进程管理 - 电梯调度_设计方案报告

操作系统第一次课程作业 - 电梯调度

目录

项目需求

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。

功能描述

1. 每个电梯里面设置必要功能键：如**数字键**、**关门键**、**开门键**、**上行键**、**下行键**、**报警键**、当前电梯的**楼层数**、**上升及下降状态**等。
2. 每层楼的每部电梯门口，应该有**上行和下行按钮**和当前**电梯状态的数码显示器。**
3. 五部电梯门口的**按钮是互联结的**，即当一个电梯按钮按下去时，其他电梯的相应按钮也就同时点亮，表示也按下去了。
4. 所有电梯初始状态都在第一层。每个电梯如果在它的上层或者下层没有相应请求情况下，则应该**在原地保持不动。**

开发环境

- **开发环境:** Windows 10
- **开发软件:**
    1. **PyCharm** *2019.1.1.PC-191.6605.12*
    2. **Qt Designer** *v5.11.2.0*
- **开发语言:** python3
- **主要引用块内容:**
    1. PyQt5 (QTimer, QtCore, QtGui, QtWidgets)
    2. pyqt5-tools
    3. threading

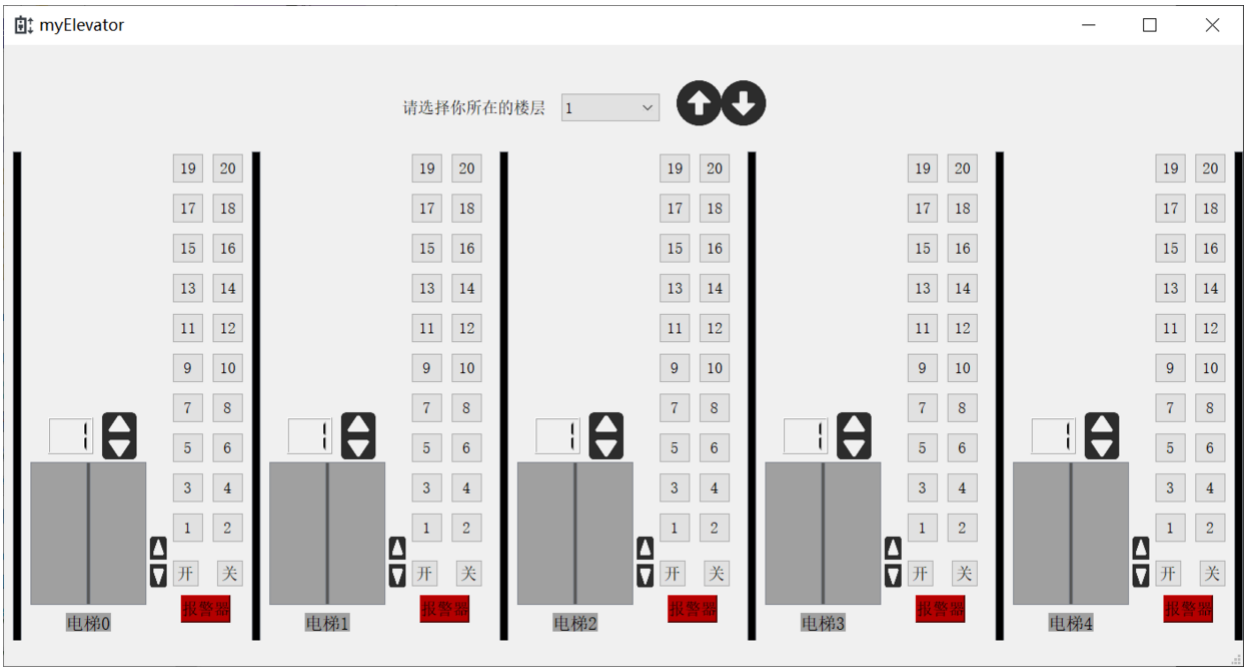
项目结构

```
│ myElevator.exe
│ README.md
│ 电梯调度_设计方案报告.md
│
├─Resources
│ ├─Button
│ │ doordown.png
│ │ doordown_hover.png
│ │ doordown_pressed.png
│ │ doorup.png
│ │ doorup_hover.png
│ │ doorup_pressed.png
│ │ down.png
│ │ down_hover.png
│ │ down_pressed.png
│ │ state.png
│ │ state_down.png
│ │ state_up.png
│ │ up.png
│ │ up_hover.png
│ │ up_pressed.png
```

```
| |
| ├─Figure
| | people.png
| |
| └─Icon
| elevator.ico
| icon.png
|
└─Src
dispatch.py
myElevator.py
myElevatorInterface.py
```

操作说明

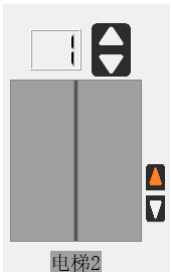
- 双击运行`myElevator.exe`, 进入电梯模拟系统如下图



- 点击每部电梯的**功能键**(*开/关键, 报警器, 楼层按钮*), 进行**单部电梯内命令处理**模拟

- 在上方**下拉框**中选择所在楼层, 并**点击上/下按钮**, 进行**多部电梯外命令处理**模拟

- **下拉框**中选择所在楼层, 并点击未被禁用**电梯的上/下行开关***(*互联结*), 进行**多部电梯外命令处理**模拟



系统分析

单部电梯内命令处理

- **内部事件:**
  1. 用户点击*楼层按钮*
  2. 用户点击*开/关门按钮*
  3. 用户点击*报警器*
- **预期响应:**
  1. 若按键楼层与当前楼层相同 => 该电梯开门(并等待用户自行关闭)
     - **若是在电梯运行过程瞬间点击该楼层, 则忽略该命令**
  2. 若按键楼层与当前楼层不同:
     1. 若电梯为静止状态 => 将该楼层信息加入消息队列中
     2. 若电梯忙碌:

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

- ## 多部电梯外命令处理
  - **外部事件:**
    1. 用户选择*所在楼层*
    2. 用户选择*上/下行方式*
    3. 用户可以点击任意没禁用*电梯的上/下行按钮* **(每部电梯的按钮是相互关联的)**
  - **预期响应:**
    1. 筛选可以响应的电梯 (没禁用的电梯)
    2. 可响应电梯中:
       1. 向上顺路:

          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 => "可调度性"定义为(用户楼层 - 该电梯当前楼层)
       2. 向下顺路:

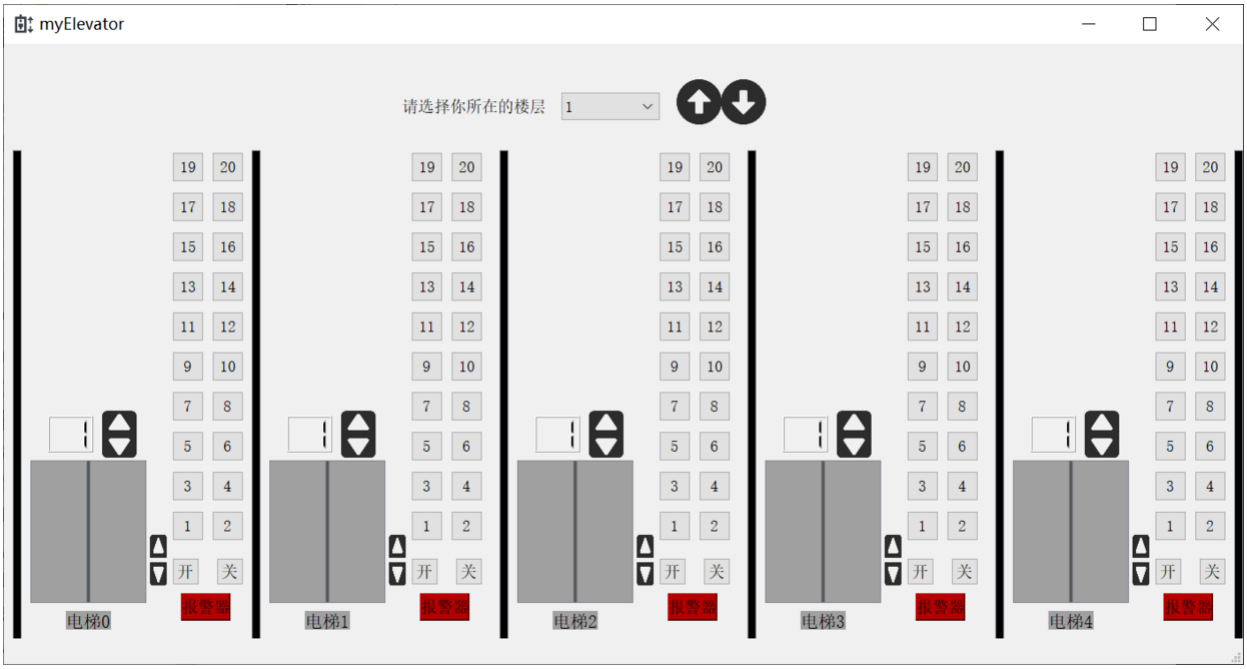
          若某电梯正在向下运动并且用户选择楼层小于该电梯当前楼层 => "可调度性"定义为(该电梯当前楼层 - 用户楼层)
       3. 某电梯静止:

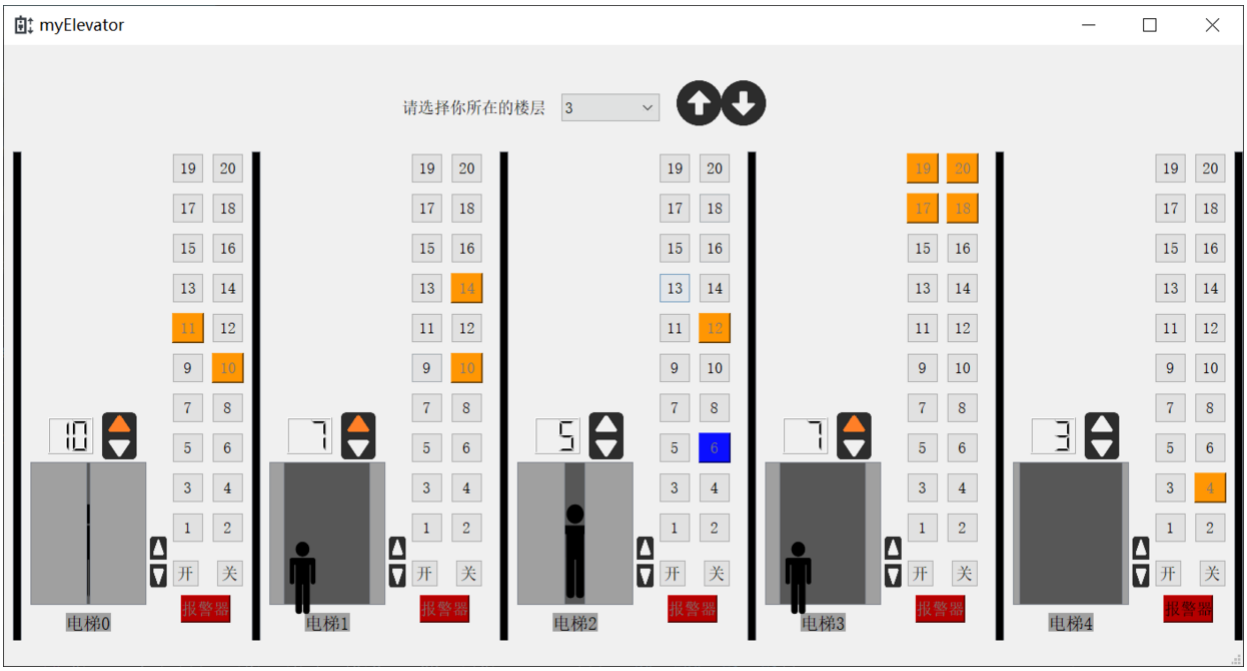
          若某电梯当前处于静止状态 => "可调度性"定义为(|该电梯当前楼层 - 用户楼层|)
    3. 选择可调度性最好的作为最佳电梯
       1. 若最佳电梯当前就在用户选择的楼层 => 该电梯开门(并等待用户自行关闭)
       2. 否则 => 将用户楼层信息加入该最佳电梯的消息队列中

系统设计

界面设计

1. **整体设计**





2. **墙模型**: *QtWidgets.QGraphicsView*
3. **电梯模型(包括电梯文字模型)**:
    1. 电梯背景 *QtWidgets.QGraphicsView*
    2. 前方两扇门 *QtWidgets.QGraphicsView*
    3. 将两扇门绑定动画 *QPropertyAnimation*
    4. 电梯文字 *QtWidgets.QLabel*
4. **电梯楼层数码管模型**: *QtWidgets.QLCDNumber*
5. **电梯上下行标志 & 门口按钮**: *QtWidgets.QGraphicsView* | *QtWidgets.QPushButton*
6. **报警器模型**: *QtWidgets.QPushButton*
7. **楼层按键模型**: *QtWidgets.QGridLayout* | *QtWidgets.QWidget* | *QtWidgets.QPushButton*
8. **开关键模型**: *QtWidgets.QPushButton*
9. **下拉框模型**: *QtWidgets.QComboBox* | *QtWidgets.QLabel* | *QtWidgets.QPushButton*

10. **小人模型**: *QtWidgets.QGraphicsView | QPropertyAnimation*

状态设计

1. **门状态**
   - `OPEN` = 0 # 开门状态
   - `CLOSED` = 1 # 关门状态
2. **电梯状态**
   - `STANDSTILL` = 0 # 静止状态
   - `RUNNING_UP` = 1 # 电梯上行状态
   - `RUNNING_DOWN` = 2 # 电梯下行状态
3. **动画状态**
   - `NOPE` = 0 # 空动画
   - `READYSTART` = 1 # 电梯即将运动
   - READYSTOP` = 2 # 电梯即将停止
4. **用户选择状态**
   - `GOUP` = 1 # 用户要上行
   - `GODOWN` = 2 # 用户要下行

类设计

1. **mywindow类:** 界面的显示

```
class mywindow(QtWidgets.QMainWindow, Ui_MainWindow):
    def __init__(self):
        super(mywindow, self).__init__()
        self.setupUi(self)
        self.setWindowTitle('myElevator')
        self.setWindowIcon(QIcon('Resources/Icon/icon.png'))
```

2. **Ui_MainWindow类:** 界面控件设置及其逻辑

```
class Ui_MainWindow(object):
        def __init__(self):
        self.Ctrl = Controler(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.animState = [NOPE] * 5  # 动画播放状态(空/即将运动/即将停止)标志位
        self.elevNow = [1] * 5  # 电梯楼层

        self.wall = []  # 墙模型
        self.elevator_back = []  # 电梯模型
        self.elevator_front = []
        self.elevator_Anim = []
        self.label = []
        self.lcdNumber = []  # 数码管模型
        self.stateshow = []  # 上下行标志模型
```

```
        self.updoorbtn = []  # 门口上下行按钮模型
        self.downdoorbtn = []
        self.warnbtn = []  # 报警器模型
        self.gridLayoutWidget = []  # 楼层按键模型
        self.gridLayout = []
        self.openbtn = []  # 开关键模型
        self.closebtn = []
        self.figure = []  # 小人模型
        self.figure_Anim = []

    def setupUi(self, MainWindow):
    def retranslateUi(self, MainWindow):

    # 报警器槽函数
    def warningClick(self):

    # 楼层按键槽函数
    def btnClick(self):

    # 外命令选择槽函数
    def chooseClick(self):

    # 开关门槽函数
    def doorClick(self):
```

3. **Controler类:** 用于电梯的控制及调度

```
class Controler(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(用列表代替队列)
        self.messQueue = []
        for i in range(0, 5):
            self.messQueue.append([])
        # 5个电梯内部不顺路消息列表
        self.messQueue_reverse = []
        for i in range(0, 5):
            self.messQueue_reverse.append([])

    # 警报器槽函数
    def warnCtrl(self, whichelev):

    # 开关门槽函数
    def doorCtrl(self, whichelev, whichcommand):

    # 开门动画
```

```
    def openDoor_Anim(self, whichelev):
    # 关门动画
    def closeDoor_Anim(self, whichelev):
    # 小人进电梯动画
    def figureIn_Anim(self, whichelev):
    # 小人出电梯动画
    def figureOut_Anim(self, whichelev):
    # 将门至于顶层
    def setDoorTop(self, whichelev):
    # 将小人至于顶层
    def setFigureTop(self, whichelev):

    # 内命令电梯运动
    def elevMove(self, whichelev, dest):

    # 外命令电梯调度
    def chooseCtrl(self, whichfloor, choice):

    # 更新电梯状态
    def updateElevState(self):
```

系统实现

内命令处理

1. **报警器**

   - 用户点击某报警器 => 更改其颜色(响应用户) => 弹出警告窗口"第i号电梯已损坏" => 调用控制器进行warnCtrl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动画
   - 如果所有5部电梯都禁用 => 设置下拉框等外命令空间禁用 => 弹出窗口"所有电梯以损坏"

```
# 报警器槽函数
def warningClick(self):
    which_warnbtn = int(self.sender().objectName()[-1])
    print("点击了{0}号报警器".format(which_warnbtn))
    self.warnbtn[which_warnbtn].setStyleSheet("background-color: rgb(255, 255,
255);")
    self.MessBox =
QtWidgets.QMessageBox.information(self.warnbtn[int(which_warnbtn)], "警告",  # 弹出
警告框
                                      "第" + str(which_warnbtn) +
"号电梯已损坏，不能继续使用")
    self.warnbtn[which_warnbtn].setStyleSheet("background-color: rgb(180, 0, 0);")

    self.Ctrl.warnCtrl(which_warnbtn)  # 调用控制器进行warnCtrl处理
```

```
# 警报器槽函数
def warnCtrl(self, whichelev):
    self.elev.elevEnabled[whichelev] = False  # 该电梯禁用
```

```
    # 省略电梯各部件禁用(详见Src中dispatch.py)

    # 五部电梯全部禁用
    arr = np.array(self.elev.elevEnabled)
    if ((arr == False).all()):
        self.elev.comboBox.setEnabled(False)  # 下拉框禁用
        self.elev.chooselabel.setEnabled(False)  # 文字禁用
        self.elev.upbtn.setEnabled(False)  # 上行按钮禁用
        self.elev.downbtn.setEnabled(False)  # 下行按钮禁用

        time.sleep(0.5)
        self.MessBox = QtWidgets.QMessageBox.information(self.elev, "警告", "所有电梯已损坏!")
```

2. 楼层按键

- 用户点击某个楼层按键 => 获取电梯编号 => 获取楼层信息 => 改变按钮背景颜色(模拟点击状态) => 将该按钮设置为不可点击状态 => 调用控制器进行elevMove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 => 打开门(并等待用户自行关闭)
  - 恢复按键背景并重新允许点击

```
# 楼层按键槽函数
def btnClick(self):
    whichbtn = self.sender()

    btn_name = whichbtn.objectName()
    buf = [int(s) for s in btn_name.split() if s.isdigit()]  # 提取字符串中的数字
    whichelev = buf[0]
    whichfloor = buf[1]
    print("{0}号电梯, {1}按键被按".format(whichelev, whichfloor))

    whichbtn.setStyleSheet("background-color: rgb(255, 150, 3);")  # 改变按钮背景颜色(模拟点击状态)
    whichbtn.setEnabled(False)  # 将该按钮设置为不可点击状态
    self.Ctrl.elevMove(whichelev, whichfloor)  # 调用控制器进行elevMove处理
```

```
# 内命令电梯运动
def elevMove(self, whichelev, dest):
```

```
    nowFloor = self.elev.elevNow[whichelev]  # 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[whichelev] == STANDSTILL:  # 电梯处于静止状态
            self.messQueue[whichelev].append(dest)  # 将目标楼层加入 消息队列

        else:
            if self.elev.elevState[whichelev] == RUNNING_UP:  # 电梯正在向上运行
                self.messQueue[whichelev].append(dest)  # 将目标楼层加入 消息队列并排序
                self.messQueue[whichelev].sort()
            elif self.elev.elevState[whichelev] == RUNNING_DOWN:  # 电梯正在向下运行
                self.messQueue_reverse[whichelev].append(dest)  # 将目标楼层加入 不顺路消息队列并排序
                self.messQueue_reverse[whichelev].sort()

    elif nowFloor > dest:
        if self.elev.elevState[whichelev] == STANDSTILL:
            self.messQueue[whichelev].append(dest)  # 将目标楼层加入 消息队列

        else:
            if self.elev.elevState[whichelev] == RUNNING_DOWN:
                self.messQueue[whichelev].append(dest)  # 将目标楼层加入 消息队列并反向排序
                self.messQueue[whichelev].sort()
                self.messQueue[whichelev].reverse()
            elif self.elev.elevState[whichelev] == RUNNING_UP:
                self.messQueue_reverse[whichelev].append(dest)  # 将目标楼层加入 不顺路消息队列并反向排序
                self.messQueue_reverse[whichelev].sort()
                self.messQueue_reverse[whichelev].reverse()

    else:  # 如果按键就为当前楼层
        if self.elev.elevState[whichelev] == STANDSTILL:  # 电梯静止 => 打开门(并等待用户自行关闭)
            self.elev.doorState[whichelev] = OPEN
            self.openDoor_Anim(whichelev)

        button = self.elev.findChild(QtWidgets.QPushButton,
                                     "button {0} {1}".format(whichelev, nowFloor))
 # 恢复按键背景并重新允许点击
        button.setStyleSheet("")
        button.setEnabled(True)
```

3. 开/关门

- 用户点击某个开/关门按钮 => 获取电梯编号 => 获取按键信息 => 调用控制器进行doorCtrl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 => 播放开门动画
- 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 => 将电梯门前的上下行开关恢复 => 播放关门动画

```
# 开关门槽函数
    def doorCtrl(self, whichelev, whichcommand):
        if whichcommand == 0:  # 如果用户要开门
            if self.elev.doorState[whichelev] == CLOSED and self.elev.elevState[
                whichelev] == STANDSTILL:  # 如果当前门是关闭状态并且电梯是静止的
                self.elev.doorState[whichelev] = OPEN  # 先将门状态更新为打开
                self.elev.elevEnabled[whichelev] = False

                self.openDoor_Anim(whichelev)

        else:  # 如果用户要关门
            if self.elev.doorState[whichelev] == OPEN and self.elev.elevState[
                whichelev] == STANDSTILL:  # 如果当前门是打开状态并且电梯是静止的
                self.elev.doorState[whichelev] = CLOSED  # 先将门状态更新为关闭
                self.elev.elevEnabled[whichelev] = True

                #将电梯门前的上下行按键熄灭
                for i in range(0, 5):
                    if self.elev.elevEnabled[i]:
                        self.elev.updoorbtn[i].setStyleSheet("QPushButton{border-
image: url(Resources/Button/doorup.png)}"
                                                             "QPushButton:hover{border-
image: url(Resources/Button/doorup_hover.png)}"

 "QPushButton:pressed{border-image: url(Resources/Button/doorup_pressed.png)}")

 self.elev.downdoorbtn[i].setStyleSheet("QPushButton{border-image:
url(Resources/Button/doordown.png)}"

 "QPushButton:hover{border-image: url(Resources/Button/doordown_hover.png)}"

 "QPushButton:pressed{border-image: url(Resources/Button/doordown_pressed.png)}")
                        self.elev.updoorbtn[i].setEnabled(True)
                        self.elev.downdoorbtn[i].setEnabled(True)
                self.closeDoor_Anim(whichelev)
```

外命令处理

1. **选择楼层及上下行**

   - 用户选择下拉框数字并点击上/下行开关 => 获取用户选择的楼层 => 获取上/下行并转换为**用户选择状态** => 调用控制器进行chooseCtrl处理

```
# 外命令选择槽函数
def chooseClick(self):
    whichfloor = int(self.comboBox.currentText())
    whichbtn = self.sender().objectName()

    if whichbtn[0] == 'd':
```

```
        if whichbtn != "downbtn":  # 如果是电梯门前的按钮
            for i in range(0, len(self.downdoorbtn)):
                if self.elevEnabled[i]:
                    self.downdoorbtn[i].setStyleSheet(
                        "QPushButton{border-image:
url(Resources/Button/doordown_pressed.png)}")
                    self.downdoorbtn[i].setEnabled(False)
        choice = GODOWN
    else:
        if whichbtn != "upbtn":  # 如果是电梯门前的按钮
            for i in range(0, len(self.downdoorbtn)):
                if self.elevEnabled[i]:
                    self.updoorbtn[i].setStyleSheet(
                        "QPushButton{border-image:
url(Resources/Button/doorup_pressed.png)}")
                    self.updoorbtn[i].setEnabled(False)

        choice = GOUP

    print("用户选择了 {0} {1}".format(whichfloor, choice))

    self.Ctrl.chooseCtrl(whichfloor, choice)  # 调用控制器进行chooseCtrl处理
```

2. 点击电梯上下行按钮

- 五部电梯门口的**按钮是互联结的** => 用户点击了某一个按钮 => 将所有没被禁用的按钮都设置为点击状态 => 设置为失效状态 => 调用控制器进行chooseCtrl处理

3. 电梯调度

- 初步筛选没损坏的电梯 => 计算每部可用电梯的"可调度性" => 选择可调度性最好的电梯作为最佳电梯
- 如果最佳电梯就在用户选择的楼层 => 打开门并等待用户自行关闭 => 播放开门动画 => 该电梯设置为禁用
- 否则 => 加入该最佳电梯的消息队列 => 将用户的目标楼层设定为特殊颜色
- 可调度性:
  - 向上顺路: 若某电梯正在`RUNNING_UP`状态并且用户选择楼层大于该电梯当前楼层 => "可调度性"定义为(用户楼层 - 该电梯当前楼层)
  - 向下顺路: 若某电梯正在`RUNNING_DOWN`状态并且用户选择楼层小于该电梯当前楼层 => "可调度性"定义为(该电梯当前楼层 - 用户楼层)
  - 某电梯静止:若某电梯当前处于`STANDSTILL`状态 => "可调度性"定义为(|该电梯当前楼层 - 用户楼层|)

```
# 外命令电梯调度
    def chooseCtrl(self, whichfloor, choice):

        # region 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(EnabledList)
        # endregion
```

```
        # region 计算每部可用电梯的"可调度性"
        dist = [INFINITE] * 5  # 可使用电梯距离用户的距离
        for EnabledElev in EnabledList:
            if self.elev.elevState[EnabledElev] == RUNNING_UP and choice == GOUP
and whichfloor > self.elev.elevNow[
                EnabledElev]:  # 向上顺路
                dist[EnabledElev] = whichfloor - self.elev.elevNow[EnabledElev]

            elif self.elev.elevState[EnabledElev] == RUNNING_DOWN and choice ==
GODOWN and whichfloor < \
                    self.elev.elevNow[EnabledElev]:  # 向下顺路
                dist[EnabledElev] = self.elev.elevNow[EnabledElev] - whichfloor

            elif self.elev.elevState[EnabledElev] == STANDSTILL:  # 该电梯此时静止
                dist[EnabledElev] = abs(self.elev.elevNow[EnabledElev] -
whichfloor)
        # endregion

        BestElev = dist.index(min(dist))  # 选择可调度性最好的电梯作为最佳电梯
        if dist[BestElev] == 0:  # 如果最佳电梯就在用户选择的楼层
            self.elev.doorState[BestElev] = OPEN  # 打开门并等待用户自行关闭
            self.openDoor_Anim(BestElev)

        else:
            self.messQueue[BestElev].append(whichfloor)  # 加入该最佳电梯的消息队列
            button = self.elev.findChild(QtWidgets.QPushButton,
                                         "button {0} {1}".format(BestElev,
whichfloor))  # 将用户的目标楼层设定为特殊颜色
            button.setStyleSheet("background-color: rgb(11, 15, 255);")
            button.setEnabled(False)
```

更新电梯状态

1. **定时器**
   - 采用定时器(1s调用一次方法更新电梯状态)

```
# 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)
```

2. **更新电梯状态**
   - 遍历五部电梯
   - 某个电梯的消息队列不为空
     - 如果电梯门是打开的 => 等待电梯关门
     - 电梯处于`STANDSTILL`状态 => 播放开门动画 => 播放小人进门动画 => 根据即将运行的方向更新电梯状态 => 动画变为`READYSTART`状态
     - 动画处于`READYSTART`状态 => 播放关门动画 => 动画变为`NOPE`状态

        - 动画处于就绪`READYSTOP`状态 => 结束该命令的处理 => 动画变为`NOPE`状态 => 电梯变为`STANDSTILL`状态
        - 除此之外 => 获取第一个目标楼层
            - 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 => 播放开门动画 => 播放小人出电梯动画 => 动画变为`READYSTOP`状态 => 将该楼层按钮恢复到原始状态
    - 某个电梯的消息队列为空 & 不顺路消息队列不为空 => 交换两个队列
    - 遍历五部电梯, 在运行过程中禁止点击报警键:
        - 如果这个电梯没被禁用
            - 如果电梯处于`STANDSTILL`状态 => 允许使用报警键
            - 否则 => 报警键禁用

```
# 更新电梯状态
def updateElevState(self):
    # print('timer clock......')

    for i in range(0, len(self.messQueue)):  # 遍历五部电梯
        if len(self.messQueue[i]):  # 某个电梯的消息队列不为空

            if self.elev.doorState[i] == OPEN:  # 如果电梯门是打开的 => 等待电梯关门
                continue

            elif self.elev.elevState[i] == STANDSTILL:  # 电梯处于静止状态
                self.openDoor_Anim(i)
                self.figureIn_Anim(i)

                if self.elev.elevNow[i] < self.messQueue[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.elevNow[i] > self.messQueue[i][0]:
                    self.elev.elevState[i] = RUNNING_DOWN

                self.elev.animState[i] = READYSTART  # 动画变为就绪运行状态

            elif self.elev.animState[i] == READYSTART:  # 动画处于就绪运行状态
                self.closeDoor_Anim(i)
                self.elev.animState[i] = NOPE  # 动画变为空状态

            elif self.elev.animState[i] == READYSTOP:  # 动画处于就绪停止状态
                self.messQueue[i].pop(0)  # 结束该命令的处理
                self.closeDoor_Anim(i)
                self.elev.animState[i] = NOPE        # 动画变为空状态
                self.elev.elevState[i] = STANDSTILL  # 电梯变为静止状态
                self.elev.stateshow[i].setStyleSheet("QGraphicsView{border-image: url(Resources/Button/state.png)}")
```

```
            else:
                destFloor = self.messQueue[i][0]  # 获取第一个目标楼层

                if self.elev.elevNow[i] < destFloor:  # 向上运动
                    self.elev.elevState[i] = RUNNING_UP
                    self.elev.stateshow[i].setStyleSheet(
                        "QGraphicsView{border-image:
url(Resources/Button/state_up.png)}")
                    self.elev.elevNow[i] = self.elev.elevNow[i] + 1  # 将当前楼层加一
并设置数码管显示
                    self.elev.lcdNumber[i].setProperty("value",
self.elev.elevNow[i])

                elif self.elev.elevNow[i] > destFloor:  # 向下运动
                    self.elev.elevState[i] = RUNNING_DOWN
                    self.elev.stateshow[i].setStyleSheet(
                        "QGraphicsView{border-image:
url(Resources/Button/state_down.png)}")
                    self.elev.elevNow[i] = self.elev.elevNow[i] - 1  # 将当前楼层减一
并设置数码管显示
                    self.elev.lcdNumber[i].setProperty("value",
self.elev.elevNow[i])

                else:  # 电梯到达目的地
                    self.openDoor_Anim(i)
                    self.figureOut_Anim(i)
                    self.elev.animState[i] = READYSTOP  # 到达目的地 => 动画变为就绪停
止状态

                    button = self.elev.findChild(QtWidgets.QPushButton,
                                                 "button {0} {1}".format(i,
self.elev.elevNow[i]))  # 恢复该按钮的状态
                    button.setStyleSheet("")
                    button.setEnabled(True)

        elif len(self.messQueue_reverse[i]):  # 如果消息队列为空 & 不顺路消息队列不为空
            self.messQueue[i] = self.messQueue_reverse[i].copy()  # 交替两个队列
            self.messQueue_reverse[i].clear()

    # 电梯在运行过程中禁止点击报警键
    for i in range(0, 5):
        if self.elev.gridLayoutWidget[i].isEnabled():  # 如果这个电梯没被禁用
            if self.elev.elevState[i] == STANDSTILL:  # 如果电梯是静止的
                self.elev.warnbtn[i].setEnabled(True)
            else:
                self.elev.warnbtn[i].setEnabled(False)
```

动画实现

1. **开门动画**

```
# 开门动画
def openDoor_Anim(self, whichelev):
    self.elev.elevator_Anim[2 * whichelev].setDirection(QAbstractAnimation.Forward)  #
正向设定动画
    self.elev.elevator_Anim[2 * whichelev +
1].setDirection(QAbstractAnimation.Forward)
    self.elev.elevator_Anim[2 * whichelev].start()  # 开始播放
    self.elev.elevator_Anim[2 * whichelev + 1].start()
```

2. **关门动画**

```
# 关门动画
def closeDoor_Anim(self, whichelev):
    self.elev.elevator_Anim[2 * whichelev].setDirection(QAbstractAnimation.Backward)
 # 反向设定动画
    self.elev.elevator_Anim[2 * whichelev +
1].setDirection(QAbstractAnimation.Backward)
    self.elev.elevator_Anim[2 * whichelev].start()  # 开始播放
    self.elev.elevator_Anim[2 * whichelev + 1].start()
```

3. **小人进电梯动画**

```
# 小人进电梯动画
def figureIn_Anim(self, whichelev):
    self.elev.figure[whichelev].setVisible(True)
    self.elev.figure_Anim[whichelev].setDirection(QAbstractAnimation.Forward)
    self.elev.figure_Anim[whichelev].start()

    s = threading.Timer(1.5, self.setDoorTop, (whichelev,))  # 1.5秒之后把门至于顶层
    s.start()
```

4. **小人出电梯动画**

```
# 小人出电梯动画
def figureOut_Anim(self, whichelev):
    self.elev.figure[whichelev].setVisible(True)
    self.elev.figure_Anim[whichelev].setDirection(QAbstractAnimation.Backward)
    self.elev.figure_Anim[whichelev].start()

    s = threading.Timer(1, self.setFigureTop, (whichelev,))  # 1s之后将人至于顶层
    s.start()
```

5. **将门置于顶层**

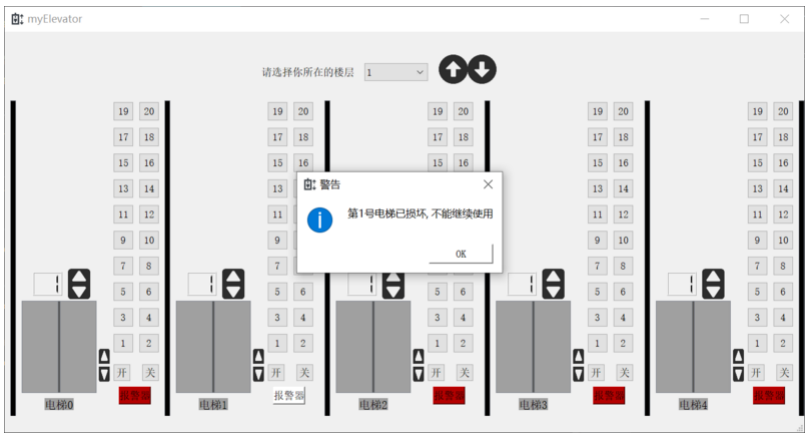
```
# 将门至于顶层
def setDoorTop(self, whichelev):
    self.elev.elevator_front[2 * whichelev].raise_()
    self.elev.elevator_front[2 * whichelev + 1].raise_()
```

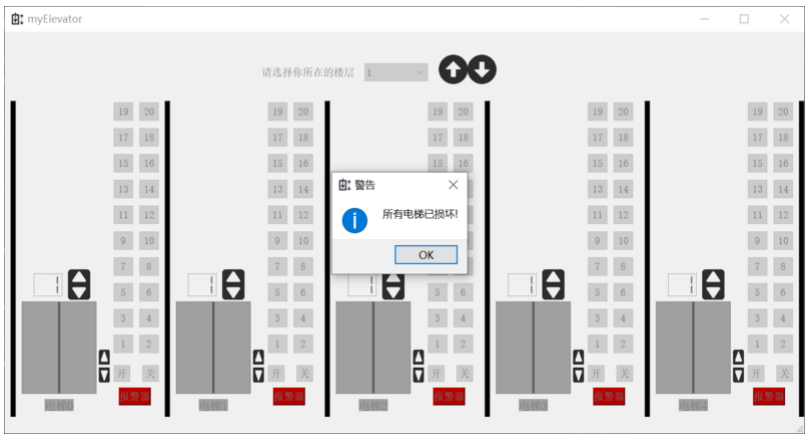
6. **将小人置于顶层**

```
# 将小人至于顶层
def setFigureTop(self, whichelev):
    self.elev.figure[whichelev].raise_()
    self.elev.figure[whichelev].setVisible(False)
```

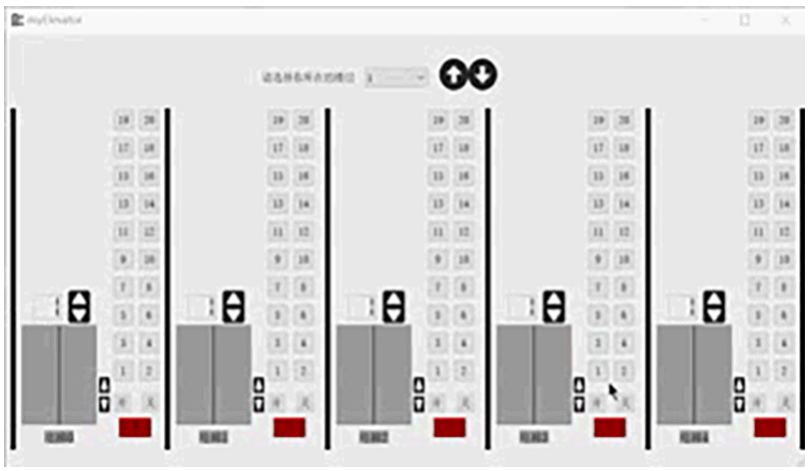
功能实现截屏展示

1. **报警器功能展示**

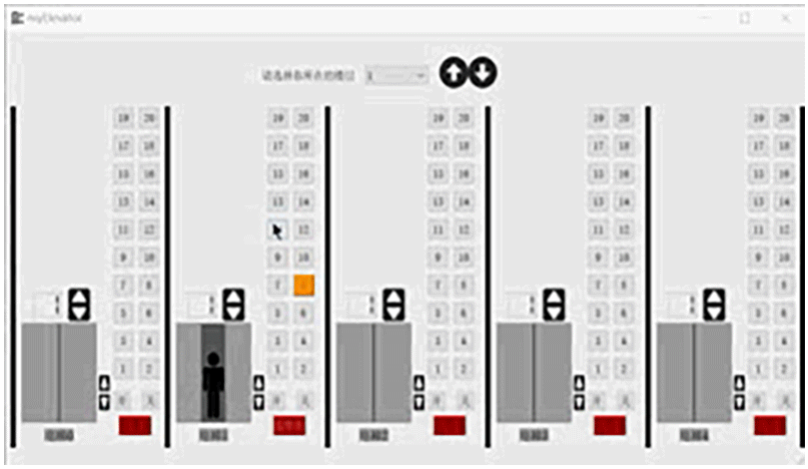




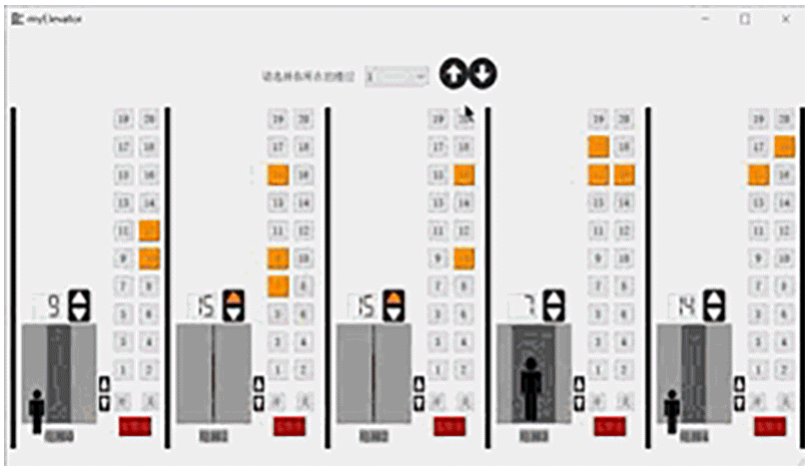
2. **开/关门功能展示**



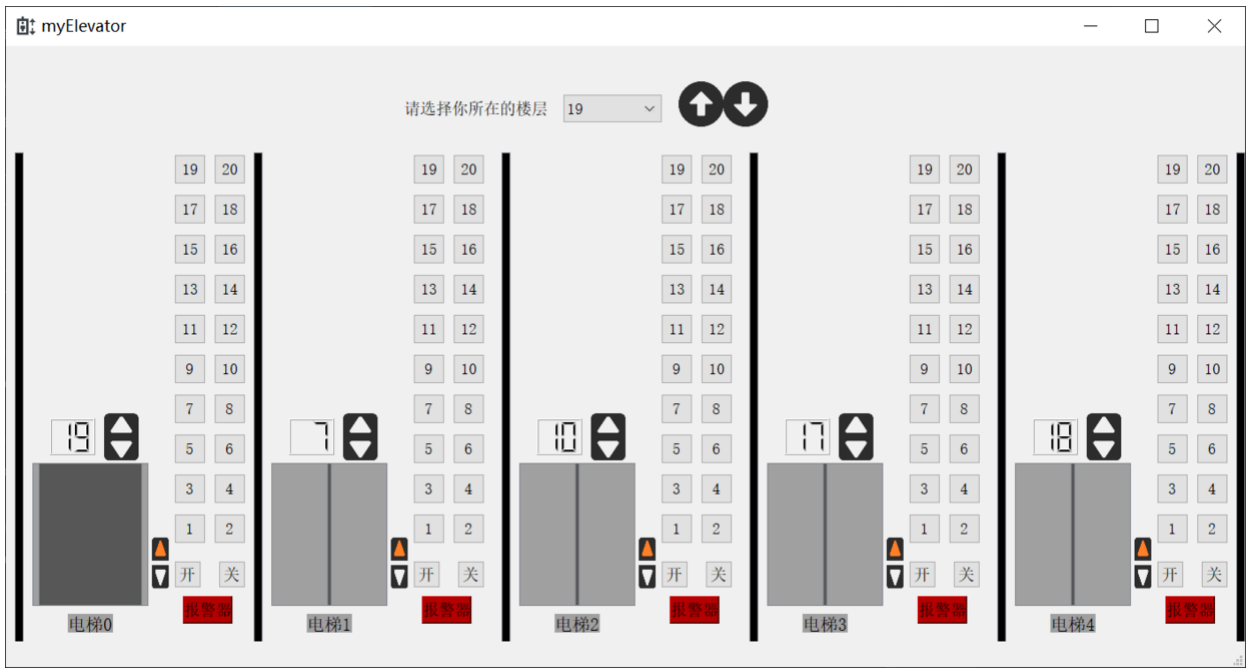
3. **内部指令处理与多线程**



4. **外部指令处理与电梯调度**



5. **上下行按钮联结**



作者

学号 1754060

姓名 张喆

指导老师 王冬青老师

上课时间 周三/周五 上午一二节

联系方式 *email:* doubleZ0108@gmail.com