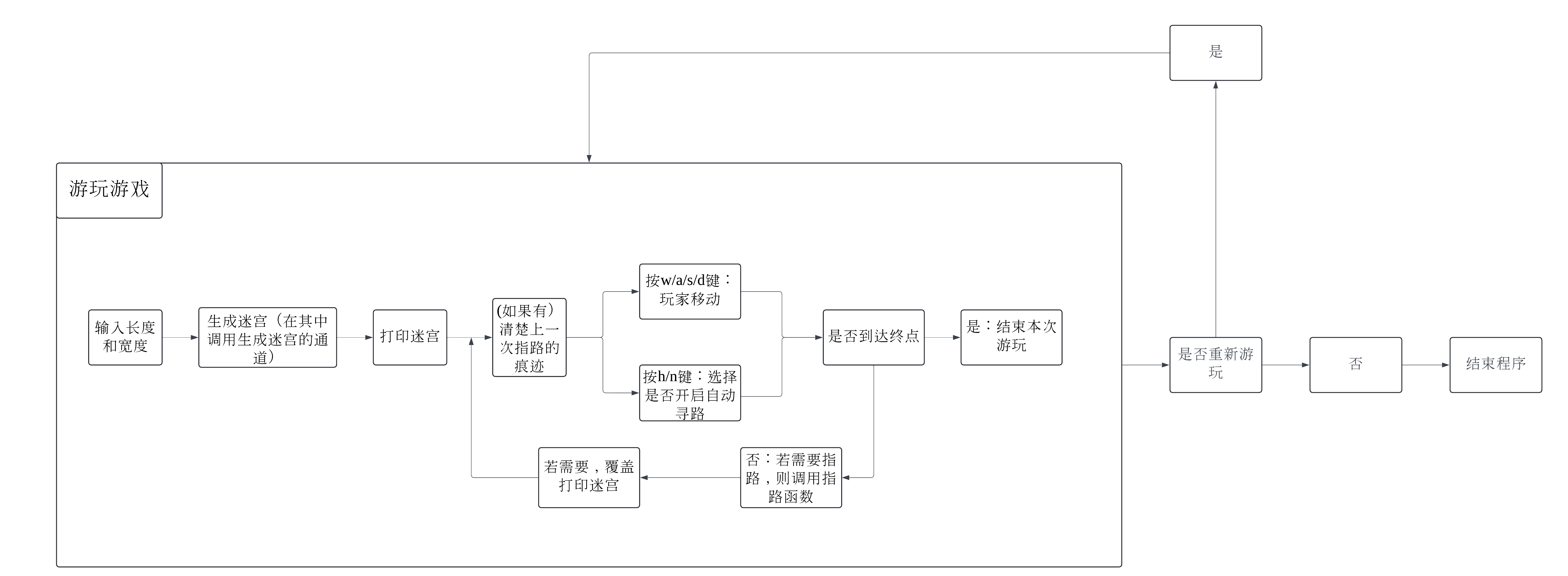
**实验报告：Maze Game**

钟梦阳 2023013202

**A 主要功能流程**

**B 辅助函数介绍**

此处介绍实现主要功能时需要的辅助函数

**一 print\_based\_on\_int** 第10行开始

构建maze数组与迷宫各组成部分的映射。使用switch语句根据迷宫二维数组中每个元素的值打印出对应的字符。

值为0: "█" //墙壁

值为1: " " //道路

值为2: "S" //起点

值为3: "E" //终点

值为4: "▲" //玩家所在位置

值为5: "×" //指路标记

**二 random\_int** 第36行开始

随机生成介于两个整数之间的整数。

**三 in\_bound** 第41行开始

判断某点是否在迷宫的边界内。

**四 generate\_random\_directions** 第46行开始

首先利用random\_int生成0、1、2、3的随机排列，然后以此为序重排方向数组，从而生成随机的方向数组并返回，方向有上下左右四种。

**五 reminder** 第224行开始

在move函数（见下）中，玩家试图移动至墙壁或边界时，调用reminder函数给出提示。提示显示0.9秒后自动擦除。

**C 主要功能介绍**

**一 输入长度或宽度**

**input\_length\_or\_width** 第146行开始

让玩家输入迷宫的长度或宽度，并验证输入的是否是数且大于0、小于等于50。若不合条件，给出相应提示。

**二 生成迷宫的通道**

**DFS\_generate\_passway** 第70行开始

**prim\_generate\_passway** 第91行开始

游玩时玩家从两者选择一个来生成迷宫。

DFS算法：

从起点开始，将其标为通道。

在每个节点，用generate\_random\_directions函数生成一个随机的方向数组，将当前位置的迷宫单元标为通道，然后使用 for循环遍历四个方向。

对每个方向，有一个间隔为1的相邻单元格(mx, my)和一个间隔为2的相邻单元格(nx, ny)。利用in\_bound函数，检查 (nx, ny)是否在迷宫内且未被访问过。若如此，则将(mx, my)标为通道，递归调用DFS\_generate\_passway，从新的位置(nx, ny)继续生成通道。

这样的递归调用使得通道沿着随机方向不断深入，直到所有可行通道都被访问并标记，迷宫通道得到生成。

prim算法：

将起始位置标为通路。创建存储前沿坐标的数组以记录生成迷宫时已探明的可挖单元格，并记录其中元素的个数frontier\_num。将起始位置加入前沿数组。设frontier\_num为1。

在前沿数组非空的情况下循环执行：

随机选择一个前沿单元格。检查该单元格的四个相邻格是否有未挖掘的点并记录之。

若存在未标记单元格，则从中随机选取一个，从前沿格朝对应方向挖两格，到达新延伸点，将其加入前沿数组，frontier\_num加一。若不存在未标记单元格，则将其从前沿数组移除，frontier\_num减一。

循环上述步骤直到前沿数组为空，此时所有可行通道都被访问并标记，迷宫通道得到生成。

**三 生成迷宫**

**generate\_maze** 第160行开始

先通过双重循环，将迷宫数组中的所有元素全部初始化为墙壁。然后设最左上角为起点，根据用户的选择使用DFS或prim算法挖掘迷宫通道（按d、g以外的键不会进入下一步）。在已经挖出的通道中随机指定终点（不与起点重合），返回终点坐标，最后把起点位置返回为玩家所在位置。

**四 打印迷宫**

**print\_maze** 第201行开始

先打印迷宫布局与操作方法。再用双重循环将迷宫打印到屏幕上（利用print\_based\_on\_int函数）。

**五 覆盖打印迷宫**

**reprint\_maze** 第214行开始

清除屏幕，再用print\_maze函数重新打印迷宫。

*此处保留了清屏不闪烁的废稿：*

*//printf("\033[%d;%dH",0,0);\ 将光标移到最左上角*

*fflush(stdout); 使上一条命令立即运行*

*当已打印内容完全显示在控制台上时没有问题。但当已打印内容较多，部分上端内容没有显示出来时，光标会移到已显示内容的左上角而非所有内容的左上角，未显示内容不会被擦除，导致“重复打印”。*

**六 玩家移动**

**move** 第236行开始

传入玩家当前坐标player\_x、player\_y和玩家在x、y方向的坐标增量dx和dy。计算玩家移动后的新坐标new\_x和new\_y。使用switch语句根据maze[new\_x][new\_y]的值判断玩家新位置的状态：

值为0：表示新位置是墙壁或边界，调用reminder函数提醒玩家。

值为1、2、3：表示新位置为通道、起点或终点。此时将新坐标记为玩家当前位置并返回。如果玩家上一步在起点（坐标为 (1,1)），则将其变回起点，否则变回通道。

**七 自动寻路**

**find\_way** 第259行开始

本函数采用BFS（广度优先搜索）算法。

传入终点坐标end\_x、end\_y与玩家坐标player\_x、player\_y。

创建数组last\_step\_x和last\_step\_y记录找路走到此处时上一步在何处，用来确定路径。创建数组 queue\_x 和 queue\_y 按序存放找路路径。创建二维数组visited记录通道是否被访问过，防止重复访问。

记front为已被探索的点的数量，初始化为1；check 为周边的点也被探索过的点的个数，初始化为0；found 标记是否找到终点，初始化为0。将玩家当前位置加入queue数组，并标记为已访问。

在check<front且found为0的条件下（表示各通道未完全探索且未找到终点）循环执行：

从queue队列中取出当前点坐标，check加一（表示周边被检查过的点多了一个）。对每个方向，计算邻近点 nx 和 ny。利用in\_bound函数，检查邻近点是否在迷宫范围内，为通路，且未被访问过。

若邻近点满足条件，将其加入queue队列，标记为已访问，front加一。将当前点坐标记为邻近点（nx、ny）的上一步的坐标last\_step\_x[nx][ny]、last\_step\_y[nx][ny]。若邻近点同时为终点，设found为1并退出循环。

然后从终点开始，根据last\_step\_x和last\_step\_y 回溯路径，直到回到玩家当前位置，此时路径确定。返回下一步应走的位置where\_to\_go\_x、where\_to\_go\_y。

如果下一步不是终点，则在迷宫中标出指路标记。

**八 游玩游戏**

**play\_game** 第326行开始

1 初始化

用srand((unsigned)time(NULL))初始化随机数生成器。

打印欢迎信息，提示玩家开始游戏。

设整数need\_help以标记是否指路，初始化为0。

2 获取迷宫尺寸

在while循环中，利用input\_length\_or\_width函数让玩家输入迷宫长度和宽度。输入值为小于等于50的正整数时结束循环。

3 生成迷宫

使用generate\_maze函数生成迷宫，设置终点位置和玩家起始位置。

清屏并调用print\_maze函数打印迷宫。

4 定义方向和按键映射

定义dx、dy数组表示上下左右四个方向的坐标位移值。

定义direction\_keys\_1和direction\_keys\_2数组分别表示小写和大写的w,s,a,d键。

5 接收玩家操作的while循环

以下内容为循环内容

如果存在上一次的指路痕迹，将其重置为通道。更新迷宫的起点、终点和玩家当前位置。

设整数need\_reprint 以标记是否重新打印迷宫，初始化为0。

用 getch() 获取玩家输入。

若玩家按下w，a，s或d键，调用 move 函数进行移动。

若玩家按下h键，设need\_help为1，请求帮助。

若玩家按下n键，设need\_help为0，取消帮助。

对以上三种情况，设need\_reprint为1，按其他按键need\_reprint仍为0。

如果玩家当前位置与终点位置相同，用reprint\_maze函数重印迷宫并显示过关信息，跳出循环结束游戏。

如果玩家请求帮助，调用 find\_way 函数寻找路径，并设置下一步位置。

根据need\_reprint的值决定是否重印迷宫。

**九 主函数**

**main** 第409行开始

声明字符变量want\_to\_play，以存储玩家是否想要重玩游戏的选择。

使用do-while循环控制游戏过程，确保游戏至少运行一次。每次循环先清除控制台上的内容，然后调用play\_game函数进行一次游戏。游戏结束后，询问玩家是否想要重新游玩：若玩家按y键，则重新开始游戏，否则结束游戏，退出程序。