电梯调度操作系统

2253206韩明洋

# OS\_电梯调度

## 1 项目分析

### 1.1 项目背景

多线程指从软件或者硬件上实现多个线程并发执行的技术。具有多线程能力的计算机因有硬件支持而能够在同一时间执行多于一个线程提升整体处理性能。多线程是指程序中包含多个执行流，即在一个程序中可以同时运行多个不同制的线程来执行不同的任务，允许单个程序创建多个并行执行的线程来完成各自的任务。多线程的好处即为提高CPU的利用率。在多线程程序中，一个线程必须等待的时候，CPU可以运行其它的线程而不是等待，大大提高程序的效率。

电梯调度问题是经典的多线程问题，本项目的目的即为：

* 通过控制电梯调度，实现操作系统调度过程；
* 学习特定环境下多线程编程方法；
* 学习调度算法。

### 1.2 项目简介

某一层楼20层，有五部互联的电梯。基于线程思想，编写一个电梯调度程序。

### 1.3 项目需求

* 电梯应有一些按键，如：数字键、关门键、开门键、上行键、下行键、报警键等；
* 有数码显示器指示当前电梯状态；
* 每层楼、每部电梯门口，有上行、下行按钮、数码显示；
* 五部电梯相互联结，即当一个电梯按钮按下去时，其它电梯相应按钮同时点亮，表示也按下去了；
* 电梯调度算法：
  1. 所有电梯初始状态都在第一层；
  2. 每个电梯没有相应请求情况下，则应该在原地保持不动；
  3. 电梯调度算法自行设计。

## 2 开发环境

* 开发环境：**Windows10**
* 开发软件：
  1. **PyCharm**
  2. **Qt Designer**
* 开发语言：**python**
* 开发工具：
  1. pyqt5 (QTimer, QtCore, QtGui, QtWidgets)
  2. pyqt5-tools
  3. ......

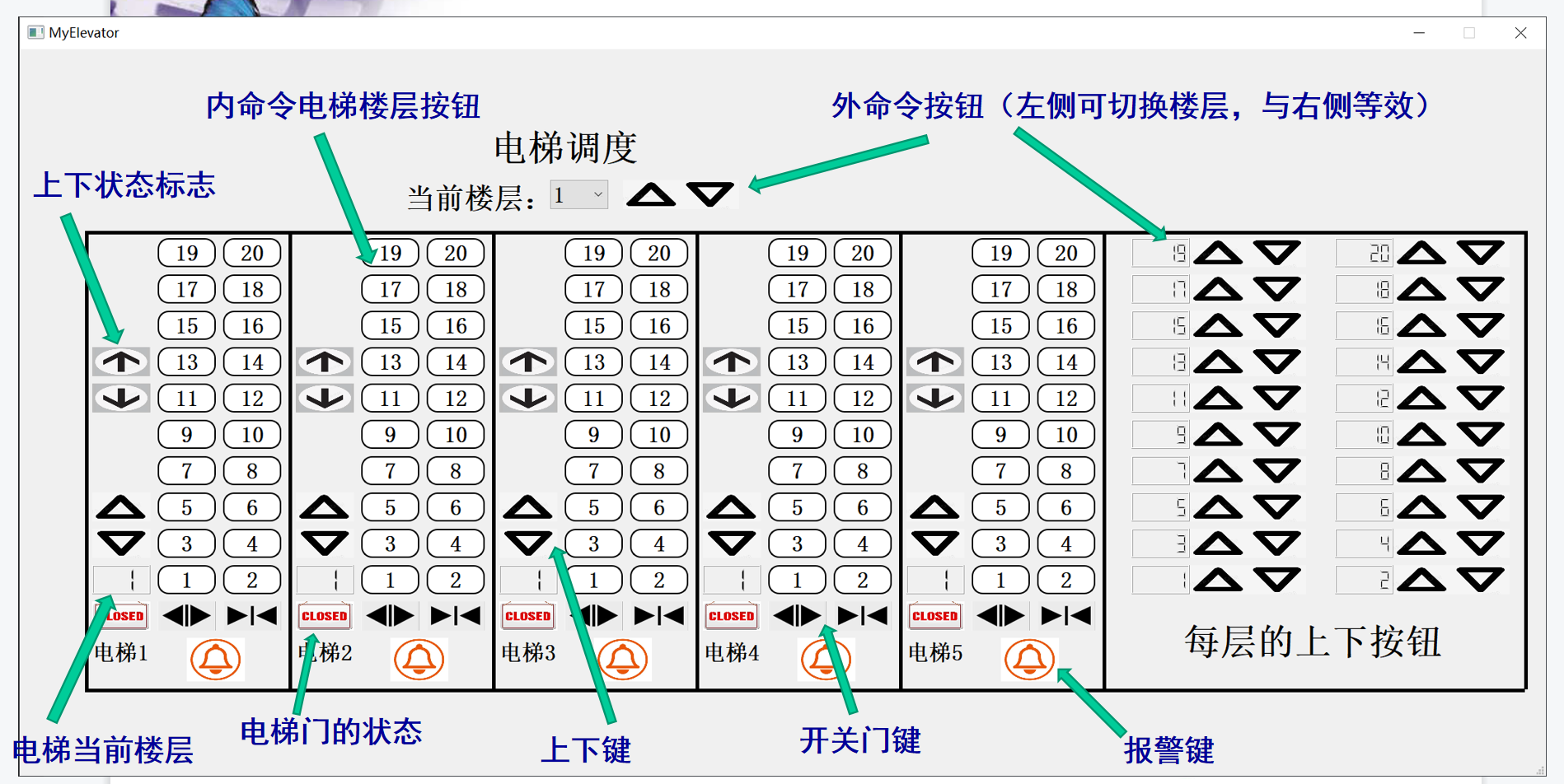
## 3 项目设计

本项目通过Qt Designer设计UI，并通过pyuic5转换为python文件实现图形化。使用python创建类和函数实现电梯调度。

五台电梯各设置其对应的消息列表（包括顺路列表和不顺路列表）

* 内命令信息加入对应电梯消息列表，
* 外命令信息通过算法选出最佳电梯后加入其消息列表。

每隔1s更新五台电梯的消息列表及其状态实现五个电梯的多线程调度。

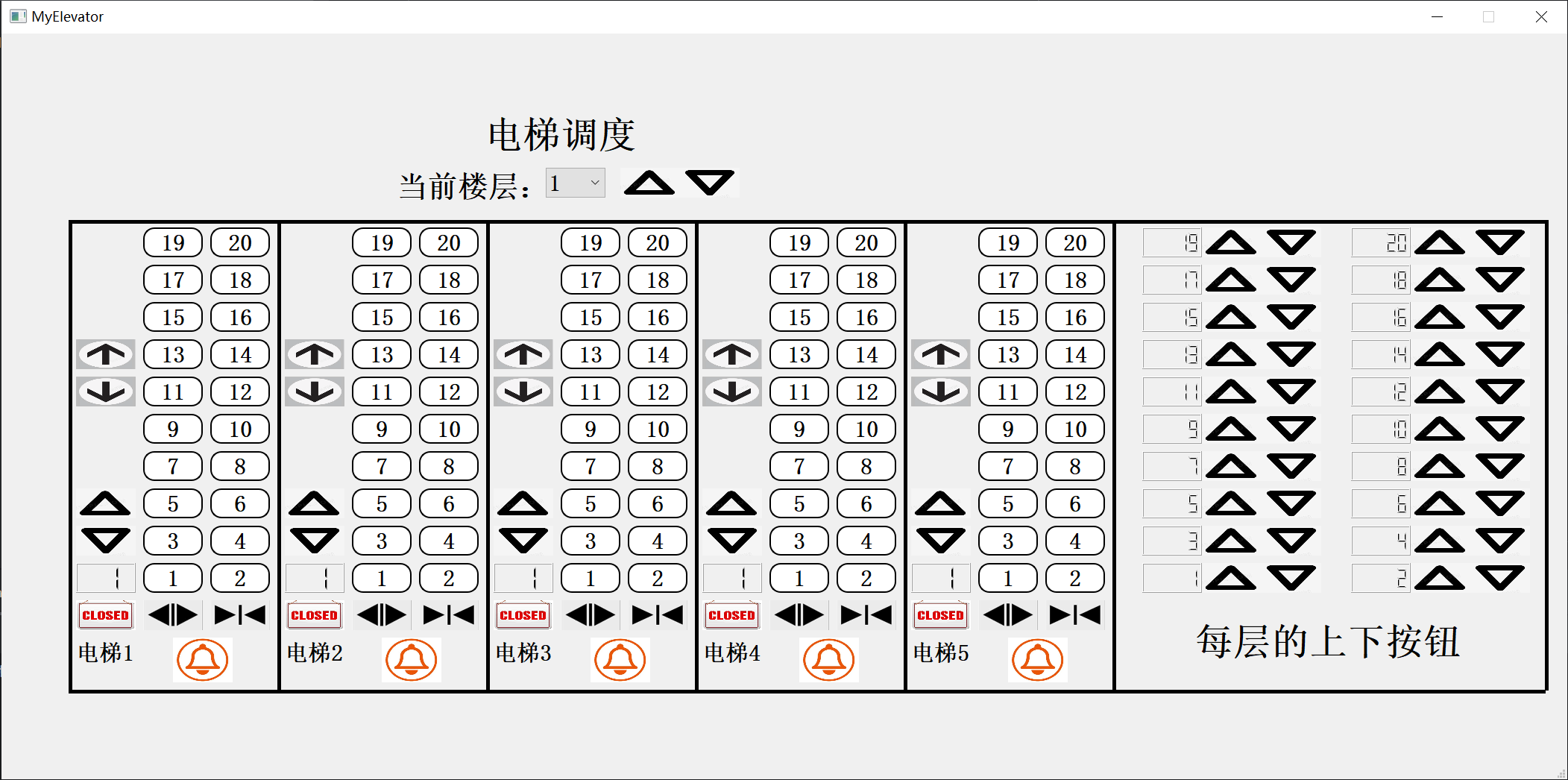


### 3.1 命令设计

* 单个电梯命令设计
* 事件：用户点击**楼层按钮**
* 响应：
  1. 若按键楼层与当前楼层相同 => 该电梯开门，并自动关闭。
  2. 若按键楼层与当前楼层不同:
  3. 若电梯为静止状态：将该楼层信息加入消息队列中。
  4. 若电梯为运行状态:
     1. 若电梯正在上行且按键楼层大于电梯此时楼层 => 将该楼层信息加入消息队列中
     2. 若电梯正在下行且按键楼层小于电梯此时楼层 => 将该楼层信息加入不顺路消息队列中
     3. 若电梯正在下行且按键楼层小于电梯此时楼层 => 将该楼层信息加入消息队列中
     4. 若电梯正在下行且按键楼层大于电梯此时楼层 => 将该楼层信息加入不顺路消息队列中
* 事件：用户点击**上下按钮**
* 响应：与外命令上下键衔接，作为当前电梯的当前楼层指令参与外命令调度。
* 事件：用户点击**报警键**
* 响应：
  1. 禁用当前电梯按钮，停止当前电梯参与调度
  2. 若该电梯任务不为空 => 该电梯任务清空，未完成的任务添加给其他电梯
  3. 若全部电梯都故障 => 弹窗提示关闭窗口
* 事件：用户点击**开关门键**
* 响应：门进行开关操作，但由于电梯状态每秒都在不断更新，为使其演示效果良好，设置了电梯门开启后在下次更新会关闭。
* 多个电梯外命令设计
* 事件：用户选择楼层/在对应楼层点击**上/下键**
* 响应：
  1. 从未故障的电梯中筛选可以响应的电梯中调度性最好的电梯。
  2. 若电梯为静止状态 => 距离为 该电梯当前楼层 - 用户楼层
  3. 若某电梯正在向上运动并且用户选择楼层大于该电梯当前楼层 => 距离为 用户楼层 - 该电梯当前楼层
  4. 若某电梯正在向下运动并且用户选择楼层小于该电梯当前楼层 => 距离为 该电梯当前楼层 - 用户楼层
  5. 其他状态的电梯距离设置为默认值
  6. 从未故障的电梯中选出距离最短的电梯作为调度性最好的电梯
  7. 若最佳电梯当前就在用户选择的楼层：该电梯开门，并自动关闭。
  8. 否则将用户楼层信息加入该最佳电梯的消息队列中。

### 3.2 界面设计

#### 3.2.1 整体设计



| 模型名 | 模型定义 |
| --- | --- |
| 外命令上下总按钮模型，外命令上下按钮模型 | **QtWidgets.QPushButton** |
| 外命令楼层选择框模型 | **QtWidgets.QComboBox** |
| 内命令楼层按钮模型 | **QtWidgets.QPushButton** |
| 报警按钮模型 | **QtWidgets.QPushButton** |
| 上下键模型 | **QtWidgets.QPushButton** |
| 上下行状态模型 | **QtWidgets.QGraphicsView** |
| 开关门按钮模型 | **QtWidgets.QPushButton** |
| 门的开关门状态模型 | **QtWidgets.QGraphicsView** |
| 楼层数码管模型 | **QtWidgets.QLCDNumber** |
| 总标签、电梯标签等标签模型 | **QtWidgets.QLabel** |
| 分割线模型 | **QtWidgets.QFrame** |

#### 3.2.3 状态设计

* 门状态
* OPEN = 0 # 开门状态  
  CLOSED = 1 # 关门状态
* 电梯运行状态
* STANDSTILL = 0 # 静止状态  
  RUNNING\_UP = 1 # 电梯上行状态  
  RUNNING\_DOWN = 2 # 电梯下行状态
* 电梯待定状态
* NOPE = 0 # 空  
  READYSTART = 1 # 电梯即将运动  
  READYSTOP = 2 # 电梯即将停止
* 用户选择状态
* GOUP = 1 # 用户要上行  
  GODOWN = 2 # 用户要下行

### 3.1 类ui\_MyElevator设计

即设计的UI的类，并于命令MyElevator类的函数绑定

class ui\_MyElevator(object):  
  
 def \_\_init\_\_(self):  
 self.Ctrl = MyElevator(self) # 与调度文件建立连接  
 self.elevEnabled = [True] \* 5 # 电梯状态(可使用/禁用)标志位  
 self.doorState = [CLOSED] \* 5 # 电梯门状态(开门/关门)标志位  
 self.elevState = [STANDSTILL] \* 5 # 电梯状态(运行向上/运行向下/静止)标志位  
 self.ReadyState = [NOPE] \* 5 # 待定状态(空/即将运动/即将停止)标志位  
  
 self.Floor = [[0 for j in range(20)] for i in range(5)] # 五个电梯的模型  
 self.label = [0 for i in range(5)] # 电梯名称的五个标签模型  
 self.Floor\_now = [0 for i in range(5)] # 五个电梯的当前层数模型  
 self.floor\_now = [0 for i in range(5)]  
 self.warning = [0 for i in range(5)] # 五个电梯的警报器模型  
 self.up = [0 for i in range(5)] # 电梯名称的五个上升模型  
 self.down = [0 for i in range(5)] # 五个电梯的下降模型  
 self.up\_logo = [0 for i in range(5)] # 电梯名称的五个上升  
 self.down\_logo = [0 for i in range(5)] # 五个电梯的下降  
 self.open = [0 for i in range(5)] # 五个电梯的开门模型  
 self.close = [0 for i in range(5)] # 五个电梯的关门模型  
 self.line = [0 for i in range(6)] # 6条分割线模型  
 self.door = [0 for i in range(6)] # 6条门模型  
  
 self.Floor\_now\_ex = [0 for i in range(20)] # 右侧按钮区域  
 self.up\_ex = [0 for i in range(20)]  
 self.down\_ex = [0 for i in range(20)]  
  
 self.outMessage = [0 for i in range(20)] # 记录每层上下行键的模型  
  
 self.up\_total = None  
 self.down\_total = None  
 self.Floor\_now\_total = None  
   
 def setupUi(self, MyElevator\_UI):  
   
 # 初始化填入按钮字符  
 def retranslateUi(self, MyElevator\_UI):  
   
 # 报警器  
 def warningClick(self):  
   
 # 楼层按键  
 def floorClick(self):  
   
 # 开关门  
 def doorClick(self):  
   
 # 上下键选择  
 def updownClick(self):

### 3.2 类MyElevator设计

即用于电梯的控制及调度的类

class MyElevator(object):  
 def \_\_init\_\_(self, Elev):  
 # 与界面文件建立连接  
 self.elev = Elev  
 # 创建定时器, 1s中更新一次电梯状态  
 self.timer = QTimer()  
 self.timer.timeout.connect(self.updateElevState)  
 self.timer.start(1000)  
 # 5个电梯内部消息列表  
 self.messageQueue = []  
 for i in range(0, 5):  
 self.messageQueue.append([])  
 # 5个电梯内部不顺路消息列表  
 self.messageQueue\_reverse = []  
 for i in range(0, 5):  
 self.messageQueue\_reverse.append([])  
   
 # 警报器函数  
 def warnCtrl(self, which\_elev):  
   
 # 开关门函数  
 def doorCtrl(self, which\_elev, which\_command):  
   
 # 内命令  
 def innerCtrl(self, which\_elev, dest):  
   
 # 外命令  
 def outerCtrl(self, which\_floor, choice):  
   
 # 更新电梯状态  
 def updateElevState(self):

### 3.3 类main\_MyElevator设计

主界面

class main\_MyElevator(QtWidgets.QMainWindow, ui\_MyElevator):  
 def \_\_init\_\_(self):  
 super(main\_MyElevator, self).\_\_init\_\_()  
 self.setupUi(self)  
 self.setWindowTitle('MyElevator')

### 3.4 main函数

if \_\_name\_\_ == '\_\_main\_\_':  
 app = QApplication(sys.argv)  
 window = main\_MyElevator()  
 window.show()  
 sys.exit(app.exec())

## 4 功能实现

### 4.1 报警器

用户点击某报警器 => 更改该电梯按键为红色 => 调用warnCtrl函数处理 =>

该电梯状态设置为禁用 => 设置该电梯各部件禁用并停止所有操作

如果该电梯还有任务未完成 => 将其未完成的命令作为外命令传递给其他电梯

如果所有5部电梯都禁用 => 设置下拉框等外命令空间禁用 => 弹窗提示关闭窗口

# 报警器  
 def warningClick(self):  
 which\_btn = int(self.sender().objectName()[-1]) - 1  
 print("点击了{0}号报警器".format(which\_btn))  
 self.warning[which\_btn].setStyleSheet("border-image:url(image/warning\_work.png)")  
 self.Ctrl.warnCtrl(which\_btn) # 调用控制器进行warnCtrl处理  
  
  
  
# 警报器函数  
 def warnCtrl(self, which\_elev):  
 self.elev.elevEnabled[which\_elev] = False # 该电梯禁用  
 self.elev.warning[which\_elev].setEnabled(False) # 报警键禁用  
 for i in range(0, 20):  
 self.elev.Floor[which\_elev][i].setStyleSheet("background-color: red;")  
 self.elev.Floor[which\_elev][i].setEnabled(False) # 楼层按键禁用  
 self.elev.open[which\_elev].setStyleSheet("border-image: url(image/open\_ban.png)")  
 self.elev.open[which\_elev].setEnabled(False) # 开门键禁用  
 self.elev.close[which\_elev].setStyleSheet("border-image: url(image/close\_ban.png)")  
 self.elev.close[which\_elev].setEnabled(False) # 关门键禁用  
 self.elev.Floor\_now[which\_elev].setEnabled(False) # 数码管禁用  
 self.elev.up\_logo[which\_elev].setStyleSheet("border-image: url(image/upEle\_ban.png)")  
 self.elev.down\_logo[which\_elev].setStyleSheet("border-image: url(image/downEle\_ban.png)")  
 self.elev.up\_logo[which\_elev].setEnabled(False) # 上下行标志禁用  
 self.elev.down\_logo[which\_elev].setEnabled(False)  
 self.elev.up[which\_elev].setStyleSheet("border-image: url(image/up\_ban.png)")  
 self.elev.down[which\_elev].setStyleSheet("border-image: url(image/down\_ban.png)")  
 self.elev.up[which\_elev].setEnabled(False) # 门口上行按钮禁用  
 self.elev.down[which\_elev].setEnabled(False) # 门口下行按钮禁用  
 self.elev.door[which\_elev].setStyleSheet("border-image: url(image/door\_close.png)")  
  
 # 将点击警报的电梯未完成的调度信息传递给其他电梯  
 for i in range(0, len(self.messageQueue[which\_elev])):  
 self.outerCtrl(self.messageQueue[which\_elev][i], 1)  
 for i in range(0, len(self.messageQueue\_reverse[which\_elev])):  
 self.outerCtrl(self.messageQueue\_reverse[which\_elev][i], 1)  
  
 self.messageQueue[which\_elev].clear() # 清空该电梯的消息列表，终止操作  
 self.messageQueue\_reverse[which\_elev].clear()  
  
 # 五部电梯全部禁用  
 arr = np.array(self.elev.elevEnabled)  
 if (arr == False).all():  
 self.elev.Floor\_now\_total.setEnabled(False) # 下拉框禁用  
 self.elev.up\_total.setStyleSheet("border-image: url(image/up\_ban.png)")  
 self.elev.down\_total.setStyleSheet("border-image: url(image/down\_ban.png)")  
 self.elev.up\_total.setEnabled(False) # 上行按钮禁用  
 self.elev.down\_total.setEnabled(False) # 下行按钮禁用  
 for i in range(0, 20):  
 self.elev.up\_ex[i].setStyleSheet("border-image: url(image/up\_ban.png)")  
 self.elev.down\_ex[i].setStyleSheet("border-image: url(image/down\_ban.png)")  
 self.elev.up\_ex[i].setEnabled(False) # 上行按钮禁用  
 self.elev.down\_ex[i].setEnabled(False) # 下行按钮禁用  
 win32api.MessageBox(0, "所有电梯已损坏!\n请关闭电梯调度页面", "警告", win32con.MB\_ICONASTERISK)

### 4.2 开关门

用户点击某个开/关门按钮 => 调用doorCtrl函数处理 =>

如果用户要开门：如果当前门是CLOSED状态并且电梯处于STANDSTILL状态 => 门的状态改为OPEN => 电梯状态更新为禁用

如果用户要关门：如果当前门是OPEN状态并且电梯处于STANDSTILL状态 => 门的状态改为CLOSED => 电梯状态更新为允许使用

# 开关门  
 def doorClick(self):  
 door\_name = self.sender().objectName()  
 which\_elev = int(door\_name[-1])  
 which\_command = 0 if door\_name[0] == 'o' else 1 # 0 => 开门 1 => 关门  
 print("{0}号电梯, 命令是{1}（0为开门，1为关门）".format(which\_elev + 1, which\_command))  
  
 self.Ctrl.doorCtrl(which\_elev - 1, which\_command) # 调用控制器进行doorCtrl处理  
   
   
# 开关门函数  
 def doorCtrl(self, which\_elev, which\_command):  
 if which\_command == 0: # 如果用户要开门  
 # 开门动画  
 self.elev.door[which\_elev].setStyleSheet("border-image:url(image/door\_open.png)")  
 # 如果当前门是关闭状态并且电梯是静止的  
 if self.elev.doorState[which\_elev] == CLOSED and self.elev.elevState[which\_elev] == STANDSTILL:  
 self.elev.doorState[which\_elev] = OPEN # 先将门状态更新为打开  
 self.elev.elevEnabled[which\_elev] = False  
 else: # 如果用户要关门  
 # 关门动画  
 self.elev.door[which\_elev].setStyleSheet("border-image:url(image/door\_close.png)")  
 # 如果当前门是打开状态并且电梯是静止的  
 if self.elev.doorState[which\_elev] == OPEN and self.elev.elevState[which\_elev] == STANDSTILL:  
 self.elev.doorState[which\_elev] = CLOSED # 先将门状态更新为关闭  
 self.elev.elevEnabled[which\_elev] = True

### 4.3 楼层键内命令

用户点击某个楼层按键 => 调用控制器进行innerCtrl处理 =>

如果按键楼层大于当前楼层:

* 电梯处于STANDSTILL状态 => 将目标楼层加入**消息队列**
* 电梯正在RUNNING\_UP状态 => 将目标楼层加入**消息队列**并排序
* 电梯正在RUNNING\_DOWN状态 => 将目标楼层加入不顺路消息队列\*\*并排序

如果按键楼层小于当前楼层:

* 电梯处于STANDSTILL状态 => 将目标楼层加入**消息队列**
* 电梯正在RUNNING\_DOWN状态 => 将目标楼层加入**消息队列**并反向排序
* 电梯正在RUNNING\_UP状态 => 将目标楼层加入**不顺路消息队列**并反向排序

如果按键就为当前楼层:

* 电梯处于STANDSTILL状态 => 开门并关门 => 恢复按键背景并重新允许点击

# 楼层按键  
 def floorClick(self):  
 which\_btn = self.sender()  
 btn\_name = which\_btn.objectName()  
 buf = [int(s) for s in btn\_name if s.isdigit()] # 提取字符串中的数字  
 which\_elev = buf[0]  
 which\_floor = buf[1]  
 if len(buf) > 2:  
 which\_floor = which\_floor \* 10 + buf[2]  
 print("{0}号电梯, 第{1}层按键被按，若不是当前楼层，则加入消息列表".format(which\_elev, which\_floor))  
 which\_btn.setStyleSheet(  
 "border-style: outset;\n"  
 "border-width: 2px;\n"  
 "border-radius: 15px;\n"  
 "border-color: black;\n"  
 "padding: 4px;\n"  
 "background-color: orange\n"  
 )  
 which\_btn.setEnabled(False) # 将该按钮设置为不可点击状态  
 self.Ctrl.innerCtrl(which\_elev - 1, which\_floor) # 调用控制器进行innerCtrl处理  
   
   
 def innerCtrl(self, which\_elev, dest):  
 nowFloor = self.elev.floor\_now[which\_elev] # 获取当前电梯位置  
  
 if nowFloor < dest: # 如果按键大于当前楼层  
 if self.elev.elevState[which\_elev] == STANDSTILL: # 电梯处于静止状态  
 self.messageQueue[which\_elev].append(dest) # 将目标楼层加入 消息队列  
 else:  
 if self.elev.elevState[which\_elev] == RUNNING\_UP: # 电梯正在向上运行  
 self.messageQueue[which\_elev].append(dest) # 将目标楼层加入 消息队列并排序  
 self.messageQueue[which\_elev].sort()  
 elif self.elev.elevState[which\_elev] == RUNNING\_DOWN: # 电梯正在向下运行  
 self.messageQueue\_reverse[which\_elev].append(dest) # 将目标楼层加入 不顺路消息队列并排序  
 self.messageQueue\_reverse[which\_elev].sort()  
  
 elif nowFloor > dest: # 如果按键小于当前楼层  
 if self.elev.elevState[which\_elev] == STANDSTILL:  
 self.messageQueue[which\_elev].append(dest) # 将目标楼层加入 消息队列  
  
 else:  
 if self.elev.elevState[which\_elev] == RUNNING\_DOWN:  
 self.messageQueue[which\_elev].append(dest) # 将目标楼层加入 消息队列并反向排序  
 self.messageQueue[which\_elev].sort()  
 self.messageQueue[which\_elev].reverse()  
 elif self.elev.elevState[which\_elev] == RUNNING\_UP:  
 self.messageQueue\_reverse[which\_elev].append(dest) # 将目标楼层加入 不顺路消息队列并反向排序  
 self.messageQueue\_reverse[which\_elev].sort()  
 self.messageQueue\_reverse[which\_elev].reverse()  
  
 else: # 如果按键就为当前楼层  
 print("该楼层即为当前楼层，直接结束")  
 if self.elev.elevState[which\_elev] == STANDSTILL: # 电梯静止 => 打开门  
 self.elev.doorState[which\_elev] = OPEN  
 # 开门动画  
 self.elev.door[which\_elev].setStyleSheet("border-image:url(image/door\_open.png)")  
 # 关门动画  
 self.elev.door[which\_elev].setStyleSheet("border-image:url(image/door\_close.png)")  
 self.elev.doorState[which\_elev] = CLOSED  
 # 如果到达目标楼层，且目标楼层与总数码管一致，则熄灭上升或下降键  
 self.elev.up\_ex[dest-1].setStyleSheet(  
 "QPushButton{border-image: url(image/up.png)}")  
 self.elev.up\_ex[dest-1].setEnabled(True)  
 self.elev.down\_ex[dest - 1].setStyleSheet(  
 "QPushButton{border-image: url(image/down.png)}")  
 self.elev.down\_ex[dest - 1].setEnabled(True)  
  
 for j in range(0, 5):  
 if self.elev.up[j].isEnabled() == False:  
 self.elev.up[j].setStyleSheet(  
 "QPushButton{border-image: url(image/up.png)}")  
 self.elev.up[j].setEnabled(True)  
 if self.elev.down[j].isEnabled() == False:  
 self.elev.down[j].setStyleSheet(  
 "QPushButton{border-image: url(image/down.png)}")  
 self.elev.down[j].setEnabled(True)  
  
 button = self.elev.findChild(QtWidgets.QPushButton,  
 "Floor\_{}\_{}".format(which\_elev + 1, nowFloor)) # 恢复按键背景并重新允许点击  
 button.setStyleSheet("border-style: outset;\n"  
 "border-width: 2px;\n"  
 "border-radius: 15px;\n"  
 "border-color: black;\n"  
 "padding: 4px;\n"  
 "background-color: white;")  
 button.setEnabled(True)

### 4.4 上下键外命令

获取用户选择的楼层 => 获取上/下行并转换为**用户选择状态** => 调用outerCtrl函数处理 =>

初步筛选没损坏的电梯 => 选择可调度性最好的电梯作为最佳电梯

1. 若电梯为静止状态 => 距离为 该电梯当前楼层 - 用户楼层
2. 若某电梯正在向上运动并且用户选择楼层大于该电梯当前楼层 => 距离为 用户楼层 - 该电梯当前楼层
3. 若某电梯正在向下运动并且用户选择楼层小于该电梯当前楼层 => 距离为 该电梯当前楼层 - 用户楼层
4. 其他状态的电梯距离设置为默认值
5. 从未故障的电梯中选出距离最短的电梯作为调度性最好的电梯

如果外部指令要添加的指令最佳电梯已存在 => 无效

否则 => 按楼层调用该电梯的内命令

# 上下键选择  
 def updownClick(self):  
 which\_floor = int(self.Floor\_now\_total.currentText())  
  
 which\_btn = self.sender().objectName()  
 flag = 1  
 if which\_btn[0] == 'd':  
 if which\_btn[5] == 'e':  
 flag = 0  
 temp = int(which\_btn[8])  
 if len(which\_btn) > 9:  
 temp = temp \* 10 + int(which\_btn[9])  
 which\_floor = int(self.Floor\_now\_ex[temp - 1].value())  
 else:  
 if which\_btn[3] == 'e':  
 flag = 0  
 temp = int(which\_btn[6])  
 if len(which\_btn) > 7:  
 temp = temp \* 10 + int(which\_btn[7])  
 which\_floor = int(self.Floor\_now\_ex[temp - 1].value())  
  
 if which\_btn[0] == 'd':  
 choice = GODOWN  
 self.down\_ex[which\_floor - 1].setStyleSheet(  
 "QPushButton{border-image: url(image/down\_work.png)}")  
 self.down\_ex[which\_floor - 1].setEnabled(False)  
 if flag:  
 for i in range(0, 5):  
 if self.elevEnabled[i]:  
 self.down[i].setStyleSheet(  
 "QPushButton{border-image: url(image/down\_work.png)}")  
 self.down[i].setEnabled(False)  
 else:  
 choice = GOUP  
 self.up\_ex[which\_floor - 1].setStyleSheet(  
 "QPushButton{border-image: url(image/up\_work.png)}")  
 self.up\_ex[which\_floor - 1].setEnabled(False)  
 if flag:  
 for i in range(0, 5):  
 if self.elevEnabled[i]:  
 self.up[i].setStyleSheet(  
 "QPushButton{border-image: url(image/up\_work.png)}")  
 self.up[i].setEnabled(False)  
 print("用户在第{0}层按下了{1}键（1为上升，2为下降）".format(which\_floor, choice))  
 self.Ctrl.outerCtrl(which\_floor, choice) # 调用控制器进行outerCtrl处理  
   
   
 # 外命令  
 def outerCtrl(self, which\_floor, choice):  
 # 防止已经禁用的电梯又被启动  
 for i in range(0, 5):  
 if self.elev.warning[i].isEnabled() == False:  
 self.elev.elevEnabled[i] = False # 该电梯禁用  
 # 初步筛选没损坏的电梯  
 EnabledList = []  
 for i in range(0, 5):  
 if self.elev.elevEnabled[i]:  
 EnabledList.append(i)  
 print("未损坏的电梯为：" + str(EnabledList) + "该处01234对应12345电梯")  
 if len(EnabledList) == 0:  
 print("所有电梯均故障，调度无法进行")  
 return  
  
 # 计算每部可用电梯的"可调度性"  
 dist = [MAXINF] \* 5 # 可使用电梯距离用户的距离  
 for EnabledElev in EnabledList:  
 # 如果该电梯正在上升，并且该选择是上升，且目标楼层在当前楼层之上  
 if self.elev.elevState[EnabledElev] == RUNNING\_UP and choice == GO\_UP \  
 and which\_floor > self.elev.floor\_now[EnabledElev]:  
 dist[EnabledElev] = which\_floor - self.elev.floor\_now[EnabledElev]  
 # 如果该电梯正在下降，并且该选择是下降，且目标楼层在当前楼层之下  
 elif self.elev.elevState[EnabledElev] == RUNNING\_DOWN and choice == GO\_DOWN and which\_floor < \  
 self.elev.floor\_now[EnabledElev]:  
 dist[EnabledElev] = self.elev.floor\_now[EnabledElev] - which\_floor  
 # 如果该电梯静止  
 elif self.elev.elevState[EnabledElev] == STANDSTILL:  
 dist[EnabledElev] = abs(self.elev.floor\_now[EnabledElev] - which\_floor)  
  
 # 防止已经禁止的电梯参与调度  
 for i in range(0, 5):  
 if self.elev.warning[i].isEnabled() == False:  
 dist[i] = MAXINF\_ex  
  
 BestElev = dist.index(min(dist)) # 选择可调度性最好的电梯作为最佳电梯  
 print("该条指令选中第{}电梯".format(BestElev + 1))  
  
  
 # 如果外部指令要添加内容电梯里已经存在，则无效  
 flag = 1  
 for i in range(0, len(self.messageQueue[BestElev])):  
 if self.messageQueue[BestElev][i] == which\_floor:  
 flag = 0  
 print("该指令重复，无效")  
 break  
 if flag == 1:  
 for i in range(0, len(self.messageQueue\_reverse[BestElev])):  
 if self.messageQueue\_reverse[BestElev][i] == which\_floor:  
 flag = 0  
 print("该指令重复，无效")  
 break  
 if flag:  
 button = self.elev.findChild(QtWidgets.QPushButton,  
 "Floor\_{0}\_{1}".format(BestElev + 1, which\_floor)) # 将用户的目标楼层设定为特殊颜色  
 button.setStyleSheet("border-style: outset;\n"  
 "border-width: 2px;\n"  
 "border-radius: 15px;\n"  
 "border-color: black;\n"  
 "padding: 4px;\n"  
 "background-color: orange;")  
 button.setEnabled(False)  
 self.innerCtrl(BestElev, which\_floor) # 调用控制器进行innerCtrl处理

### 4.5 状态更新

筛选未损坏的电梯 => 遍历五部电梯 => 某个电梯的消息队列不为空 =>

如果电梯门处于开门状态 => 关门

* 电梯处于STANDSTILL状态 => 开门 => 根据即将运行的方向更新电梯状态 => 待定状态变为READYSTART状态
* 待定状态处于READYSTART状态 => 关门 => 待定状态变为NOPE状态
* 待定状态处于READYSTOP状态 => 结束该命令的处理 => 待定状态变为NOPE状态 => 电梯变为STANDSTILL状态

获取顺路消息列表中第一个目标楼层 =>

* 向上运动: 当前楼层小于目标楼层 => 电梯状态变为RUNNING\_UP => 显示向上运行图标 => 将当前楼层加一并设置数码管显示
* 向下运动: 当前楼层大于目标楼层 => 电梯状态变为RUNNING\_DOWN => 显示向下运行图标 => 将当前楼层减一并设置数码管显示
* 电梯到达目的地: 当前楼层等于目标楼层 => 开门 => 待定状态变为READYSTOP状态 => 将该楼层按钮恢复到原始状态

某个电梯的顺路消息队列为空且不顺路消息队列不为空 => 交换两个队列

# 创建定时器, 1s中更新一次电梯状态  
 self.timer = QTimer()  
 self.timer.timeout.connect(self.updateElevState)  
 self.timer.start(1000)  
   
# 更新电梯状态  
 def updateElevState(self):  
  
 # 初步筛选没损坏的电梯  
 EnabledList = []  
 for i in range(0, 5):  
 if self.elev.elevEnabled[i]:  
 EnabledList.append(i)  
  
 for i in range(0, 5): # 遍历五部电梯  
 if self.elev.warning[i].isEnabled() == False:  
 continue  
 if self.elev.doorState[i] == OPEN: # 如果电梯门是打开的 => 等待电梯关门  
 self.elev.door[i].setStyleSheet("border-image:url(image/door\_close.png)") # 关门动画  
 self.elev.doorState[i] = CLOSED  
  
 if len(self.messageQueue[i]) > 0: # 某个电梯的消息队列不为空  
 if self.elev.elevState[i] == STANDSTILL: # 电梯处于静止状态  
 self.elev.door[i].setStyleSheet("border-image:url(image/door\_open.png)") # 开门动画  
  
 if self.elev.floor\_now[i] < self.messageQueue[i][0]: # 根据即将运行的方向更新电梯状态  
 self.elev.elevState[i] = RUNNING\_UP  
 elif self.elev.floor\_now[i] > self.messageQueue[i][0]:  
 self.elev.elevState[i] = RUNNING\_DOWN  
  
 self.elev.ReadyState[i] = READY\_START # 变为就绪运行状态  
  
 elif self.elev.ReadyState[i] == READY\_START: # 处于就绪运行状态  
 # 关门动画  
 self.elev.door[i].setStyleSheet("border-image:url(image/door\_close.png)")  
 self.elev.ReadyState[i] = NOPE # 变为空状态  
  
 elif self.elev.ReadyState[i] == READY\_STOP: # 处于就绪停止状态  
 self.messageQueue[i].pop(0) # 结束该命令的处理  
  
 self.elev.door[i].setStyleSheet("border-image:url(image/door\_close.png)") # 关门动画  
 self.elev.ReadyState[i] = NOPE # 变为空状态  
 self.elev.elevState[i] = STANDSTILL # 电梯变为静止状态  
 self.elev.up\_logo[i].setStyleSheet("QGraphicsView{border-image: url(image/upEle.png)}")  
 self.elev.down\_logo[i].setStyleSheet("QGraphicsView{border-image: url(image/downEle.png)}")  
  
 else:  
 destFloor = self.messageQueue[i][0] # 获取第一个目标楼层  
 if self.elev.floor\_now[i] < destFloor: # 向上运动  
 self.elev.elevState[i] = RUNNING\_UP  
 self.elev.up\_logo[i].setStyleSheet("QGraphicsView{border-image: url(image/upEle\_work.png)}")  
 self.elev.floor\_now[i] = self.elev.floor\_now[i] + 1 # 将当前楼层加一并设置数码管显示  
 self.elev.Floor\_now[i].setProperty("value", self.elev.floor\_now[i])  
  
 elif self.elev.floor\_now[i] > destFloor: # 向下运动  
 self.elev.elevState[i] = RUNNING\_DOWN  
 self.elev.down\_logo[i].setStyleSheet("QGraphicsView{border-image: url(image/downEle\_work.png)}")  
 self.elev.floor\_now[i] = self.elev.floor\_now[i] - 1 # 将当前楼层减一并设置数码管显示  
 self.elev.Floor\_now[i].setProperty("value", self.elev.floor\_now[i])  
  
 else: # 电梯到达目的地  
 self.elev.door[i].setStyleSheet("border-image:url(image/door\_open.png)") # 开门动画  
 self.elev.doorState[i] = OPEN  
  
 self.elev.ReadyState[i] = READY\_STOP # 到达目的地 => 变为就绪停止状态  
 # 如果到达目标楼层，且目标楼层与总数码管一致，则熄灭上升或下降键  
 if self.elev.up\_ex[destFloor - 1].isEnabled() == False:  
 self.elev.up\_ex[destFloor - 1].setStyleSheet(  
 "QPushButton{border-image: url(image/up.png)}")  
 self.elev.up\_ex[destFloor - 1].setEnabled(True)  
 if self.elev.down\_ex[destFloor - 1].isEnabled() == False:  
 self.elev.down\_ex[destFloor - 1].setStyleSheet(  
 "QPushButton{border-image: url(image/down.png)}")  
 self.elev.down\_ex[destFloor - 1].setEnabled(True)  
 if destFloor == int(self.elev.Floor\_now\_total.currentText()):  
 for j in EnabledList:  
 if self.elev.up[j].isEnabled() == False:  
 self.elev.up[j].setStyleSheet(  
 "QPushButton{border-image: url(image/up.png)}")  
 self.elev.up[j].setEnabled(True)  
 elif self.elev.down[j].isEnabled() == False:  
 self.elev.down[j].setStyleSheet(  
 "QPushButton{border-image: url(image/down.png)}")  
 self.elev.down[j].setEnabled(True)  
  
 button = self.elev.findChild(QtWidgets.QPushButton,  
 "Floor\_{0}\_{1}".format(i + 1, self.elev.floor\_now[i])) # 恢复该按钮的状态  
 button.setStyleSheet(  
 "border-style: outset;\n"  
 "border-width: 2px;\n"  
 "border-radius: 15px;\n"  
 "border-color: black;\n"  
 "padding: 4px;\n"  
 "background-color: white;\n")  
 button.setEnabled(True)  
  
 elif len(self.messageQueue\_reverse[i]): # 如果消息队列为空 & 不顺路消息队列不为空  
 self.messageQueue[i] = self.messageQueue\_reverse[i].copy() # 交替两个队列  
 self.messageQueue\_reverse[i].clear()

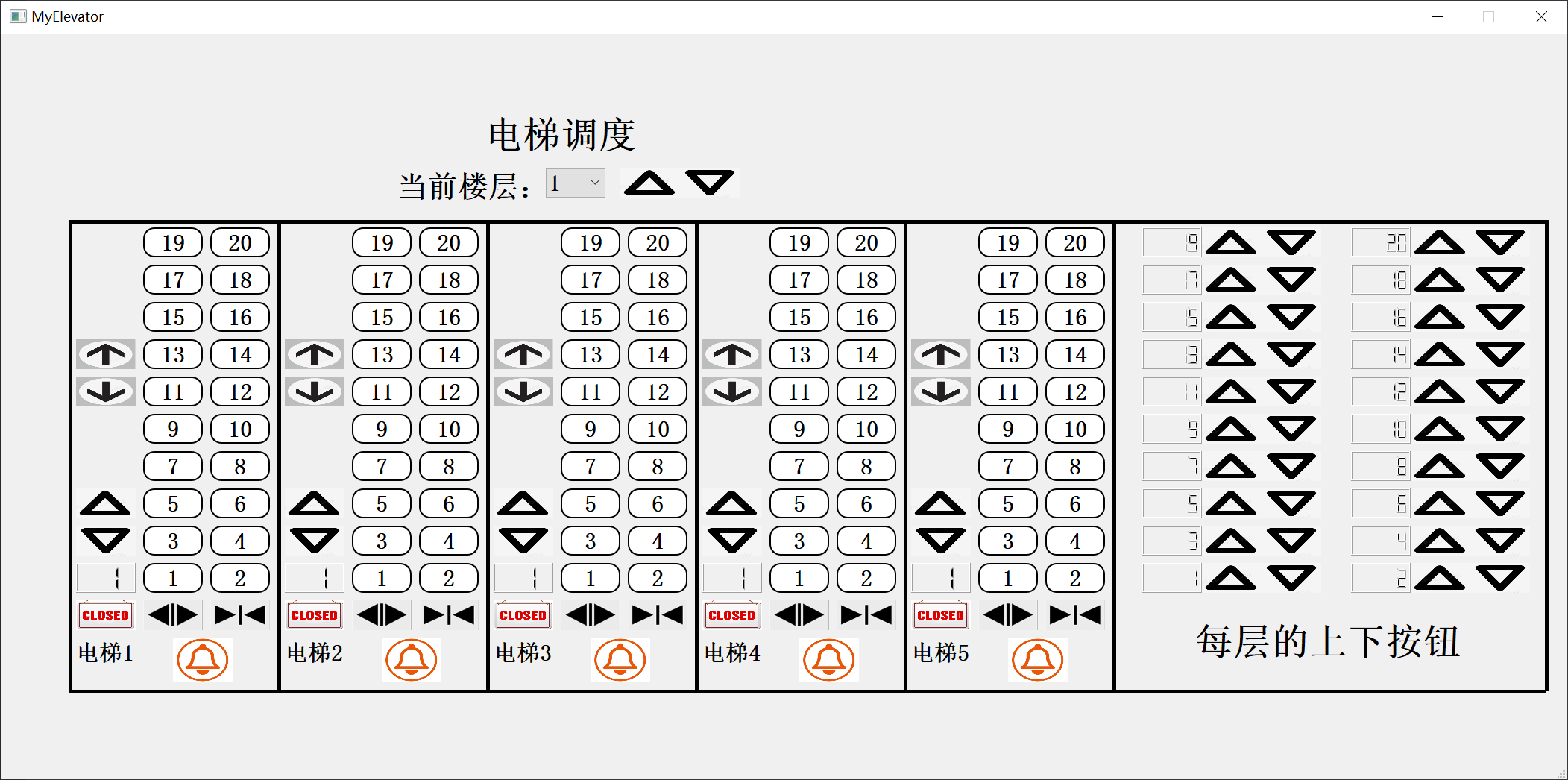
## 5 How To Run

执行该文件的方法如下两种方法啊：

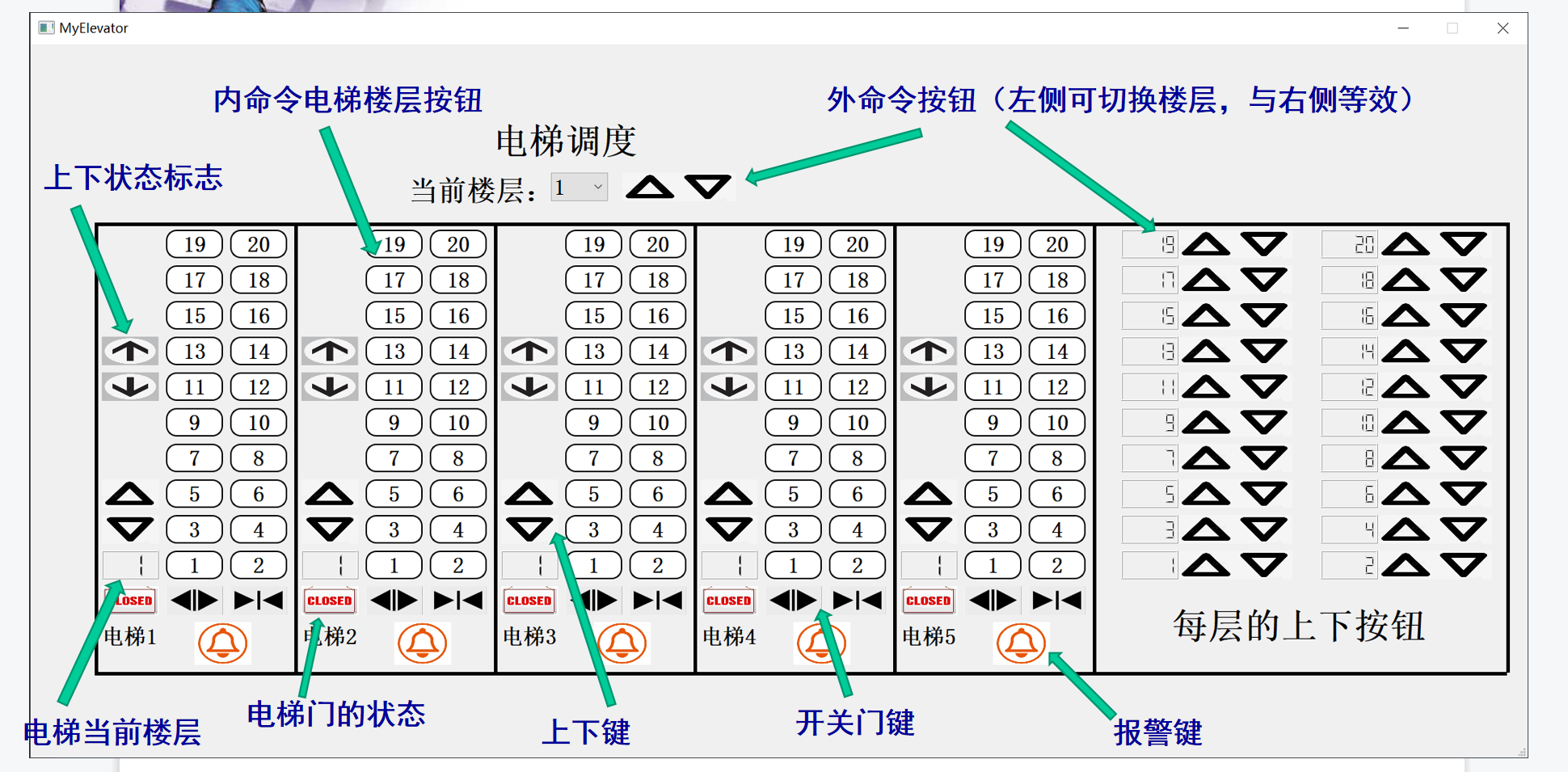
* 打开该目录下 **code和exe** 文件夹，直接运行main.exe （若直接在服务器上运行，可能需要等待 **1.5 min** 左右）
* 打开该目录下 **code和exe** 文件夹，运行main.py程序 （若直接在服务器上运行，可能需要等待 **十几秒**）

**若直接在服务器上运行，运行时间可能较长，请耐心等待......**

弹出程序界面如下：



其各部分标志及按键功能如下图标识所示：



* 点击内命令电梯楼层按钮，为电梯添加任务，电梯将向目标楼层前进。
* 点击楼层内上下键，与外命令绑定，对应楼层右侧按钮亮起。
* 点击外命令左侧总上下按钮，可选择1-20层，并点击其上下按钮，选择合适的电梯为其添加任务，同时电梯的上下等待标志亮起。
* 点击外命令楼层对应上下按钮，选择合适的电梯为其添加任务，同时电梯的上下等待标志亮起。
* 点击报警键，该电梯任务清空，未完成的任务添加给其他电梯，禁用当前电梯按钮，停止当前电梯调度。
* 点击开关门键，开关电梯门（由于电梯状态每秒都在不断更新，为使其演示效果良好，设置了电梯门开启后在下次更新会关闭）。
* 上下状态标志、上下等待标志、门的状态、当前楼层数码版均为演示标志，不是可以点击的按钮。