项目说明文档

操作系统

——电梯调度

作 者 姓 名： 崔宸睿

学 号： 2152614

指 导 教 师： 张惠娟

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

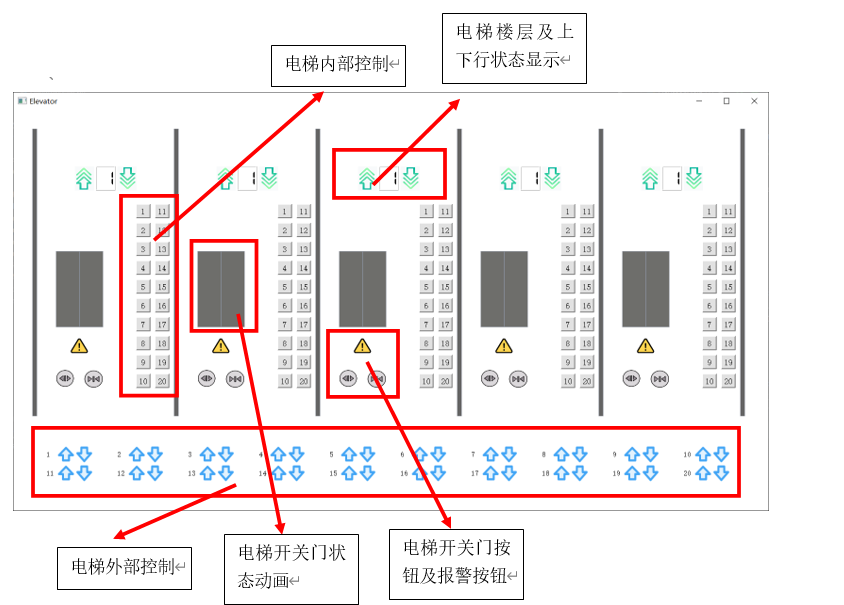
1 如何启动项目

本项目的开发环语言为python，使用PyCharm Community Edition 2022.3.3作为开发工具。

本项目需要安装依赖PyQt5，即需要在python环境中运行指令pip install PyQt5，安装依赖后运行ElevatorMain.py文件。

注：本项目也使用了Pyinstaller进行项目打包，生成的.exe文件名为ElevatorMain.exe.但效果并不好，所有静态图片资源都没有体现出来，所以仍然建议安装PyQt5依赖后运行ElevatorMain.py文件。

2 项目使用说明



电梯内部控制模块：点击对应楼层，电梯会前往该楼层（注：不要点击当前电梯所在的楼层）

电梯楼层及上下行状态显示模块：中间的数字实时显示当前电梯所在的楼层，两侧的箭头指示当前电梯的运行方向。

电梯外部控制模块：外部调度电梯，点击相应楼层后系统会自动选择一步电梯前往相应楼层。

电梯开关门状态动画：在合适的时机显示电梯开关门的动画，通过调用QtCore.QPropertyAnimation()实现

电梯开关门按钮及报警按钮：三个按钮中最上面的按钮是电梯禁用按钮，按下即使该电梯禁用，且不可被恢复，左侧是开门按钮，按下后电梯门开，右侧是关门按钮，按下后电梯门关。

在python console中出现的文字仅用于debug，不代表实际效果。

实际效果如视频所示：<实际效果.mp4>

3 项目文件结构及类设计

项目主要由三个.py文件组成：

|  |  |
| --- | --- |
| 文件名 | 作用 |
| ElevatorMain.py | 项目入口，启动整个项目 |
| Dispatcher.py | 封装了调度器类Dispatcher |
| ElevatorUI.py | 封装了电梯UI和控件的响应函数 |

下面具体介绍几个重要的类：

3.1 Dispatcher类

调度器类，在文件Dispatcher.py中定义。封装了各个电梯的调度状态以及在单击UI界面上的控件时触发的底层响应，调度算法被写在外部响应和内部响应模块中。

重要的类方法简要说明如下：

|  |  |
| --- | --- |
| 类方法名称 | 作用 |
| runTime | 计算某部电梯到达某目标楼层的运行时间 |
| externalCtrl | 外部控制函数，负责选择电梯响应外部控制的命令 |
| innerCtrl | 内部控制函数，负责响应内部控制命令 |
| doorCtrl | 门控制函数，负责响应开门关门命令并更新电梯状态 |
| warnCtrl | 警报控制函数，负责响应警报命令并禁用相关电梯 |
| updateState | 负责更新电梯状态，每一秒钟更新一次 |

**3.2 UI\_Window类**

界面窗口类，在文件ElevatorUI.py中定义，封装了UI的初始化操作及各类点击事件，启动函数中的Elevator类也是继承自UI\_Window类。重要的类方法说明如下：

|  |  |
| --- | --- |
| 类方法名称 | 作用 |
| setupUI | 初始化UI界面 |
| doorClick | 单击门控事件，绑定Dispatcher类中的doorCtrl函数 |
| externClick | 电梯外部控制事件，绑定Dispatcher类中的externalCtrl函数 |
| floorClick | 电梯内部调度事件，绑定Dispatcher类中的innerCtrl函数 |

4 调度算法设计

4.1 运行时间的求解：runTime函数

runTime函数有2个参数，分别是电梯序号elevNo和目标楼层targetFloor，目的是为了计算电梯序号为elevNo的电梯到第targetFloor层需要的时间。各种基本操作的时间已被定义在类中，运行通过一层时间RUN\_TIME = 1，等待接客时间WAIT\_TIME = 3。

若电梯处于静止状态，则运行时间就是当前楼层减去目标楼层的绝对值\*RUN\_TIME，若电梯处于运动状态，则需要记录电梯在到达目标楼层的路途中是否有其他请求，若有其他请求，则运行时间需要再加上WAIT\_TIME\*其他请求的个数。在电梯处于运动状态时还需要分顺行方向和逆行方向讨论，顺行方向下想要求得到达目标楼层的时间直接加上WAIT\_TIME\*其他请求的个数即可；若目标楼层处于电梯当前运行方向的逆行方向，则总时间=顺行方向到最后一层的时间+电梯顺行停止时间+顺行方向的最后一层到目标楼层的时间，代码逻辑如下：

def runTime(self, elevNo, targetFloor):

t = INF

if 电梯静止:

t = abs(self.curFloor[elevNo] - targetFloor) \* RUN\_TIME

else:

cnt = 0 //计数变量

if 目标楼层处于电梯顺行方向:

记录是否在发出命令的顺行方向还有其他请求

t = abs(self.curFloor[elevNo] - targetFloor) \* RUN\_TIME + cnt \* WAIT\_TIME

else:

t = abs(self.curFloor[elevNo] - FinalFloor) \* RUN\_TIME + len(self.comingList[elevNo]) \* WAIT\_TIME + abs(FinalFloor - targetFloor) //# 总时间=正向运行到最后一层的时间+电梯顺行停止时间+最后一层到目标楼层的时间

return t

**4.2 外部控制调度：externalCtrl函数**

externalCtrl函数有2个参数，分别是目标楼层targetFloor和外部命令方向direction。

在确定调度哪个电梯之前需要先确定各个可用的电梯到达目标楼层targetFloor的运行时间，这就需要调用runTime函数，然后选出运行时间最短的电梯，将其编号存储在变量minIndex中，代表需要将调度命令添加到电梯编号为minIndex的响应队列中。

接下来需要处理的就是要如何将该楼层加入到电梯的响应队列中。若该楼层位于电梯的顺行方向上，则需要加入电梯的正向运行队列comingList，此时按照电梯运行的方向分类讨论，若电梯下行，则电梯的正向运行队列需要以降序重排，因为此时楼层越高，电梯遇到的反而越早；若电梯上行，则需要升序重排，因为此时楼层越低，电梯遇到的越早。若该楼层位于电梯的逆行方向上或此时电梯恰好位于该楼层，则需要加入电梯的逆向运行队列reverseList。此时若电梯上行，则逆向队列需要降序重排，因为若是在逆向方向上，该楼层越高反而电梯遇到的越早；反之则升序重排。代码逻辑如下：

def externalCtrl(self, targetFloor, direction):

runningTime = []

for elevator in availableElev:

if 编号为elevator的电梯可用:

runningTime[elevator] = self.runTime(elevator, targetFloor)

global minIndex

minIndex = runningTime.index(min(runningTime))

if目标楼层处于电梯顺行方向:

向comingList[minIndex]中添加命令targetFloor

if 电梯上行:

comingList升序排序

else:

comingList降序排序

else:

向reverseList[minIndex]中添加命令targetFloor

if 电梯上行:

reverseList降序排序

else:

reverseList升序排序

**4.3 内部控制调度：innerCtrl函数**

innerCtrl函数有2个参数，分别是记录发出内部控制命令的电梯编号的参数elevNo和记录目标楼层的参数targetFloor。

innerCtrl函数的实现参考了look算法和scan算法，假设电梯沿当前方向向上或向下运行，当电梯在运行的方向上不再有请求或电梯到达当前运行方向终点时立即改变运行方向。这涉及到电梯状态的更新，具体实现将在类方法updateState中介绍。下面介绍内部调度算法的具体思路。

1.若目标楼层比当前楼层更高：若此时电梯为静止状态或上行状态，则将命令添加到正向列表comingList，然后将comingList升序重排；若此时电梯为下行状态，则将命令添加到逆向列表reverseList，然后将reverseList升序重排。

2.若目标楼层比当前楼层更低：若此时电梯为静止状态，则将命令添加到正向列表comingList并将其降序重排；若此时电梯上行，则将命令添加到逆向列表reverseList并将其降序重排；若此时电梯下行，则将命令添加到正向列表comingList，并将其降序重排。

3.若电梯当前就在目标楼层：若此时处于静止状态，直接开门并复原电梯内部按钮即可；若此时处于非静止状态，则应将该命令加入逆向列表。

代码逻辑如下：

def innerCtrl(self, elevNo, targetFloor):

if目标楼层更高:

if 电梯为静止或上行状态:

将目标楼层加入正向列表

正向列表升序重排

else: // 电梯下行

将目标楼层加入逆向列表

逆向列表升序重排

elif目标楼层更低:

if 电梯静止:

将目标楼层加入正向列表

正向列表降序重排

elif 电梯上行:

将目标楼层加入逆向列表

逆向列表降序重排

else: // 电梯下行

将目标楼层加入正向列表

正向列表降序重排

else: // 电梯当前处于目标楼层

if电梯处于静止状态:

执行开门动作

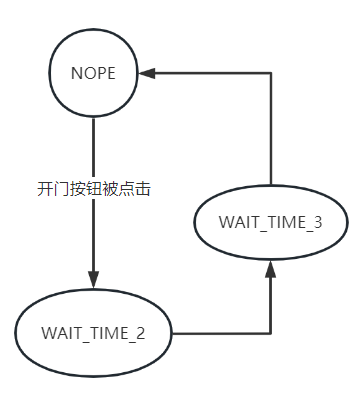
模拟电梯内部楼层按钮复原

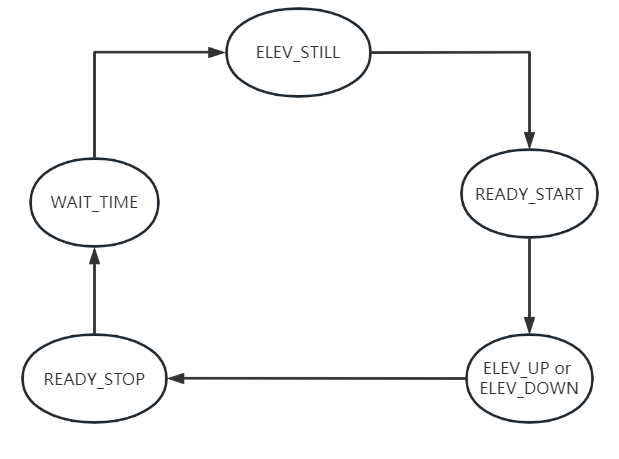
else: // 电梯处于非静止状态

将目标楼层加入逆向列表

**4.4 电梯状态更新：updateState函数**

该函数负责电梯各个状态之间的相互转化，并调用相关的动画效果函数，同时负责处理正向逆向队列中的调度任务。该函数通过计时器能够做到每1秒钟调用一次。电梯各种状态之间的转化如下：

 1.animationState:这个参量主要记录电梯动画状态，主要包括就绪运行状态READY\_START，就绪停止状态READY\_STOP，手动开门等待状态一WAIT\_TIME\_2，手动开门等待状态二WAIT\_TIME\_3以及空状态NOPE。WAIT\_TIME\_2和WAIT\_TIME\_3的设定主要为了方便手动开门能够等待时间更长。下图是开门按钮点击后的状态变化图

 2.elevState：这个参量主要记录电梯的运动状态，主要包括电梯上行ELEV\_UP，电梯下行ELEV\_DOWN，电梯静止ELEV\_STILL，结合就绪运行状态READY\_START，就绪运行结束状态READY\_STOP，和正常的等待开门接客时间WAIT\_TIME可得到电梯运行状态变化图为：

那么，我们为什么要设置READY\_STOP和READY\_START状态呢？主要是为了标识此刻电梯是否能展示动画效果。READY\_START状态标识此时电梯能进行开门，关门，上行，下行等动作，而READY\_STOP态表明电梯不能进行任何动作。同时，READY\_START状态的出现说明一个事务的最终结束，电梯由ELEV\_STILL（静止状态）开始向上下行状态过渡，并开始处理下一个事务，而READY\_STOP态的出现则说明电梯到达目标楼层，开始由上下行状态向WAIT\_TIME（等待上下客状态）和ELEV\_STILL（静止）的状态转换。

最后，当处理完顺行队列的事务后，需要处理逆行队列的事务。

函数的代码逻辑如下：

def updateState(self):

for each elevator:

if电梯不可用：:

continue

if self.animationState[elevNo] == WAIT\_TIME\_3:

电梯启用

关门

self.animationState[elevNo] = NOPE

continue

if self.animationState[elevNo] == WAIT\_TIME\_2:

self.animationState[elevNo] = WAIT\_TIME\_3

continue

if 开门按钮被点击:

self.animationState[elevNo] = WAIT\_TIME\_2

continue

// 处理命令队列

if 顺行队列不空:

if 电梯门开:

continue

// 静止状态，电梯转化为静止状态说明一些事务结束，开始处理下一个事务

if 电梯静止:

电梯开门

if 目标楼层大于当前楼层: // 电梯接下来要上行

self.elevState[elevNo] = ELEV\_UP

elif 目标楼层小于当前楼层: // 电梯接下来要下行

self.elevState[elevNo] = ELEV\_DOWN

else: //仍在目标楼层

self.elevState[elevNo] = ELEV\_UP

continue

self.animationState[elevNo] = READY\_START // 转为就绪态

// 处理ready\_start状态，ready\_start状态的出现说明一个事务的结束，关门并开始处理下一个事务

if self.animationState[elevNo] == READY\_START:

关门

self.animationState[elevNo] = NOPE // 动画状态置空

// ready\_stop状态->等待换客

if self.animationState[elevNo] == READY\_STOP:

self.animationState[elevNo] = WAIT\_TIME

continue

if self.animationState[elevNo] == WAIT\_TIME:

从顺行列表中弹出当前任务

状态回溯并置空

// 电梯可用，继续处理未完成的命令

elif self.warnState[elevNo] == AVAILABLE:

if目标楼层大于当前所处楼层:

self.elevState[elevNo] = ELEV\_UP

电梯上行

elif 目标楼层小于当前所处楼层

self.elevState[elevNo] = ELEV\_DOWN

电梯下行

else: //电梯到达目标楼层

相关状态置空

self.animationState[elevNo] = READY\_STOP # 运行-> 就绪停止

// 处理反向命令队列

elif 逆行列表非空: // 顺行列表处理结束，开始处理逆行队列

将逆行列表加入到顺行列表中

清空逆行列表