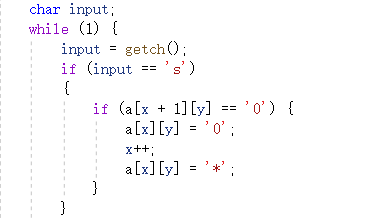
第一步：迷宫游戏的核心（雏形）

迷宫首先由墙和路组成。可以通过字符串（二维字符串数组）对迷宫地图进行存储。

迷宫的边界都设为墙，防止出现bug。

以0表示路，以1表示墙。同时用一个辨识度较高的符号表示老鼠本身（比如“\*”）。

使用getch（）函数对输入流进行判断，来实现通过键盘对老鼠的操纵。

如右图显示，当输入s时，先进行判断，如果下面一格不是墙，就能对老鼠执行向下操作。

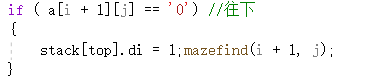
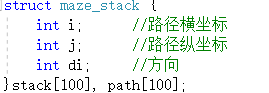
（与此同时，需要注意的是：向下操作是先将下面一格变成“\*”（老鼠），再将原来位置变成路）

其他三个方向也同理。

第二步：增加辨识度

用“■”表示墙，用“ ”表示路，以代替原本的“0”，“1”。

这仅仅需要在输出数组前先进行一层判断即可，但却对迷宫的美观性有着相当不错的提升。

第三步：增加寻路模块

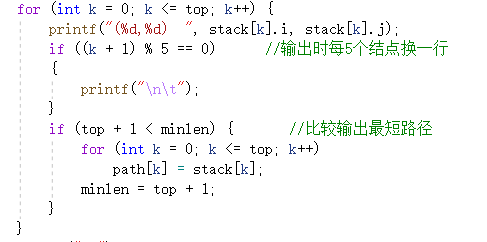
首先设立一个数组对走过的路进行存储，i，j存储坐标，di存储方向。（以方便后面的显示）

补充：其实更好的数据结构应当使用栈，但是考虑到需要对路径进行输出，而单纯的栈无法做到从栈底进行输出。因此最终采用的还是数组的数据结构（但是使用了top进行对栈的模拟，因为我们需要更改的永远是栈顶，所以进栈即top++，出栈即top--）

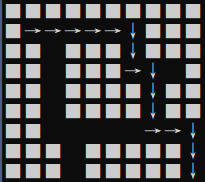
寻路主要采用的是递归的方法，如最左的图所示。在进入一层递归前，先执行进栈，将该点标记为走过，并对相关的值进行赋值操作。然后依次进行右下左上的判断。如果可以走通，记录下方向后就进入到下一层递归中。

递归可以保证走的路线永远不会重复，而标记的方式可以避免走“回头路的现象”。在此种方式下，必然可以走过迷宫的所有路线。

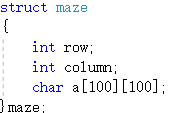
此外，在“尝试”走过了四个方向后，将该点的位置标记为未走过，并将其出栈，此时即可返回上一层。由于上一层已经执行过该方向的尝试，下一步必然会从另一个方向走（“也有可能是继续执行消除标记，出栈，返回上一层”）。在该种方法的进行下，必然可以保证，迷宫中的所有路径都被遍历。

 如果走到了终点，就对数组进行输出。同时比较数组长度，如果更小，就写入最短路径。

递归终止的条件即为栈内没有元素。此时，终止递归，并输出最短路径。

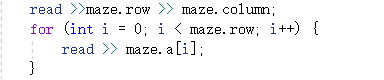
 最后还可对迷宫进行打印，运用之前记录的方向信息以及箭头符号，可以直观的输出最短路径（而非其他路径的坐标式输出）

（部分路线如右图）→

第四步：对文件的处理的操作

迷宫文件的读入主要是使用了ifstream和ofstream函数实现对文件的操作。

在此之前我先对迷宫做了一个结构。加入了迷宫的行数和列数后，在对迷宫的输出，对到达终点的判断（迷宫右下角）上有了更为完善的方法。

（在此之前我最先想使用的是更为标准的文件操作，如格式化写入等，这样可以很方便的读入二维数组，但是发现格式化写入，写入的是乱码，同时，规整的数据反而会读出有误的数值。最后放弃了这种较为方便的方式）

但是因为在文件中写入了行数和列数，可以很方便的读入一维数组，并将其传到二维数组中。这在一定程度上解决了该问题。

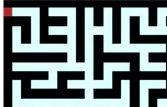
有了文件的读取与写入后，为了可以在程序开始时选取已经写好的迷宫。尝试了各种在ifstream的函数中即右图中的nm处（这里需要的变量是文件的相对路径，需要在双引号的包裹下使用）。尝试了各种转义字符，无效后。在网上找到了sprintf函数。即变量的部分可采用自己输入的数据。在此种方式下才得以实现对迷宫文件的选择。（写入时也同理）。

第五步 迷宫的随机生成

由于并不是所有的guanka都有对应的迷宫文件，因此在读取到不存在的文件时，我希望可以根据用户输入的行数和列数随机生成一个迷宫。

在深度优先、递归分割等随机算法我最后还是选择了prim随机生成算法

（已是在对随机算法的查找过程中，我了解到迷宫其实有两种普遍的表达方式）

一种为左图所示的墙类型。

一种为右图所示的网格类型。

这两种在细节上都各有出入，但大体

上还是一致的。

首先把所有的网格都视为墙；也就是二维数组所有元素设为1；

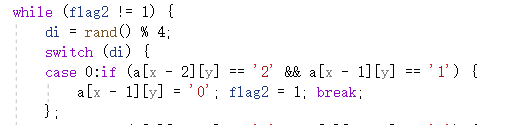
在原本是路的位置中随机选择一个网格，也就是在描述迷宫的二维数组中，选择行、列索引都是奇数的。例如[3,3]，[5,9]之类的。将这个网格变为路，也就是二维数组对应的数字变为0。然后将这个网格周围的原本是路的网格添加到候选网格列表中去，将这个网格对应的数字变成2。（现在这些奇数网格有三种状态1.未经过任何操作（对应数字仍为1）2.候选网格（对应数字为0）3.已访问网格（对应数字为2，已经变成路了））。所谓周围就是之间隔一堵墙的网格。也就是网格的行或者列索引差2。

当候选网格列表不为空时，进行下面的循环：

1. 在候选网格中随机选中一个网格。
2. 查找该网格周围的已被访问的网格，随机选中一个打通与其之间的墙。（注意到候选网格周围一定会至少有一个被已访问的网格，否则该网格不可能进入候选状态）。
3. 将该网格周围对应数字为1的网格（未被操作的奇数网格）纳入候选网格。

在此种随机算法下生成的网格绝对不会形成回路，且每一处路必定都可以走到（变相的生成的是一棵数）其实我感觉这和最小生成的prim算法相比，不过是将其选最短的一条路，变成了随机选一条边。其余基本一致，所以这也是该算法也叫prim的原因吧。

细节详述：

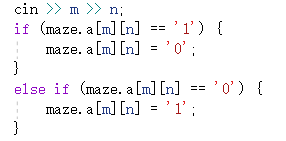
为了实现随机选取，这里采用了rand（）函数，种子是最普遍的当前时间。

在随机打通周围已访问网格上，采取了更为粗暴的无限循环。右图是其部分。它会一致循环下去，直到有墙被打通，flag2=1，退出循环。

而在选取候选网格时，则采用了rand（）%（候选网格个数），来选取对应数字的候选网格。在这之后再将该网格踢出存储结构，个数-1。同时候选网格个数为0时，也将结束随机生成（随机生成完成了）

注意，迷宫大小必须行列数为单数，且大于等于5（1和3。。。本来就没有迷宫）

第六步 迷宫编辑的优化

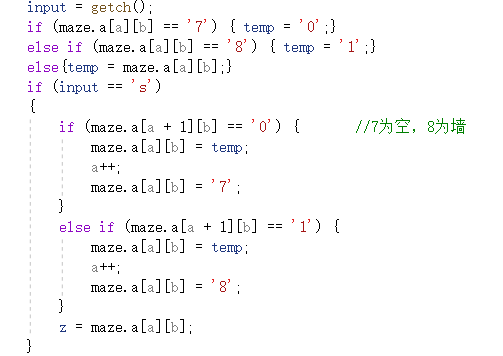
迷宫的编辑实际上是很简单的。毕竟迷宫只有两种形态，要么把1的地方变成0，要么把0的地方变成1。所以要实现要很简单，就像左图一样，通过输入坐标来修改对应坐标的0、1值就可。但是简单的方法往往不一定便于操作。先不说输入坐标修改效率低下。而且一个个的数迷宫坐标也十分恼人。

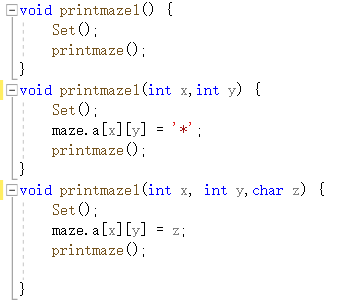
于是便有了控制台下我认为最好的办法：

像走迷宫一样，用键盘控制另一个可以“跑到墙上”的光标，在迷宫中移动，如果想要修改，就移动到那个位置，并按下空格。

这个方法下我希望可以在墙上和在路上有不同的显示，就像这样，路显示空心圆，墙显示实心圆。代码其实与迷宫移动模块基本一样，只不过多了一种情况罢了。

这种方法带来的一个问题就是，你的原本位置不像迷宫移动那样，必定是路。你还额外需要一个temp的中间变量，来记录你原本位置是路还是墙，从而，在你移往下一个位置时，通过记录的temp将原来的位置正确还原，并记录下新的temp。（当然temp需要一定的转化 ）（同时，如果修改老鼠所在位置的路会产生一些很困扰的问题，但在我写的如下代码中，编辑光标是移不到老鼠的所在位置的。）

在迷宫操作的时候按下“x”进入编辑模式，我打算在编辑模式中加入一个可以随时保存迷宫的方法。比如说按下“y”进行保存。保存时自然要将“老鼠”，“编辑光标”等不适0,1的元素转化为0,1后才能保存。因此，为了保存后，光标、玩家还在当前位置，就需要对这些额外数据采用中间变量存储，比如代码中的变量z。

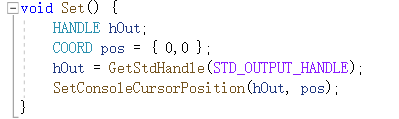


（同时在保存完后的打印迷宫中，还需要对这些变量进行额外打印，不然打印的只会是一个空白的迷宫，虽然按下任意方向键后，就会又出现，但始终会让用户产生困扰。）于是针对打印迷宫又做了如下细化。这些分别是普通的打印迷宫，额外打印老鼠位置，额外打印编辑光标的。

额外的一些小细节：

1. 闪屏问题：

迷宫游戏当然不能一遍一遍的不断在下面打印迷宫啦，所以我一开始使用的是

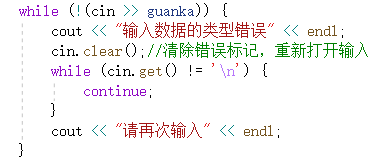
system（“cls”）指令。该指令会将控制台的屏幕清屏。迷宫较小的时候由于打印比较快，所以还没什么问题，但当迷宫较大的时候，如此就会产生屏闪现象。对于用户的眼睛体验实在不太友好。于是这里改用了重置输入位置的方式，将每次的输入位置移动到控制台的左上角。如此清屏->重头输入变便会改完从头开始覆盖。这种方法消除了清屏现象的产生。

右图的COORD pos = { 0,0 };设置控制台的光标位置为（0,0）；GetStdHandle(STD\_OUTPUT\_HANDLE)得到标准输出句柄；

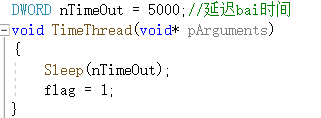
SetConsoleCursorPosition(hOut, pos);将光标移动到初始位置。

当然这样的覆盖有时会导致字体的遮盖等等不良影响，因此在某些位置，还需与system（“cls”）混用。

1. 输入格式问题

考虑到有时可能因为输入问题导致程序异常报错，针对输入做了一系列的规格化。

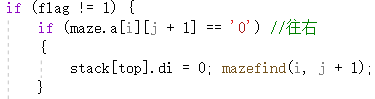
右边即为代码实现，他会对输入的内容进行判断，并不断循环，直到你输入正确的数据为止。

1. 寻路问题

由于有时路线过多，光搜寻并输出所有路径便会耗去大量时间。所以在寻路模块的基础上，加上了时间控制，一旦到指定时间，便会自动终止寻路。

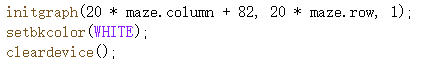
具体代码实现是使用了<process.h>头文件下的\_beginthread方法。

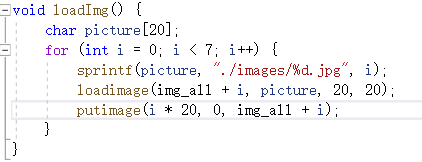




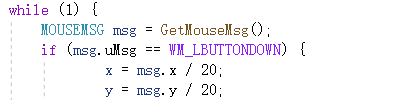
图形界面下的一些改动：

由于学习成本的原因，在短时间内我只能使用esayx（一款基础的图形绘制库）

1.首先是使用initgraph制作画板（长宽根据迷宫的大小动态变化，右侧多出来的区域做成按钮栏）

同时设置背景颜色，将屏幕清空已显示出背景颜色。

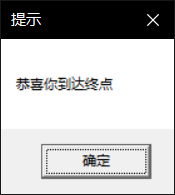
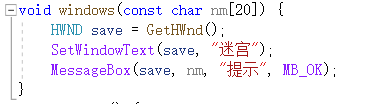
同时使用loadimage函数导入所需要的图片，并在之后的迷宫打印函数中，用putimage替代原本的符号。成果如右图显示。（部分迷宫）

2.在实现图形显示后，我的第一想法是实现对迷宫编辑的优化。

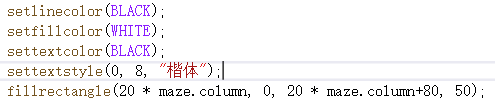
如右图所示使用GetMouseMsg获得鼠标的操作权柄，当检测到鼠标左键点击时，进行判断点击的是哪一个格子，并对原数组的该位置的数值进行修改，在这之后输出迷宫，以达到使用鼠标修改迷宫的目标。

补充：不知道为什么，在编辑迷宫中使用单纯的system（“cls”）会有延迟，要过好几秒才会显示更改。到是使用上述的重置控制台光标的方法后，延迟却消除了。没有搞清楚这里面的原理）

3.同时，通过弹出窗口，替代了之前的控制台信息的输出，让用户界面更加友好。窗口这里不能单纯输入字符串，通过一番尝试后，改为const char型，才得以实现窗口函数的普适性。

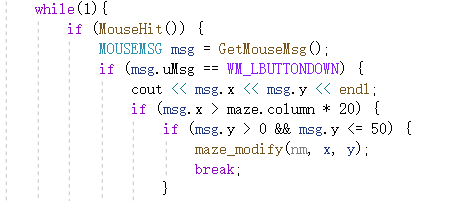
如右图所示

（

4.既然做了图形化界面，许多输入就不一样要靠键盘了。可以通过鼠标点击按钮的方式来实现。

Easyx有一个很不方便的问题就是——它仅仅是一个简陋的图形绘制库，而非标准的gui界面。所以

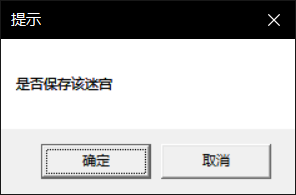
按钮之类的实现会较为繁琐。

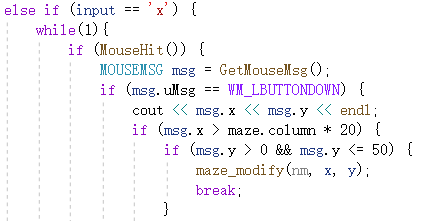
具体过程如图所示，需要先设置图形的边框，底色（按钮的形状），接下来设置文字的样式，颜色，再分别创建两者，先图形再文字，使文字浮于图形之上。

最后还需设置鼠标点击相关位置的触发，才能实现一个按钮的功能。

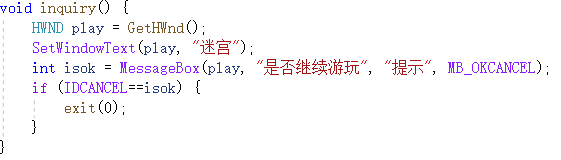
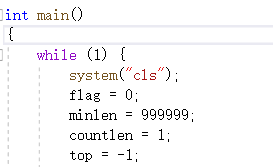
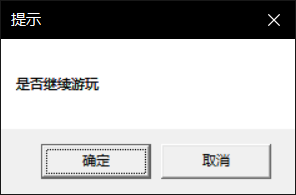
最终成果如下图所示。



当你进入编辑状态时，还会额外显示一个退出编辑的按钮，点击该按钮即可退出编辑。当然，在退出时他也会弹窗提示是否保存。（如右图）

按钮的存在还有一个十分严重的问题。由于我的移动还是靠键盘操作，所以在原本的程序中我同时设置了对键盘输入的读取和鼠标点击的读取，但在实际的调试中发现，所有的点击全部失效，程序在获取我键盘输入时无法获取鼠标的输入。无奈之下我只好设置了按下“x”之后，停止对键盘输入的读取，转而专门对鼠标的读取，总算是实现了按钮的功能，只不过在每按下一个按钮时需要额外按下“x”后才行。（当然，在迷宫编辑中只有对鼠标的读取，所以此时无需按下“x”）代码实现如上图所示。（也许有更好的办法，但是我只想到了这些）

5.最后，在老师的建议下，我还对主程序添加了循环，并重置了大部分的全局变量。

如右图所示，并在程序的结尾添加了这个弹窗。已实现不断选择关卡，无需每选一关都要重新运行的麻烦。当点击取消时便会退出程序。