project报告

成员：冯钰茹12112243，王璐12113041，付翼铭12111604

## 整体结构

共采用9个类进行数据的存贮与运行。该9个类分别为：

#### BlockType

枚举类，用于记录方块5个不同种类，分别为SQUARE(2\*2)， HORIZONTAL(1\*2), VERTICAL(2\*1), SINGLE(1\*1), BLANK(空格)。

#### Blockfiled

用于记录每种block的一些基本属性。

成员变量：方块类型blockType(变量类型为BlockType)，方块对应的数字number（变量类型为int) (若方块类型为SINGLE，则该number是该方块上的数字；若方块类型为SQUARE，则该number是方块左上角的数字；若方块类型为HORIZIONTAL，则该number是方块左边的数字；若方块类型为VERTICAL，则该number是方块第一行上的数字；若方块类型为BLANK，则该number = 0），方块的宽度(width)（变量类型为int) ，方块的长度(height)（变量类型为int) 。

构造器：传入参数为方块类型和方块对应的数字。在其中根据不同的方块类型采用switch结构初始化变量。

#### Block

用于记录方块的全部属性：基本属性、坐标、哈希值

成员变量：xPos(x坐标)，yPos(y坐标)，hashString, hasCode等

两个参数不同的构造器，一个用于传入Blockfield和x、y坐标，一个用于传入Block和x、y的坐标变化得到新的block。

哈希值的计算方式：将方块类型转化为字符串类型，采用StringBuilder方法将x、y、number以字符串形式加入，得到hashString

#### Board

用于记录棋盘的全部属性：棋盘宽度、高度，一个int型的二维数组matrix存储棋盘元素，一个Block数组存贮棋盘上的方块，棋盘的hashString和hashCode等。

一个布尔型参数isNewBoard判断新得到的棋盘是否为新的棋盘（前面步骤没有出现过的），棋盘每一步的变化步骤用一个字符串everyStep记录，一个一维int数组存储棋盘上的方块数字，一个一维string数组存储棋盘上的方块类型等。

两个不同参数的构造器：一个用于传入二维int数组matrix得到新的棋盘，一个用于传入blocks数组和棋盘的长宽构造新的棋盘。

getType方法：根据传入的x、y值返回当前棋盘该位置上的方块类型。

isblockInBoundary方法：根据传入的方块和x、y坐标判断该方块是否合理地存在在棋盘上。

isValidMove方法：根据传入的方块Block和移动类型判断是否为合法移动。

hashString方法：存棋盘的哈希值，将棋盘上所有block的hashString采用StringBuilder方法加在一起得到棋盘的hashString。

#### Game

用于创造游戏的调用对象。

成员变量：initialBoard，一个二维的动态int数组记录所有的路径

一个传入初始棋盘状态的构造器，一个main方法中写入Frame对象运行游戏。

#### GameSolver

展现游戏的核心解决算法。

成员变量：Game类的game对象，一个String和Board类型的HashMap，一个静态的int型变量sp记录最短路径的长度。

构造器：传入game用于初始化，在其中调用bfs方法进行可用棋盘的搜索。

内部类node：Board类型的currentBoard、父节点指针（node parent)，一个int型的变量step存放从起点开始到当前棋盘的步数。

solve方法：采用广搜解决问题。

getNextMatrix方法：根据传入的棋盘，需要移动的方块，移动类型得到新的matrix

Next方法：传入当前棋盘，需要变化的block，以及棋盘的一些相关参数。得到当前块所有可行的下一步棋盘。

endBoard方法：根据初始棋盘，将初始棋盘的matrix对应的方块上的数字放入一个动态数组中，调用排序得到升序的matrix，将其传入棋盘类的构造器用于生成最终的理想棋盘。

* bfs方法：

根据当前传入的棋盘currentBoard寻找可行解。

其中队列q用于在合适的时候放入和取出结点。动态数组distance用于记录最短路径中的matrix，栈path用于记录最短路径上的结点

采用一般的广度优先算法思路。

每一次在循环中取出当前结点存储的棋盘状态与endBoard得到的棋盘最终状态比较。

若当前棋盘无解，则sp = -1，输出No表示无解

若当前棋盘有解，输出Yes表示有解，用一个循环将path栈中存储的步骤取出，得到从起点开始到终点的最短路径上的每个结点，将每个节点中的棋盘状态放入distace动态数组用于后期输出操作。

#### Frame

构造一个可视化界面，放入两个按钮，一个点击后可获取下一步行棋，一个用于传入棋盘的初始txt文件。在这个类中初始化传入的棋盘，将其显示在界面上。

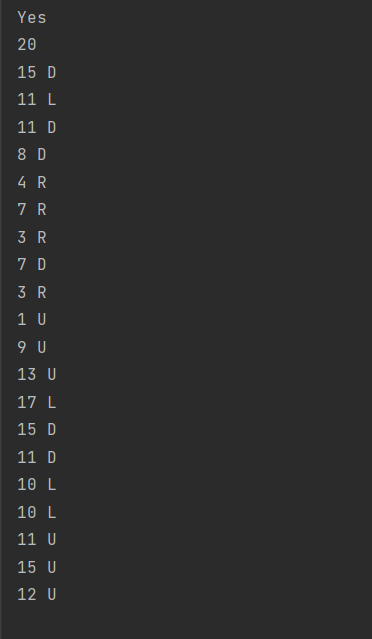
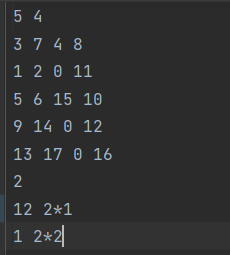
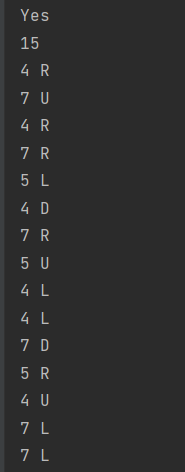
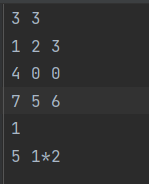
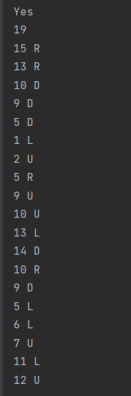
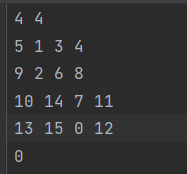
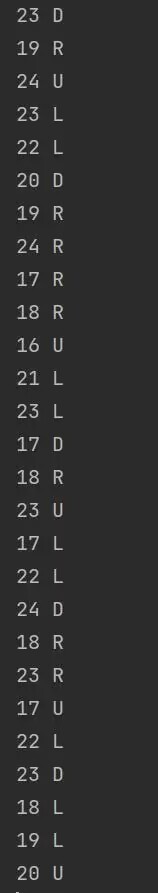
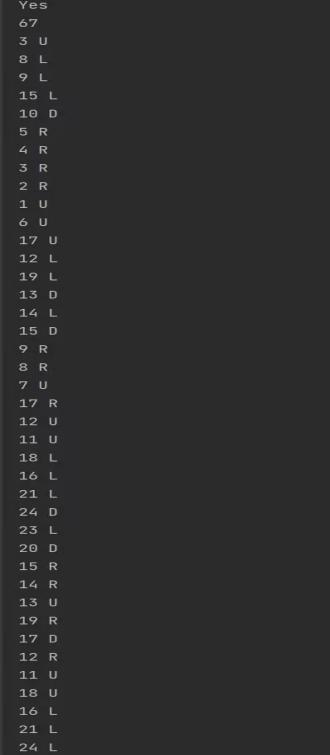
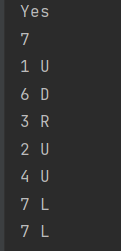
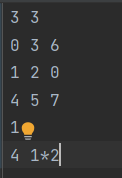
#### DrawPanel

一个继承了JPanel的类，用一个contains方法判断该方块是否为单个方块。若不是单个方块，则根据方块的类型在paint方法中将“绑”在一起的用红线绕一周显示出来。

## 算法优化

1. 利用hashMap，用hashString存储已经存在过的棋盘，从而减少搜索。同时，hashString的构造方式有效避免了冲突。
2. 广搜过程中加入一些限制。利用差值加入限制函数，固定前两行，前三行等方法，限制搜索的节点，提高搜索效率。

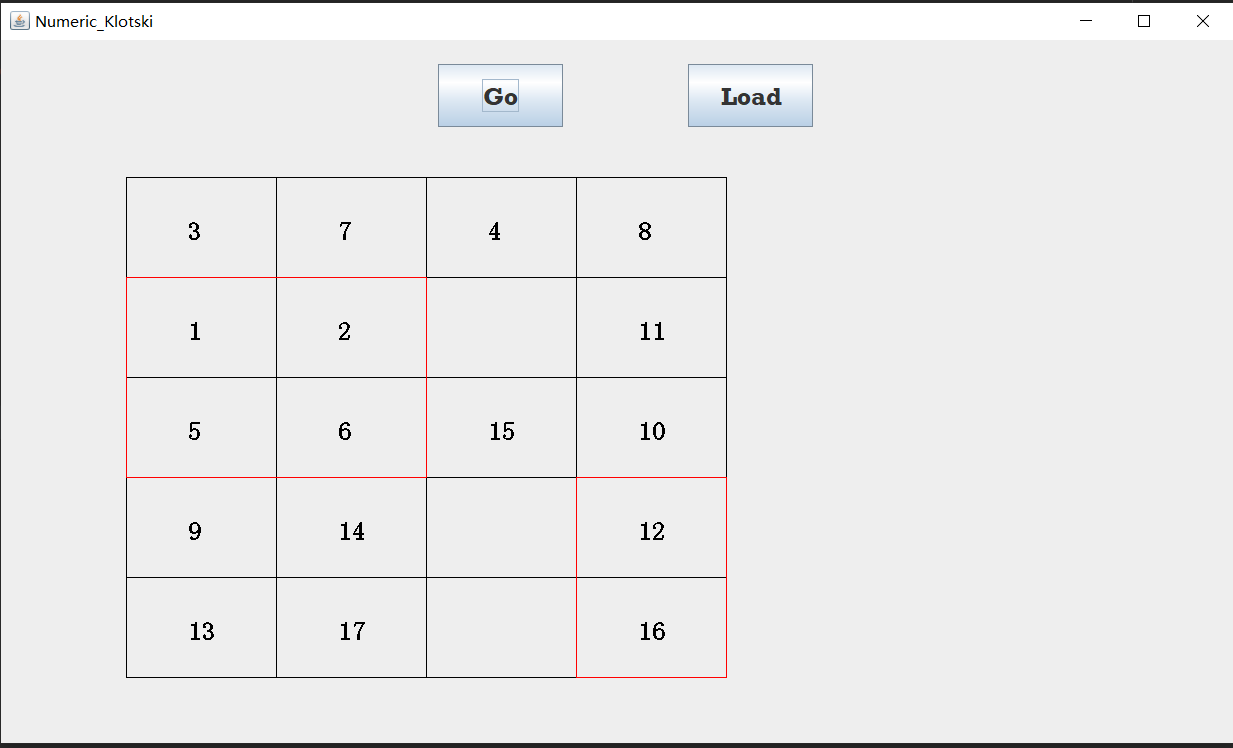
## 数据测试

1. 
2. 
3. 
4. 
5. 

## 测试方法

Game类中开始运行，弹出窗口，点击load，输入文件路径（例：.\input\sample），点击确定后开始运行，命令行中输成no或者输出yes，步数及每一步走法，点击go按钮。则可以在界面上进行移动。移动结束后，弹出窗口。

基本界面如下：



## 算法比较

1. 写了深搜算法与广搜算法进行比较，深搜算法在解决3\*3问题是步数已达到500步左右。同时，在写深搜算法是发现，由于棋盘状态指数级增长，若使用递归方法实现深搜，会造成栈溢出，最后采用了非递归方式实现深搜。
2. 深搜和广搜结合，当广搜遍历到一定程度时采用深搜避免内存爆炸。（不同于在广搜中采用限制结点的方式筛选）