int(m.x) / imSize;
            //点击放路
            if (map[key][clicked_i][clicked_j] == 0)
                map[key][clicked_i][clicked_j] = 1;
            Show();
        }
        break;
        case WM_MBUTTONDOWN://中键按下
        {
            int clicked_i = int(m.y) / imSize;
            int clicked_j = int(m.x) / imSize;
            //点击放路
            if (map[key][clicked_i][clicked_j] == 4)
                map[key][clicked_i][clicked_j] = 1;
            Show();
        }
        break;
        case WM_MBUTTONUP://中键弹起
        {
            int clicked_i = int(m.y) / imSize;
            int clicked_j = int(m.x) / imSize;
            //移动终点
            granarys.i = clicked_i;
            granarys.j = clicked_j;
            if (map[key][clicked_i][clicked_j] == 1)
                map[key][clicked_i][clicked_j] = 4;
            Show();
        }
        break;
        }
```

```
    }
}

//-----------------------------关卡选择函数---------------------------------------
void Select2()//选关界面
{
    putimage(0, 0, &cover5);
    setlinecolor(GREEN);
    setlinestyle(30);
    //鼠标操作
    while (true) {
        ExMessage m;//定义一个消息变量
        m = getmessage(EM_MOUSE);//获取鼠标消息
        switch (m.message)
        {
        case WM_MOUSEMOVE:
            //  鼠标移动的时候画红色的小点
            putpixel(m.x, m.y, RED);
            break;

        case WM_LBUTTONDOWN:
        {
            if (m.x <= 220 && m.x >= 135 && m.y <= 230 && m.y >= 140)//1
            {
                key = 0;
                return;
            }
            if (m.x <= 385 && m.x >= 305 && m.y <= 230 && m.y >= 140)//2
            {
                key = 1;
                return;
            }
            if (m.x <= 530 && m.x >= 450 && m.y <= 230 && m.y >= 140)//3
            {
                key = 2;
                return;
            }
        }
```

```
            break;
            }
        }
        key = 2;//0,1,2 表示关卡数
        _getch();
        return;
}

//------------------------------登录界面函数---------------------------------------
void Select()//登录界面
{
        initgraph(width, deepth);
        putimage(0, 0, &cover4);//登录界面
        setlinecolor(GREEN);
        setlinestyle(30);
        while (true) {
            ExMessage m;//定义一个消息变量
            m = getmessage(EM_MOUSE);//获取鼠标消息
            switch (m.message)
            {
            case WM_MOUSEMOVE:
                // 鼠标移动的时候画红色的小点
                putpixel(m.x, m.y, RED);
                break;

            case WM_LBUTTONDOWN:
            {
                if (m.x <= 570 && m.x >= 250 && m.y <= 280 && m.y >= 160)return;//Play
区域
                if (m.x <= 570 && m.x >= 250 && m.y <= 400 && m.y >= 280)//Select 区域
                {
                    Select2();
                    return;
                }
                if (m.x <= 570 && m.x >= 250 && m.y <= 530 && m.y >= 420)//Exit 区域
                {
                    key = 3;
                    return;
```

```
                }
            }
            break;
            }
        }
}

//------------------------------播放开头动画函数-----------------------------------
void Animator()//播放动画
{
    putimage(0, 0, &cover7);
    char input;
    input = _getch();
    if (input == 'v')//按 v 播放动画
    {
        initgraph(width, deepth);
        putimage(0, 0, &cover1);
        Sleep(5000);
        putimage(0, 0, &cover2);
        Sleep(5000);
        putimage(0, 0, &cover3);
        Sleep(5000);
    }
    if (input == ' ')return;//空格跳过动画
}
```

3.总界面（如果有）代码或者主函数代码。

```
//------------------------------游戏界面绘制函数-----------------------------------
void Show()
{
    int i, j;
    cleardevice();
    for(i=0;i<bound;i++){
        for(j=0;j<bound;j++){
            if (map[key][i][j] == 0)//绘制墙体 1
                putimagePng(j * imSize, i * imSize, &wall);
            if (map[key][i][j] == 3)//绘制墙体 2
```

```
            putimagePng(j * imSize, i * imSize, &wall_2);
        if (map[key][i][j] == 4)//绘制粮仓
            putimagePng(j * imSize, i * imSize, &granary_Png);
        if (map[key][i][j] == 7)//显示全部路径
        {
            setfillcolor(GREEN);
            fillrectangle(j * imSize, i * imSize, (j + 1) * imSize, (i + 1) * imSize);
        }
        if (map[key][i][j] == 8)//显示最短路径
        {
            setfillcolor(RED);
            fillrectangle(j * imSize, i * imSize, (j + 1) * imSize, (i + 1) * imSize);
        }
        if (map[key][i][j] == 1)
            //通道
            if (map[key][i][j] == 9)
            {
                if (direction == 4){
                    putimage(j * imSize, i * imSize, &mouse_down);
                }
                else if (direction == 3){
                    putimage(j * imSize, i * imSize, &mouse_right);
                }
                else if (direction == 2){
                    putimage(j * imSize, i * imSize, &mouse_up);
                }
                else if (direction == 1){
                    putimage(j * imSize, i * imSize, &mouse_left);
                }
            }
        if (player.i == granarys.i && player.j == granarys.j)//到达终点
        {
            Texttip();
            key++;
            closegraph();
            if (key != 3)
                Start();
            else
```

```
                {
                    Texttip2();
                    closegraph();
                    return;
                }
            }
            if (item == 3)//绘制编辑网格（绿色的小网格）
            {
                setlinecolor(GREEN);
                rectangle(j * imSize, i * imSize, (j + 1) * imSize, (i + 1) * imSize);
            }
        }
    }
    //绘制玩家图像
    i = player.i;
    j = player.j;
    if (direction == 4){
        putimagePng(j * imSize, i * imSize, &mouse_down_Png);
    }
    else if (direction == 3){
        putimagePng(j * imSize, i * imSize, &mouse_right_Png);
    }
    else if (direction == 2){
        putimagePng(j * imSize, i * imSize, &mouse_up_Png);
    }
    else if (direction == 1){
        putimagePng(j * imSize, i * imSize, &mouse_left_Png);
    }
    Texttip3();
    FlushBatchDraw();
}
```

以上为游戏主界面绘制函数。绘制了墙体、粮仓处奶酪、通道、路径、图像等。

主函数代码：

```
int main()
{
    Jindutiao();
```

```
        Start();
        while (1) {
            Show();
            Input();
            if (key == 3) break;
        }
        return 0;
}
```