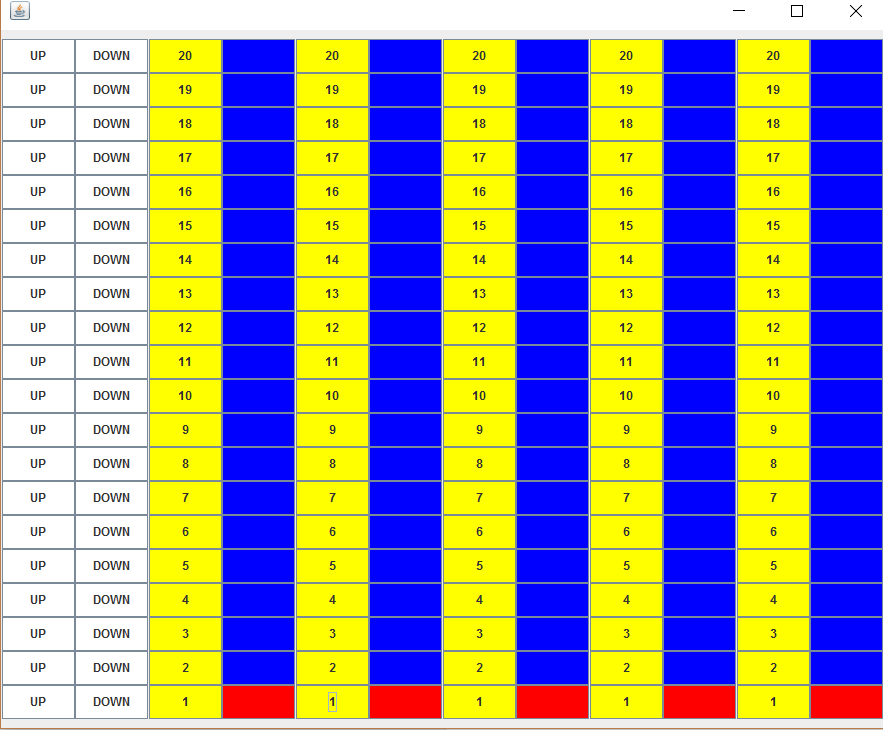
**设计方案的描述报告**

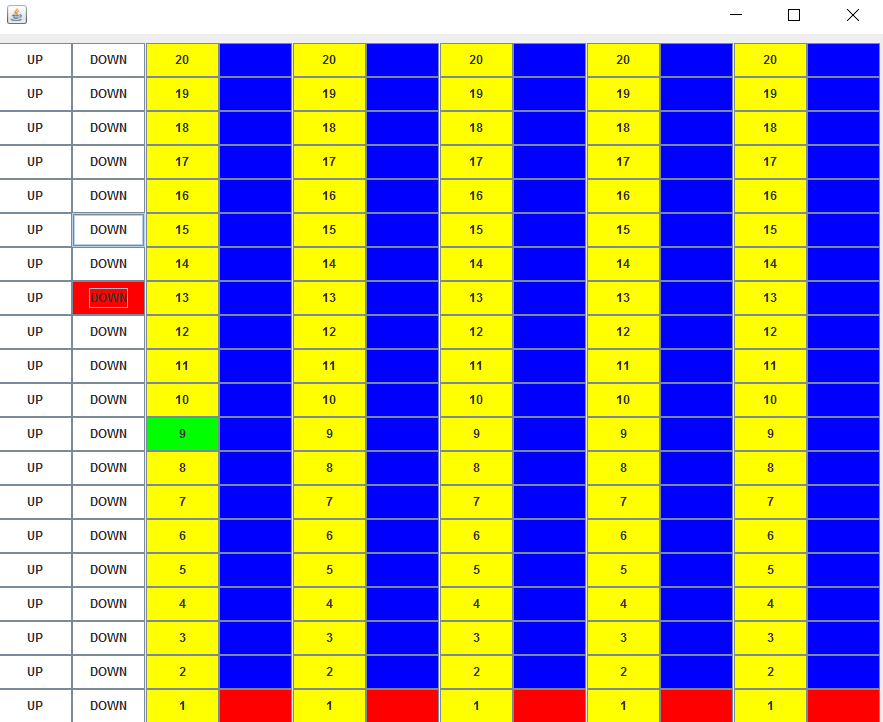
在我的项目中，实现了20层楼，五部电梯的电梯调度功能。

**一．图形界面如下：**



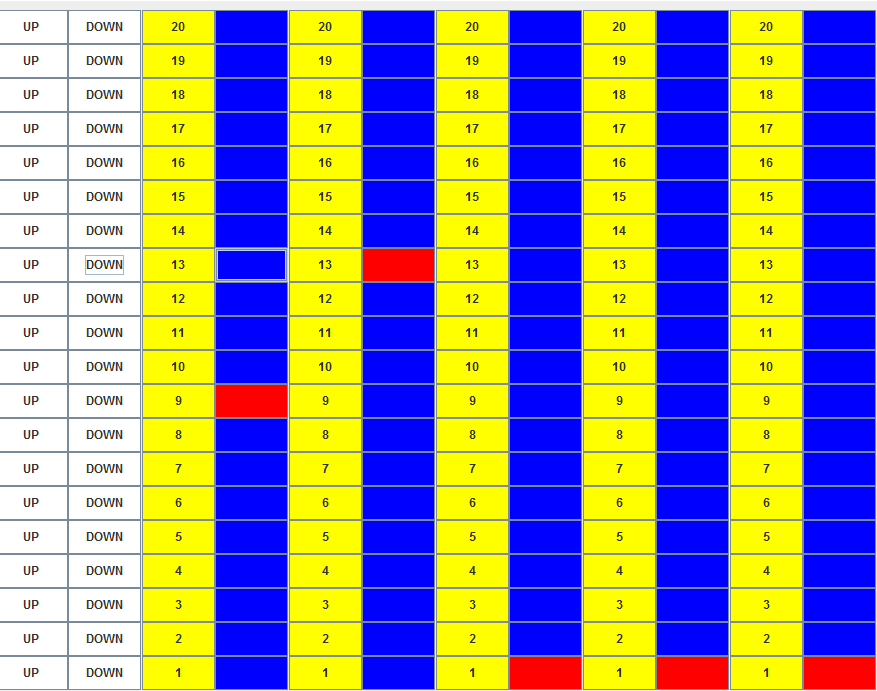
左边两侧白色的按钮分别代表每个楼层外部的上下按钮，UP代表向上，DOWN代表向下。右边五个（20，2）的格子代表五个电梯，每个电梯内，左手边黄色的标有数字的按钮为该电梯内部的按钮（前往1—20楼层），右手边蓝色的代表电梯运行的空间，红色色块代表当前电梯所在位置。

点击按钮触发电梯的运行，例如：



点击13层的down,和一号电梯内部的九层按钮，按钮会响应并变色。

运行结束后结果如下：



到达楼层后按钮颜色恢复。

注：其他复杂的操作，请运行可执行文件，或者将源代码重新编译后自行实践。

**二．项目实现**

**1.界面布局**.

本程序是基于 java swing库实现的。源文件共有四个：MAIN.java, ELThread.java, View .Java和YButton.java. 其中ELThread.java 是将每个电梯作为一个线程，并在其中创建一个JPanel，在这个容器中添加Button等控件，实现电梯内部的运行和调度。在电梯内部，我定义了电梯运行方向，并默认初始值方向为stop.



并初始化每一层的电梯状态为false。

**2.监听事件**.

控件布局好后，我的Click监听事件通过对方向的判断，寻找下一个电梯终点。每当点击一个楼层按钮，会将该电梯的状态设置为true. 当电梯停止时，直接设置终点为点击的楼层。当电梯上升时，终点为上升楼层的最高楼层。当电梯下降时，终点为下降楼层中最低的楼层。这样方便实现当同方向过程中遇到多个点击过的楼层，只需要判断它的状态，作出停靠，改变状态的动作。

之后，在电梯线程运行过程中，我通过判断电梯运行的终点和当前楼层位置的比较，作出向上移动还是向下移动的操作。

**3.电梯移动.**

第一步，从当前楼层到终点楼层遍历，并每隔0.5秒改变楼层的颜色，实现电梯移动的效果。在便利过程中，每到一个楼层都会判断该楼层的状态。如果该楼层状态为true，即代表该楼层也是一个目的地，我们就让线程在这个楼层停止0.7秒，代表停靠该层。并且将该层的按钮颜色复原。最终到达终点楼层，并将该楼层记录为当前楼层。

第二步，在第一步的过程中，我们都是默认电梯是在单一方向运行的。但实际生活中，总会出现例如电梯在上升过程中有人按了比当前楼层低的楼层。我们的第二步就是解决这个问题的。每当单方向运行结束后，我们还要对整个楼层进行一次遍历，如果发现了有楼层的状态仍为true，这就代表是反方向的目的地，我们就将其设置为目的地，并执行反方向的移动操作。

**4.外部按钮**

View.java 我们以这个作为整个程序的Frame，在该容器里添加上述的五部电梯（线程），在该容器中添加外部按钮UP，DOWN. 并实现对线程的调度。我们对外部的UP，DOWN 按钮分别设置了一个数组表示该楼层是否按下按钮。按下按钮，我们便改变外部按钮的状态，并执行电梯的运行。我们将全部楼层一一遍历，假设某楼层向上按钮被按下，我们便会对五部电梯进行遍历，如果某电梯处于停止，或者正在向上且当前楼层低于该楼层时，我们便会调用该电梯响应该事件，并且在响应结束后，将该楼层的状态值改为5+响应的电梯序号（目的为当运行结束后，恢复楼层外部按钮，5没有特殊意义，可以用其他数字替代。）

电梯间的调度算法为，当外部按钮按下时，遍历五部电梯，找到停止的电梯或者方向与目的地相同的电梯，通过对五个电梯当前位置和目的地的距离做出比较，找到最近的符合要求（方向，状态）的电梯，进行调度运行。当五个电梯都在运行且都不符合调度条件时，外部的按钮会等待至有电梯符合条件再调度。在这个Frame中，我们新建了一个Thread 来管理其他的线程。

在MAIN.java 中，就是新建一个Frame 对象，并使之运行。

在YButton.java中，原先是对swing中JButton的一个改写，让按钮（楼层）本人拥有状态属性，但之后采用了状态的数组在外部进行管理，因此YButton和JButton没有分别，这个源文件可以忽略。（其他源文件中，Button均使用了YButton定义，但已经没有其他含义）