浙江大学

《程序设计专题》大程序实验报告

项目名称： MAGIC MAZE

学年学期： 2022-2023 学年春学期

小组序号： 第七小组

组　　长： 卢星余 3220100425

组　　员： 黄　晨 3220101215

葛大勇 3220100494

目录

第一章 需求介绍

（1）基本功能要求

- 地图生成功能
  ①地图中至少需包含玩家、障碍物、起点、终点等元素。
  ②支持交互式手动编辑地图和自动随机生成地图。
- 迷宫求解功能
  ①手动求解，使用键盘方向键走迷宫，到达终点即胜利。
  ②程序自动求解，可视化显示最短路线、所有可行路线等，支持单步执行、自动执行等。
- 支持菜单和工具栏（快捷键）功能
  ①文件：新建地图、加载地图、保存地图、退出等。
  ②地图编辑：随机生成、手动编辑、元素选择等。
  ③地图求解：手动求解、程序求解、展示所有可行解等。
  ④帮助：使用说明。

（2）较高功能要求

- 手动求解的提示功能：提示下一步可走的格子。
- 程序求解过程可视化：除了可视化最短路径，可以将求解的过程可视化。
- 游戏性增强：给玩家添加生命值属性，增加门/钥匙道具，消耗钥匙可以打开门，在此基础上找到最优路径。

（3）额外完成的功能点

- 支持玩家与 NPC 进行交互
- 支持玩家在随机生成迷宫模式下自定义选择三种不同的迷宫难度
- 在原先门/钥匙的基础上，进一步增强游戏性，增加怪物、生命蘑菇、生命之钻等迷宫元素，引入财富 值系统
- 基于 libgraphics 和 WIN32API 增强了 libgraphics 的图形功能
- 实现独特的像素风视觉效果
- 支持背景音乐的播放与暂停功能
- 构建基本完善的背景故事和游戏世界观

第二章 功能介绍

数据结构定义

我们首先定义了一个数据结构来储存坐标

```
typedef struct {
    int x, y;
} Position;
```

然后我们定义一个数据结构来储存玩家的相关信息

```
typedef struct {
    Position position;
    int life;
    int keys;
    int wealth;
} Player;
```

然后我们定义一个数据结构来储存每一只怪兽的相关信息

```
typedef struct {
    Position position;
    int direction;
} Monster;
```

下面是地图元素相关的数据结构定义

```
typedef struct {
    Position position;
    char type;
} Element;
```

然后我们定义一个数据结构将地图的长宽与玩家、怪物、地图元素等变量储存在其中

```
typedef struct {
    int width, height;
    Player player;
    Monster monsters[MAX_MONSTERS];
    int num_monsters;
    Element **elements;
} Maze;
```

我们定义了解法结构体变量将解指针和路径长度储存其中

```
typedef struct {
    Position *path;
    int length;
} Solution;
```

最后我们使用链表来储存所有解

```
typedef struct SolutionNode {
    Solution solution;
    struct SolutionNode *next;
} SolutionNode;
```

```
typedef struct {
    SolutionNode *head;
    int count;
} SolutionList;
```

地图生成功能

①基于 dfs 算法，利用 **new_maze()**函数和 **generate_random_maze()**函数进行地图随机自动生成，利用 **set_player_start_position()** 函数和 **set_exit_position()**函数设置起点终点的位置

②通过 **draw_scene_edit_maze()**函数实现交互式手动编辑

- **new_maze()**函数：创建新的迷宫，包括玩家、障碍物、起点和终点等元素。

  ```
  Maze *new_maze(int width, int height, int num_monsters) {
  ```

  函数 创建新的迷宫(宽度，高度，怪物数)：
  为迷宫分配内存
  设定迷宫的宽度、高度、怪物数
  为迷宫的元素分配内存
  对迷宫数组内的元素进行墙壁或空地的赋值
  设置迷宫的起点（位置[1][1]）类型为 'S'
  设置迷宫的终点（位置[宽度 - 2][高度 - 2]）类型为 'E'
  初始化怪物的位置，所有怪物的初始位置都设为（0，0）

  返回迷宫

- **generate_random_maze()函数：自动生成迷宫。**

  ```
  Maze *generate_random_maze(int width, int height, int num_keys, int num_doors,int num_mushrooms,int num_coins);
  ```

  它首先确保输入的迷宫宽度和高度都是奇数。然后，创建一个全新的迷宫，并将其全部填充为墙壁。

  接着，函数使用深度优先搜索算法来形成迷宫的路径，并将路径存储下来。

  然后，函数根据路径在迷宫中随机放置指定数量的钥匙和门。
  紧接着，设置玩家的起始位置和迷宫的出口位置。
  最后，在迷宫中随机放置指定数量的蘑菇和金币。

- **set_player_start_position() 函数：在迷宫的起始位置（1，1）设定玩家的起始点。**
- **set_exit_position()**函数：在迷宫的右下角设定出口点。
- **draw_scene_edit_maze()**函数：实现交互式手动编辑地图功能
  引用按钮功能进行地图编辑

迷宫求解功能

①主要通过 **move_player()**函数实现手动求解，通过使用键盘方向键走迷宫，通过 **check_victory()**函数进行胜利检测

- **move_player()**函数：玩家移动函数
  对结构体中储存玩家 x，y 方位的值进行修改
- **check_victory()函数：胜利检测函数**
  对玩家所处位置是否是数组中终点位置进行判定

②通过 **find_shortest_path()**函数和 **solve_maze ()**函数实现可视化显示对短路线以及所有可行路线求解

- **find_shortest_path()**函数：寻找迷宫中的最短路径。使用 bfs 算法结合 **is_valid_move(Maze *maze, Position pos, int keys)函数求解，同时使用 Solution 结构体储存解。**
- **下面是对求出所有解的功能实现的描述**
  首先我们定义一个链表结构体来储存所有解

```
typedef struct SolutionNode {
 Solution solution;
  struct SolutionNode *next;
} SolutionNode;
```

  然后我们使用如下程序进行求解，使用了深度优先搜索（dfs 算法）进行求解。

```
void dfsmaze(Maze *maze, SolutionList *solution_list,
Position pos, Position *path, int path_len) ;
void solve_maze(Maze *maze, SolutionList *solution_list);
```

  由于迷宫的特殊性，故所有解的数目一般维持在 300 个以上，故采取快捷键唤出控制台输出，而不是采用按照一定频率刷新图形界面输出

③支持单步执行和自动执行

其核心还是 **find_shortest_path()**，利用其求得迷宫的最短路径，然后通过 **draw_scene_mazegame_with_auto_path(Solution *solution)**函数实现绘制以及移动玩家，单步执行采用按动一次按钮调用一次 **draw_scene_mazegame_with_auto_path(find_shortest_path(maze))**，自动执行则依赖于 libgraphics 库的时间回调函数间隔指定时间调用一次 **draw_scene_mazegame_with_auto_path(find_shortest_path(maze))。**

支持菜单和工具栏（快捷键）功能

- **Maze *load_maze(const char *filename)**函数，这个函数的目的是从指定的文件名中读取迷宫数据并将其加载到程序中。步骤如下：

  首先，尝试打开指定的文件。如果文件无法打开，函数将返回一个错误信息并终止。

  从文件中读取三个整数，它们表示迷宫的宽度、高度和怪物的数量。然后创建一个新的迷宫，尺寸和怪物的数量是已知的。

  接下来，对于文件中列出的每个怪物，函数会读取其位置并将其存储在迷宫中。

  然后，函数将读取和处理迷宫的每一个元素。如果遇到代表玩家的'S'字符，它也会保存玩家的位置。

  最后，关闭文件并返回已加载的迷宫。

- **void save_maze(Maze *maze, const char *filename)**函数，此函数的目标是将迷宫的当前状态保存到指定的文件中。步骤如下：

  尝试打开指定的文件，如果无法打开，则返回一个错误信息。

  在文件中写入迷宫的宽度、高度和怪物的数量。

  对于迷宫中的每个怪物，将它的位置写入文件。

  接下来，对于迷宫中的每个元素，将其类型写入文件。如果元素没有类型，就写入一个空格字符。

  关闭文件。

- 地图编辑功能，**void change_elements(Maze *maze, int x, int y, char element)**函数实现编辑元素的选择，**draw_scene_edit_maze()**函数实现交互式手动编辑地图功能

  `void change_elements(Maze *maze, int x, int y, char element) ;`

  此函数的目标是更改迷宫中指定位置的元素类型。步骤如下：

  首先，函数会检查给定的坐标是否在迷宫内。如果不在，它将打印一个错误信息并退出。

  如果给定的坐标在迷宫内，它将改变指定位置的元素类型，并打印出一个调试信息来表示元素已经被改变。

  `void draw_scene_edit_maze();`

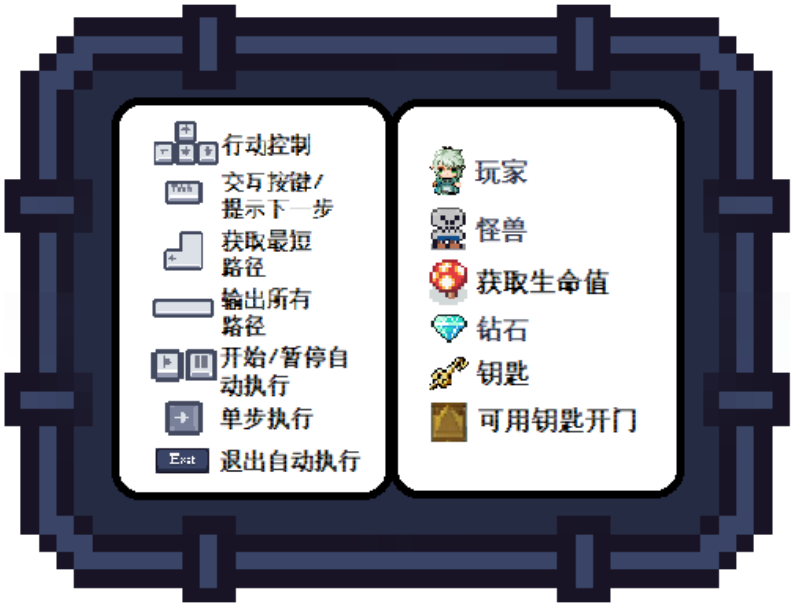
  这个函数的目标是在迷宫编辑界面中绘制迷宫和相关的按钮。步骤如下：

  设定当前界面为编辑迷宫界面，并绘制背景图片。

  创建并绘制若干个按钮，包括确认按钮、开始按钮和保存按钮。

  创建并绘制几个文本框，用于输入 X 轴坐标、Y 轴坐标和要替换的元素。

  调用 draw_maze 函数，以在特定位置和尺寸绘制迷宫。
- button【7】exit 按钮实现退出功能
- 帮助：使用说明界面



手动求解的提示功能

我们首先使用 **find_shortest_path(Maze *maze)**函数找到迷宫的最短解，然后传入到相应的场景绘制函数当中，但是和提示最短路径时不同的是，我们只绘制玩家的下一步而不是接下来的所有路径，从而实现下一步提示功能。

程序求解过程可视化

我们首先使用 **find_shortest_path(Maze *maze)**函数找到迷宫的最短解，然后传入到该场景下的界面绘制函数当中。同时在绘制的时候我们让玩家移动到解的第一个位置。在此基础上，我们依据全局的时间回调函数，按照一定时间间隔完成上述流程，实现程序求解在图形界面上的可视化。

游戏性增强

增加生命值属性、门/钥匙道具

定义了相关数据结构，同时在碰撞检测函数里面设计了相关元素的碰撞逻辑。

当玩家或者怪兽发生移动的时候，会进行碰撞检测。如果是钥匙，那么玩家携带的钥匙数会增加，如果是门，则会消耗钥匙来打开，如果钥匙不足，则无法打开门。

支持玩家与 NPC 进行交互

我们在图形界面上设置了几位游戏NPC（包括游戏引导NPC和彩蛋NPC），同时给他们划定了各自的交互响应区域。当玩家处于该区域之内，同时在键盘上按下交互按键，则会触发相应的交互效果。

支持玩家在随机生成迷宫模式下自定义选择三种不同的迷宫难度：

我们设计了一个难度选择界面，通过点击三个代表难度的按钮，我们会传入三个不同的 model 参数到生成迷宫的界面，每个不同的参数包含各自的一套迷宫生成预设参数。

进一步增强游戏性、引入财富值系统：

增加生命值属性相关元素、财富值系统、引入怪兽角色

定义了相关数据结构，同时在碰撞检测函数里面设计了相关元素的碰撞逻辑

当玩家或者怪兽发生移动的时候，会进行碰撞检测。玩家在遇到怪兽时生命值会减少，拾取生命蘑菇时生命值会增加，拾取生命之钻则会增加财富值。

支持背景音乐的播放与暂停功能：

我们在菜单界面当中创建了一个”sound”按钮，点击就会进入到相关界面，我们在界面当中创建了一个暂停和一个播放按钮，当点击播放按钮就传入音频文件的地址实现播放，点击暂停按钮则会传入一个空的地址实现暂停播放效果。

第三章 特色功能说明

一、像素风视觉效果（见运行截图）

二、迷宫游戏性和可玩性增强

进一步增强游戏性，增加怪物、生命蘑菇、生命之钻等迷宫元素，引入财富值系统。玩家在遇到怪兽时生命值会减少，拾取生命蘑菇时生命值会增加，拾取生命之钻则会增加财富值。同时，生命值为 0 则游戏失败。已整合进碰撞检测函数

三、创新性实现 libgraphics 图形库功能增强

①基于 WIN32 API 和 libgraphics 库，通过自建函数，实现了类似于带透明度参数的 PNG 文件贴图功能的 BMP 位图文件贴图函数

其特点是使用了 COLORREF crTransparent = RGB(255, 255, 255);将白色设置为透明色传入 TransparentBlt()函数，从而实现伪透明度贴图。

① 允许开发者自定义创建带贴图的按钮

在 WIN32API 创建按钮的基础上增加了传入图片的参数

② 允许开发者自定义创建可交互的文本框

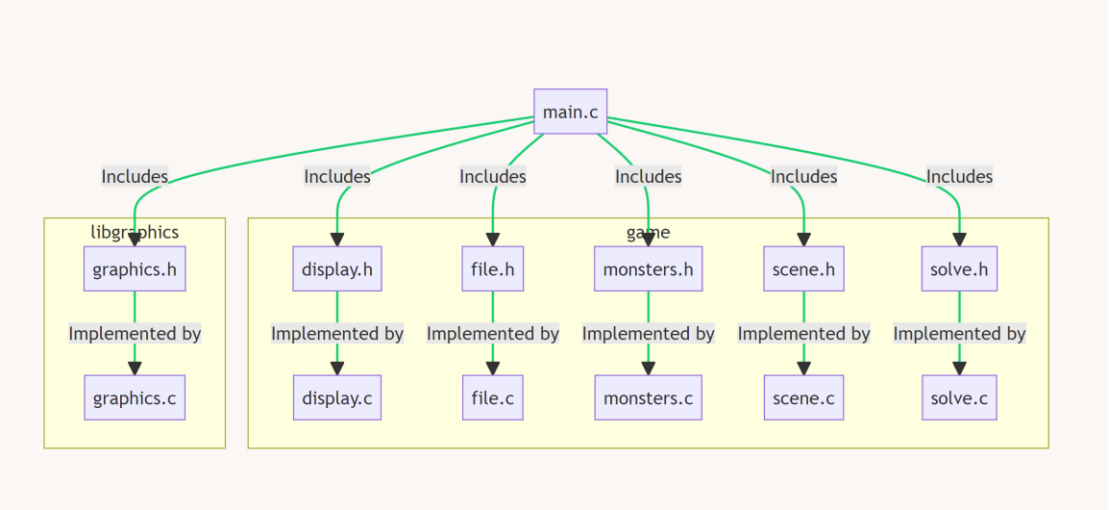
基于 WIN32API 和 libgraphics 实现，具体查看代码

③ 内嵌按钮以及场景转化逻辑到 GraphicsEventProch 函数

实现了场景之间的转换和交互。

第四章 项目文件说明

项目主要架构



main.c 文件，主要定义了一些关键函数和全局变量，主要进行了游戏的初始化、键盘事件的处理以及定时器事件的处理。

全局变量

*Maze maze：定义了一个迷宫地图的结构体，用来存储迷宫的信息。

SolutionList solution_list：定义了一个迷宫解的链表结构，用来存储迷宫的解。

int height, width, num_keys, num_doors, num_mushrooms, num_coins, num_monsters：定义了迷宫的一些参数变量，如高度、宽度、钥匙数、门数、蘑菇数、金币数以及怪物数。

int scene, mode, inmenu, musicmode, direction：定义了一些全局的状态变量，如当前场景、模式、是否在菜单、音乐模式以及玩家的移动方向。

函数

Main 函数：进行游戏的初始化，包括生成随机迷宫、设置窗口大小、初始化图形界面、设置玩家位置、绘制初始界面、注册键盘和定时器事件等。

KeyboardEvent 函数：处理键盘事件，包括在不同场景下根据按键信息移动玩家、绘制迷宫、求解迷宫等。

TimerEvent 函数：处理定时器事件，主要是控制怪物的移动频率以及在程序自动求解迷宫时进行迷宫的绘制

display.h 文件，主要负责 display.c 文件中函数的函数声明。

display.c 文件，主要负责绘制迷宫的显示部分。

1. **ClearDisplay** 函数：清除当前活动窗口的内容，准备绘制新的内容。
2. **draw_maze** 函数：根据迷宫的数据结构，绘制迷宫的每一个元素（包括玩家、墙壁、起点、终点等）。每个元素都有对应的图片文件进行展示。
3. **draw_maze_with_path** 函数：在 **draw_maze** 的基础上，增加了绘制路径的功能，可以用于程序自动求解迷宫时，将解决路径展示出来。
4. **draw_maze_with_next_path** 函数：这是一种特殊的路径绘制，只绘制下一步的路径，可以用于手动求解时，提示玩家下一步可以走的路径。
5. **draw_maze_with_path_and_auto_move** 函数：该函数实现了绘制带有路径和自动移动效果的迷宫界面的功能。

file.h 文件，主要负责 file.c 文件中函数的函数声明。

file.c 文件，这些代码的主要功能是生成一个迷宫游戏并对迷宫进行一系列操作的实现，包括生成迷宫、保存和加载迷宫以及改变迷宫元素等等。

1. **place_mushrooms** 和 **place_coins** 函数：在迷宫中随机位置放置指定数量的蘑菇和硬币。
2. **new_maze** 函数：创建一个新的迷宫，包括墙壁、开始位置和结束位置，以及随机数量的怪物。
3. **make_accessible** 函数：确保迷宫的指定位置的上下左右四个方向都可以到达。
4. **dfs_generate** 函数：使用深度优先搜索算法生成迷宫。
5. **is_passage** 和 **is_wall** 函数：判断给定坐标的元素类型是否为通道或墙壁。
6. **fix_doors** 函数：修复迷宫中所有门的位置，确保每个门都至少与两面墙相邻，且门后必有通道。
7. **place_keys_and_doors** 函数：在迷宫中随机放置指定数量的门和钥匙。
8. **set_player_start_position** 函数：在迷宫的起始位置设定玩家的起始点。
9. **generate_random_maze(int width, int height, int num_keys, int num_doors,int num_mushrooms,int num_coins):**这个函数用来生成一个随机迷宫。首先，它会确保宽度和高度都是奇数，然后创建一个新的迷宫并填满墙壁。然后，它使用深度优先搜索(DFS)生成迷宫并记录路径，放置钥匙和门，设置玩家的起始位置，设置出口位置，放置蘑菇和硬币。最后，它返回生成的迷宫。
10. **save_maze(Maze *maze, const char *filename):**这个函数将迷宫保存到指定的文件中。首先，它尝试打开指定的文件，如果不能打开，它会报告错误。然后，它将迷宫的宽度、高度和怪物数量写入文件。接着，它将每个怪物的位置写入文件。最后，它遍历迷宫的每个元素，并将其类型写入文件。
11. **load_maze(const char *filename):**这个函数从指定的文件中加载迷宫。首先，它尝试打开指定的文件，如果不能打开，它会报告错误并返回 NULL。然后，它从文件中读取迷宫的宽度、高度和怪物数量，并创建一个新的迷宫。然后，它读取每个怪物的位置并设置到迷宫中。最后，它遍历迷宫的每个元素，并从文件中读取其类型。
12. **change_elements(Maze *maze, int x, int y, char element):**这个函数用于更改迷宫中的元素。它接收一个迷宫、两个坐标和一个字符作为参数，然后检查坐标是否在迷宫的范围内。如果是，它就将指定的字符设置为该位置的元素类型。如果不是，它就报告无效的坐标。

monster.h 文件，负责 monster.c 文件中函数的声明。

monster.c 文件，这些函数用于在迷宫中生成和移动怪物。

1. **clear_walls_around_position(Maze *maze, Position pos):**这个函数清除了指定位置周围一格范围内的墙壁。它通过在指定位置的周围循环，并检查每个位置是否在迷宫的边界内，然后检查这个位置是否是墙壁或门，如果是，就将其清除。
2. **generate_monster_in_maze(Monster *monster, Maze *maze):**这个函数在迷宫中随机生成一个怪物。它通过随机选择一个位置，直到这个位置是空白的，然后随机选择一个方向，将怪物放置在这个位置，并设置它的方向。
3. **generate_monsters_in_maze(Maze *maze, int num_monsters):**这个函数在迷宫中随机生成指定数量的怪物。它首先检查要生成的怪物数量是否超过了最大限制，如果超过了，就将其设置为最大限制。然后，它循环生成指定数量的怪物，并清除每个怪物周围的墙壁。

4. **move_monster(Monster *monster, Maze *maze,int order):**这个函数移动迷宫中的一个怪物。它首先获取怪物当前的位置，然后随机选择一个方向。如果这个方向是墙壁或门，怪物就反向移动，否则就向选择的方向移动。
5. **move_monsters(Maze *maze):**这个函数移动迷宫中的所有怪物。它通过循环调用 **move_monster** 函数来移动每个怪物。

scene.h 文件主要负责 scene.c 文件中函数的函数声明。
scene.c 文件主要包含了许多场景的绘制。这些函数的主要功能是设置游戏的各种场景，并在这些场景中放置一些必要的元素（如按钮，图片，NPC，迷宫等）。
下面对代码中的每一个函数进行简单的解释：

1. **DrawInitialSurface()**函数：用于绘制游戏的标题界面，这包括标题图、开始游戏和设置按钮等。
2. **DrawNPCSurface()**函数：用于绘制 NPC 界面，包括绘制 NPC，设置玩家的位置，以及根据玩家的方向绘制玩家的图片。
3. **randomOrManualChoose()**函数：用于绘制一个让用户选择随机生成或者手动编辑地图的界面。
4. **DrawPickmodeSurface()**函数：用于绘制选择游戏难度的界面，这包括困难，正常，简单等三种模式。
5. **draw_scene_edit_maze()**函数：用于绘制编辑地图界面，用户可以在这个界面上对迷宫地图进行修改。
6. **draw_scene_mazegame()**函数：用于绘制迷宫游戏界面，包括背景、石头、树、花、蘑菇等元素以及迷宫本身。
7. **draw_scene_mazegame_with_path(Solution *solution)**函数：绘制迷宫游戏场景并显示解决迷宫的路径。
8. **draw_scene_mazegame_with_next_path(Solution *solution)**函数：绘制迷宫游戏场景并显示解决迷宫的下一步路径。
9. draw_scene_mazegame_with_auto_path(Solution *solution)函数：

在这些函数中，都有一个变量 **scene** 用于记录当前的场景，这对于游戏逻辑的控制是非常重要的。并且这些函数中都使用了一些图形绘制函数如 **draw_bmp()**，这些函数通常是用于在指定位置绘制特定的图片。**CreateImageButton()**函数是用于创建图片按钮的，它接收一些参数如图片的位置和大小以及按钮的 id 等。**CreateTextBox()**函数是用于创建文本框的。

solve.h 文件，主要负责 solve.c 文件中函数的函数声明并且定义关键的数据结构。
solve.c 文件，其主要功能是实现角色的移动，检查碰撞，以及检查游戏是否胜利。以下是主要的函数及其功能：

1. **find_element(Maze *maze, Position position)**: 在迷宫中查找给定位置的元素。如果该位置没有元素，则返回 NULL。
2. **move_player(Maze *maze, int dx, int dy)**: 移动玩家到给定的方向。如果新的位置没有发生碰撞，那么玩家的位置将被更新，并检查是否取得胜利。
3. **move_player2(Maze *maze, int dx, int dy)**: 这个函数用于玩家角色在非游戏场景时在一定范围内移动。
4. **check_collision(Maze *maze, Position newPosition)**: 检查玩家的新位置是否会碰撞。首先检查迷宫元素，然后检查怪物。如果新的位置有碰撞，返回 true；否则返回 false。如果玩家死亡，也返回 true。

5. **check_victory(Maze *maze)**: 检查玩家是否到达迷宫的出口，即是否获得胜利。如果玩家到达迷宫的出口，返回 true；否则返回 false。

接下来是迷宫求解模块，具有搜索最短路径，寻找所有路径，以及提供下一步提示的功能。先定义了两种数据结构：QueueNode 和 Queue。QueueNode 表示队列中的一个节点，包含了节点的位置(Position pos)、距离起始点的距离(int distance)以及拥有的钥匙数(int keys)。Queue 则是用于存储 QueueNode 的队列。

1. **isempty(Queue *queue)**: 检查一个队列是否为空。如果为空则返回 true，否则返回 false。
2. **dequeue(Queue *queue)**: 从队列中删除一个元素并返回。如果队列为空则会退出程序。
3. **free_queue(Queue queue)* 和 **free_distance_matrix(int **matrix, int height)**: 这两个函数用于释放队列和距离矩阵所占用的内存。
4. **create_queue(int capacity)**: 创建一个给定容量的队列，并返回队列的指针。
5. **enqueue(Queue *queue, Position pos, int distance, int keys)**: 将新节点添加到队列中。
6. **create_distance_matrix(int height, int width)**: 创建一个给定大小的距离矩阵，并返回其指针。
7. **is_valid_move(Maze *maze, Position pos, int keys)**: 检查是否可以移动到新位置。如果可以，则返回 true，否则返回 false。
8. **find_shortest_path(Maze *maze)**: 在迷宫中寻找从起始位置到终点的最短路径，并返回路径的信息。
9. **dfs(Maze *maze, Position current, Position end, Position *path, int path_length, Position **all_paths, int *num_paths, int keys, bool **visited)**: 深度优先搜索迷宫中所有的路径。
10. **solve_maze(Maze *maze, SolutionList *solution_list)**: 寻找迷宫中所有的路径，储存路径到链表。
11. **next_step_hint(Maze *maze)**: 给出迷宫的下一步提示。
12. **dfsmaze(Maze *maze, SolutionList *solution_list, Position pos, Position *path, int path_len)**: 对于迷宫进行深度优先搜索，并将路径信息存储在 solution_list 中。

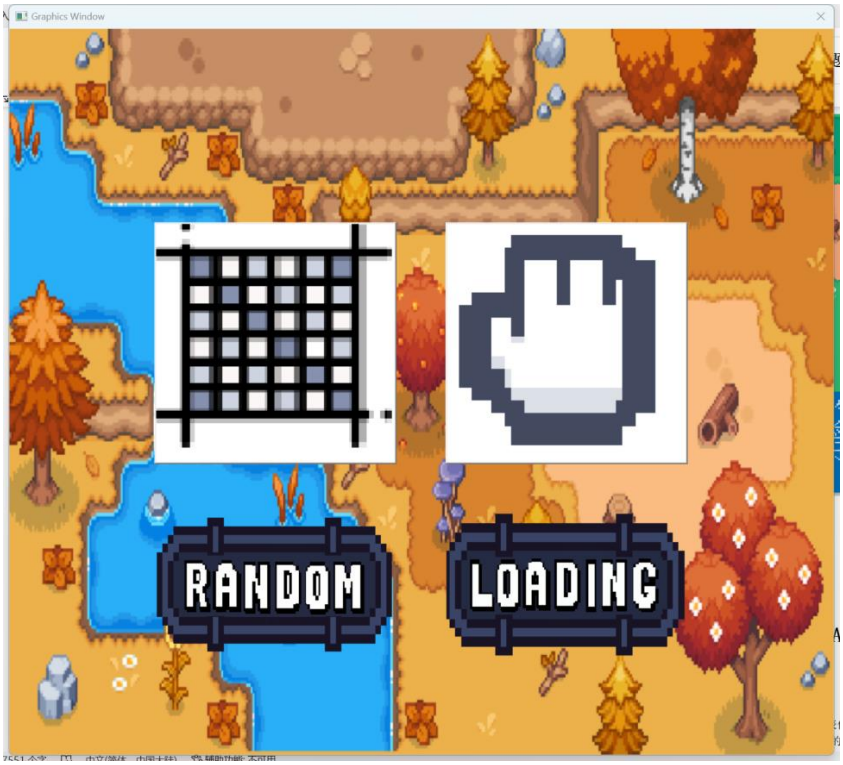
第五章 运行截图

初始界面

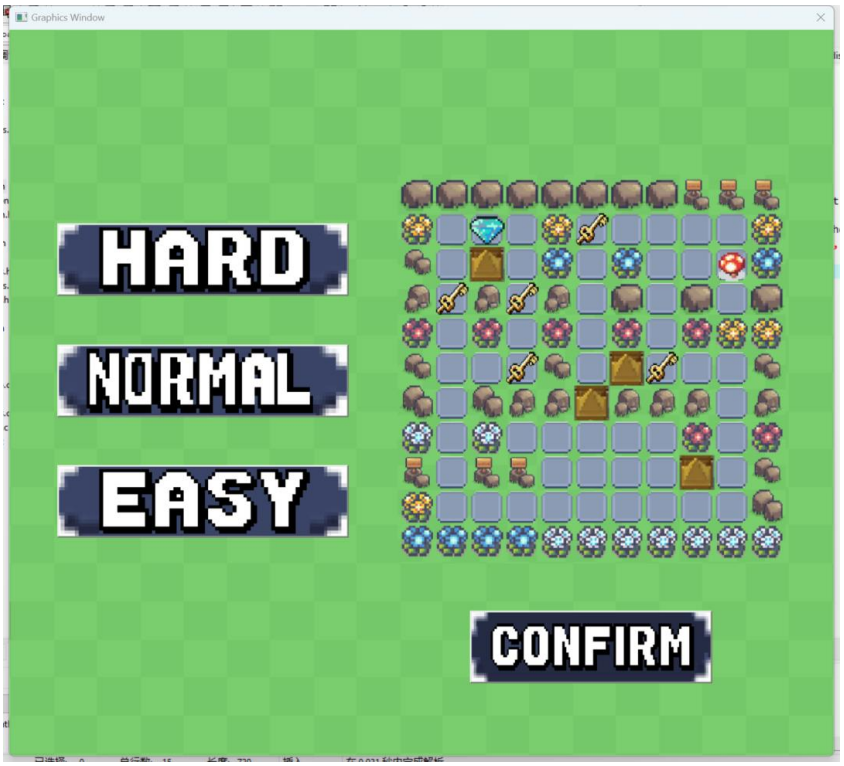
NPC 界面（1）

NPC 界面（2）

选择随机创建或者加载迷宫界面



随机创建—选择难度界面



手动编辑地图界面

手动求解游戏界面

提示最短路径

提示下一步


Graphics Window

输出所有解


Graphics Window
Total number of solutions: 19656
Solution length: 22
Solution length: 22
Solution length: 22
Solution length: 18
Solution length: 24
Solution length: 24
Solution length: 24
Solution length: 20
Solution length: 30
Solution length: 30

程序求解页面

程序求解自动执行和单步执行

菜单界面


Graphics Window
Start
?
Sound
Tell Story
Exit
作者
北海无河
星之坏比
Pastime


Graphics Window
Start
?
Sound
Tell Story
Exit

游戏说明界面

声音开关界面

背景故事界面

彩蛋界面 1

彩蛋界面 2

附录 A：小组成员贡献

组长：卢星余（50%）

1.完成所有后端函数，具体实现了迷宫的创建、怪物的生成、碰撞检测、地图的编辑、迷宫的保存与加载、程序自动求解与手动求解等等功能，使用 BFS 和 DFS 算法实现了迷宫的求解。

2.对图形库进行了优化，实现了自定义贴图，自定义创建贴图按钮以及相关响应架构，自定义创建文本交互框等功能。

3.完成手动编辑地图界面并实现功能。

4.撰写实验报告梳理文件架构部分。

组员 1：黄晨（25%）

1.绘制初始界面并接入开始游戏按钮。

2.绘制 NPC 界面并实现与 NPC 交互功能。

3.绘制迷宫难度选择界面并与实现难度选择功能。

4.绘制菜单界面，实现在不同场景下的继续功能，查看游戏说明及背景故事功能，bgm 音乐的开关控制功能以及退出游戏功能。

5.绘制游戏最终结算的胜利界面与失败界面。

组员 2：葛大勇(25%)

1.绘制迷宫随机生成与加载地图界面并实现功能。

2.绘制操作说明与游戏故事界面。

3.绘制游戏迷宫场景界面以及部分子界面并接入迷宫求解功能

4.调整改进部分界面 UI 设计与贴图。

5.撰写实验报告。

附录 B：全代码

代码量过大，故单独打包

Github 已开源，也可访问 https://github.com/LUXINGYU23/Magic-Maze 获取代码