# 处理机管理 电梯调度

2253230 张正阳 2024/5/5

[处理机管理 电梯调度 1](#_Toc166099226)

[1. 项目目的 2](#_Toc166099227)

[2. 项目简述与功能 2](#_Toc166099228)

[3. 项目设计 2](#_Toc166099229)

[3.1 项目开发环境 2](#_Toc166099230)

[3.2 项目运行方式 2](#_Toc166099231)

[3.3 项目多线程 2](#_Toc166099232)

[3.4 项目整体设计 3](#_Toc166099233)

[4. 电梯调度算法 3](#_Toc166099234)

[5. 项目实现 4](#_Toc166099237)

[5.1 图形化界面 4](#_Toc166099238)

[5.2 类 4](#_Toc166099240)

[5.2.1 Elevator\_Window 类 4](#_Toc166099241)

[5.2.2 Elevator\_Thread 类 5](#_Toc166099242)

[5.3 函数 5](#_Toc166099243)

[5.3.1 Elevator\_Update 函数 5](#_Toc166099244)

[5.3.2 Set\_Elevator\_Goal\_Internal 函数 7](#_Toc166099245)

[5.3.3 Set\_Elevator\_Goal\_External\_Up 函数 7](#_Toc166099246)

[5.3.4 Set\_Elevator\_Goal\_External\_Down 函数 8](#_Toc166099247)

[5.3.5 Elevator\_Pause 函数 8](#_Toc166099248)

[5.3.6 Open\_Door 函数 8](#_Toc166099249)

[6. 运行界面展示 9](#_Toc166099250)

[7. 项目总结与心得 10](#_Toc166099251)

## 项目目的

* 学习调度算法
* 学习特定环境下多线程编程方法
* 通过实现电梯调度，体会操作系统调度过程

## 项目简述与功能

* 楼共有20层，其中安装了五部电梯。
* 每部电梯内部都设有数字键，乘客可以在电梯内部选择目标楼层。此外，电梯内部还设有显示屏，用于显示电梯当前所处的楼层以及运行状态（向上、向下或停止）。
* 每层楼的电梯外部都配有上行和下行按钮，乘客可以按下按钮来请求电梯。
* 当某一层楼的外部按钮被按下时，系统根据电梯调度算法自动选择最合适的电梯来响应请求。
* 每部电梯都配有故障暂停键，按下后电梯停止运行，直到故障修复后再按下重启键。
* 所有电梯的初始状态都是在第一层。如果某部电梯在其上层或下层没有相应的请求情况下，则会保持原地不动。
* 该程序基于线程的思想，使用了 QThread 实现了多线程编程，使得每部电梯都可以独立运行并响应乘客请求。

## 项目设计

### 项目开发环境

* 系统：Windows 11 家庭中文版
* IDE：Viusal Studio code 1.89.0 (User Setup)
* Python 解释器：通过 conda 部署 Python 环境，Python 版本为 3.8；
* 通过 pip安装 PyQt5

### 项目运行方式

* 直接运行：

已经通过 pyinstaller 生成了 Windows 上的可执行文件。在 Windows 系统上点击 My\_Elevator.exe，即可运行程序

**注 意 ， My\_Elevator.exe 一 定 要 与 background.png ， door\_close.jpg ， door\_half.jpg，door\_open.jpg 四张图片放在一个目录下。**

* 编译运行
  + Python 版本：python 3.8
  + 安装 PtQt5（pip install PyQt5）
* 运行源码：

进入源码所在目录，运行源码（python My\_Elevator.py）

### 项目多线程

在本项目中，使用了 PyQt5 库中的 QThread 来实现多线程编程，具体步骤如下：

* 创建自定义线程类：

定义一个继承自 QThread 的类，该类将作为线程的执行体。

在自定义线程类中重写 run() 方法，该方法包含了线程的实际执行逻辑。

* 连接信号和槽：

为了在线程执行期间与主线程进行通信，使用信号和槽机制。

定义自定义信号，并在适当的地方发射该信号。

* 创建线程对象：

在主程序中，创建自定义线程类的实例。

* 启动线程：

调用线程对象的 start() 方法来启动线程，使其开始执行 run() 方法中定义的逻辑。

* 处理线程的结束：

如果需要在线程完成后执行一些操作，可以连接 finished 信号到相应的槽函数，以处理线程结束时的清理工作或其他操作。

### 项目整体设计

在项目中，主要涉及以下几个部分：

* Elevator\_Window类：

负责定义和显示图形界面页面，包括电梯的内部和外部按钮、电梯状态显示等。

* Elevator\_Thread类：

为电梯运行的线程类，每个电梯都创建一个该线程的实例，实现多线程运行。

在该类中实现了电梯的运行逻辑，通过发射信号来更新电梯状态。

* Elevator\_Update函数：

负责更新电梯的运行状态，并对图形界面进行相应的更新显示。是控制电梯运行的关键函数，根据电梯当前状态、目标楼层等信息进行状态更新和图形界面的刷新。

* Set\_Elevator\_Goal\_Internal、Set\_Elevator\_Goal\_External\_Up、Set\_Elevator\_Goal\_External\_Down函数：

当外部按钮被按下时，根据调度算法选择响应该请求的电梯。分别处理内部按钮、外部上行按钮和外部下行按钮的点击事件，根据调度算法确定将请求分配给哪个电梯。

* 全局变量：

用于记录电梯运行的状态、待处理的请求等重要信息，以便在不同函数和线程中进行访问和更新。

## 电梯调度算法

本项目综合了多种算法思想，以平衡算法的复杂性、性能等因素。其算法思想和实现如下：

算法思想：

1. 当电梯外部有按钮被按下时，即时选择相应该请求的电梯并且加入该电梯的任务列表。

2. 电梯在一个方向上连续相应乘客请求，尽量少换方向，避免饥饿现象，同时使多部电梯同时运行，避免部分电梯长期闲置。

算法实现：

- 电梯内部按钮被按下，直接加入该电梯的任务列表。

- 电梯外部的按钮被按下时，按照下述优先原则选择相应请求的电梯：

1. 未运行的电梯

2. 正在运行，运行方向与请求方向一致、并且无需变换方向就能运行到请求楼层的电梯

3. 正在运行，运行方向与请求方向一致、但变换方向才能运行到请求楼层的电梯

4. 正在运行，运行方向与请求方向相反、且需要变换方向才能运行到请求楼层的电梯

5. 正在运行，运行方向与请求方向相反、且无需变换方向就能运行到请求楼层的电梯

算法合理性分析：

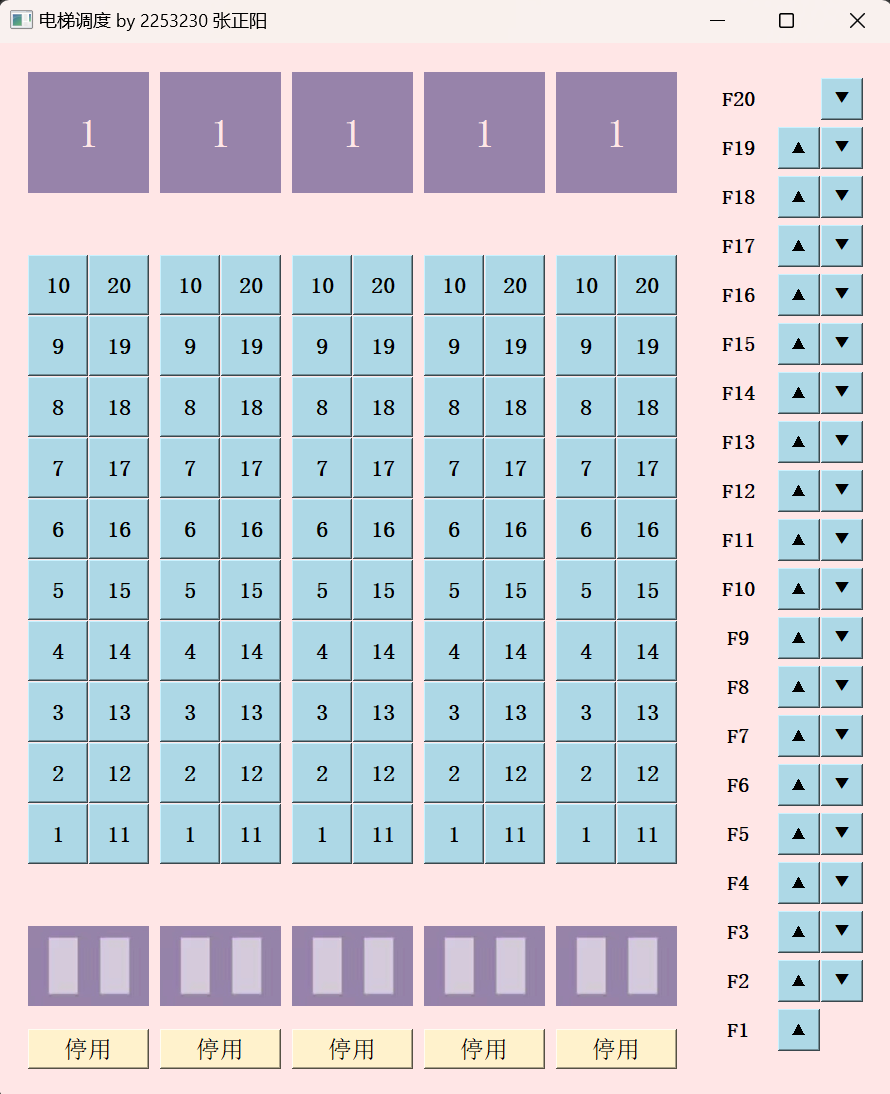
- 该算法考虑了多种情况，优先选择未运行的电梯，以尽快响应乘客请求。

- 在运行的电梯中，选择最理想的电梯，使得电梯能够顺路接乘客，减少额外行程。

- 对于需要变换方向的情况，考虑了电梯当前的运行状态和方向，选择合适的电梯响应请求。

## 项目实现

### 图形化界面

****

左侧部分的图标及功能如下所述：

1.楼层显示屏：位于左侧部分上方的紫色方形，用于显示电梯当前所处的楼层和运行状态（向上、向下、静止）。

2.楼层按钮：中间两个纵列的粉色按钮，代表电梯内部的楼层按钮。当乘客按下某个按钮时，表示选择了前往该楼层的目的地。按下后按钮会变成鹅黄色，表示目标楼层已选中。

3.故障按钮：位于底部的紫色按钮，上面标有“Pause”。按下该按钮后，电梯将停止运行，按钮状态变为“Start”。按下“Start”按钮后，电梯将恢复运行。

4.门状态显示：底部紫色块表示电梯门的状态。默认情况下，电梯门处于关闭状态。当电梯到达目标楼层时，将显示开门与关门的动画。

### 类

#### Elevator\_Window 类

这个类继承自 QWidget，是电梯调度系统的主窗口界面。它利用 PyQt 库来创建窗口和布局，并包含了电梯内部和外部的按钮以及一些标签和样式设置。

主要的布局管理器是whole\_layout`，它是一个水平布局，将窗口分为左右两个部分。左侧是电梯内部的布局管理器grid\_elevator\_internal`，右侧是电梯外部的布局管理器grid\_elevator\_external`。

在电梯内部的布局中，有一些 QLabel 用于显示当前楼层和电梯门状态，还有一些 QPushButton 用于表示电梯内部的楼层按钮和暂停按钮。这些按钮和标签都通过grid\_elevator\_internal布局管理器进行布局。

在电梯外部的布局中，有一些 QLabel 用于显示楼层号，以及一些 QPushButton 用于表示电梯外部的上行和下行按钮。这些按钮和标签都通过grid\_elevator\_external布局管理器进行布局。

最后，设置了窗口的样式、标题和位置，并显示窗口。特别说明的是，此类中也设置了各个按钮点击事件与相应处理函数的连接。

#### Elevator\_Thread 类

这个类继承自 QThread，用于在后台运行电梯的状态更新和门的开关动画。在这个类中，定义了两个信号update\_signal和open\_signal，分别用于更新电梯状态和控制门的开关动画。

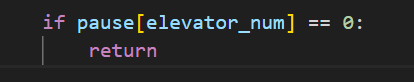
在run方法中，通过一个无限循环来不断发送update\_signal信号来更新电梯的状态。然后，使用time.sleep方法暂停 1 秒钟，模拟电梯状态的更新间隔。

如果open\_door[self.int - 1]为 1，表示该电梯需要打开门。在这种情况下，通过发送open\_signal信号来控制门的开关动画。

### 函数

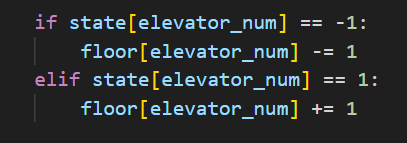
#### Elevator\_Update 函数

Elevator\_Update 函数用于**更新电梯的状态、所在楼层、任务列表**并对图形页面做相应更改。按照下述顺序进行：

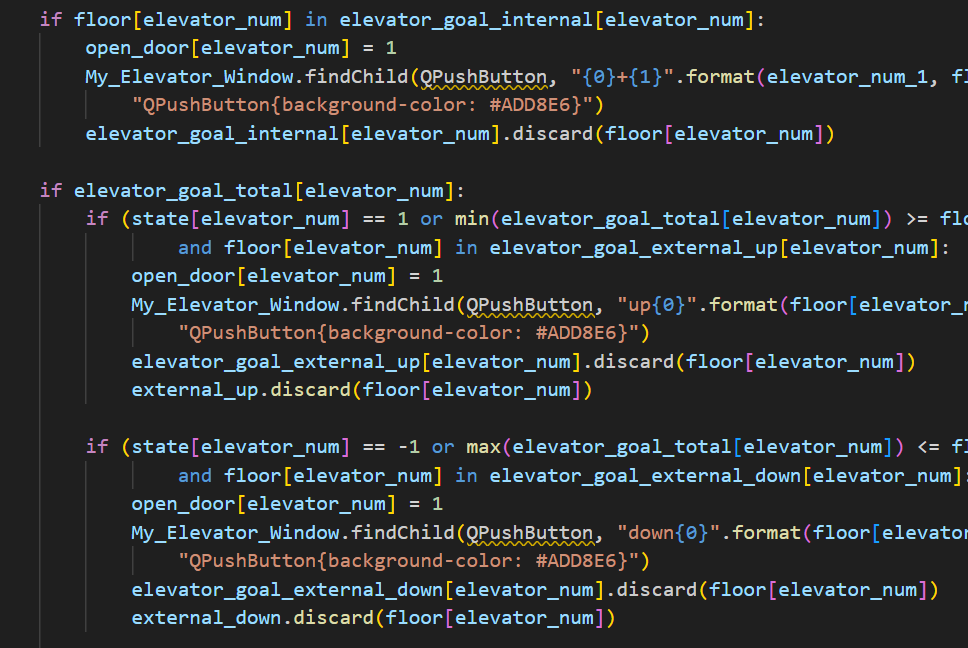


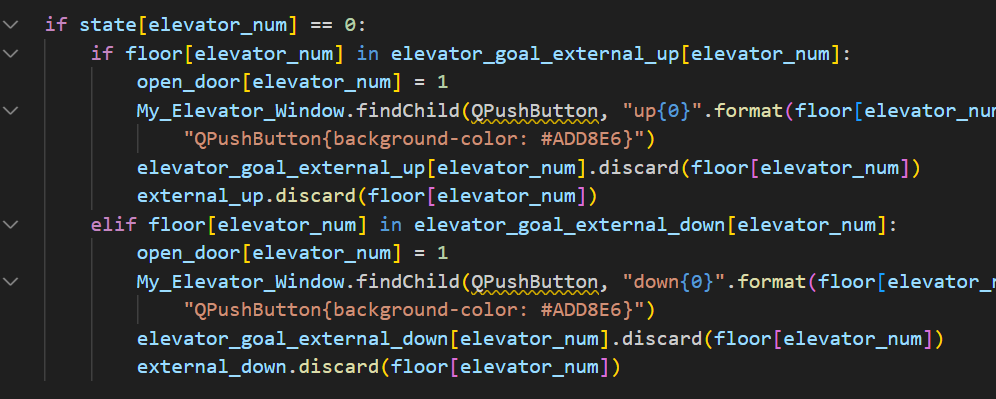
① 首先检查电梯是否被暂停，如果被暂停，则不需要更新状态，直接返回。

➁ 然后根据电梯的运行状态 state，更新电梯所在楼层 floor。

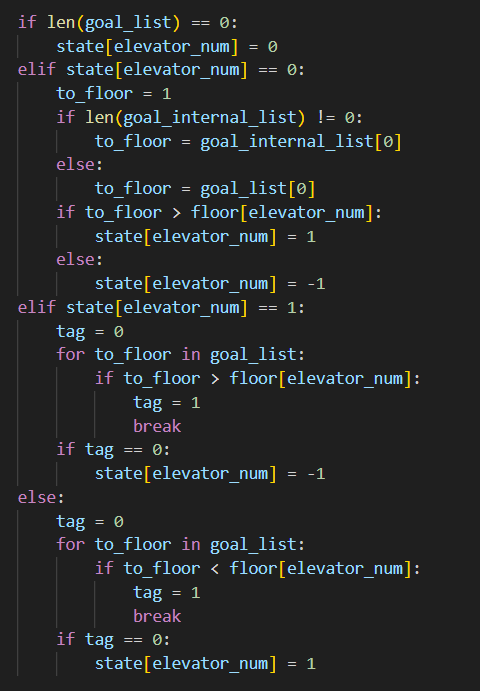


③ 接下来是处理当前需要开门的情况：如过当前楼层位于电梯的任务列表，且运行方向与任务请求方向不矛盾，则开门，并且更新任务列表（取🎧已完成的任务）。

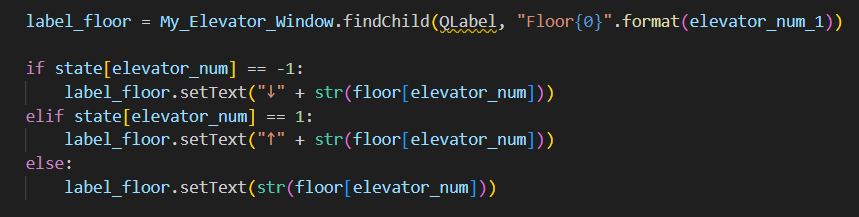




④根据当前状态和目标楼层列表的情况更新电梯的状态：如果当前运行方向上还有目标楼层，则不改变状态；否则，若反方向上有目标楼层，则改变运行方向；若两个方向都没有目标楼层，则置 state 为 0，表示停止。



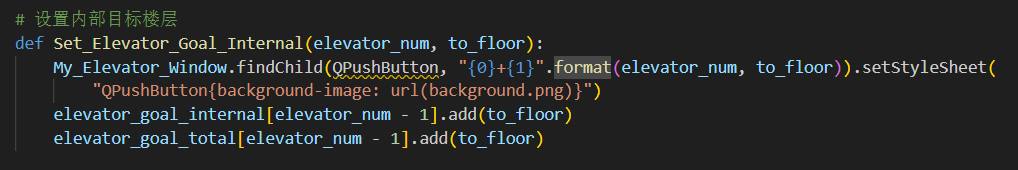
⑤ 最后，根据更新后的电梯状态和电梯楼层，更新电梯内部的显示屏内容。



#### Set\_Elevator\_Goal\_Internal 函数

电梯内部按下按钮后，调用 Set\_Elevator\_Goal\_Internal 函数来更新电梯的任务列表。函数接受两个参数：elevator\_num 表示电梯的编号，to\_floor 表示要设定的目标楼层。函数首先使用 findChild 方法找到对应电梯和目标楼层的按钮，并将按钮的样式修

改为设定目标楼层的状态。然后，将目标楼层添加到电梯的内部目标楼层集合和总目标楼层集合中。



#### Set\_Elevator\_Goal\_External\_Up 函数

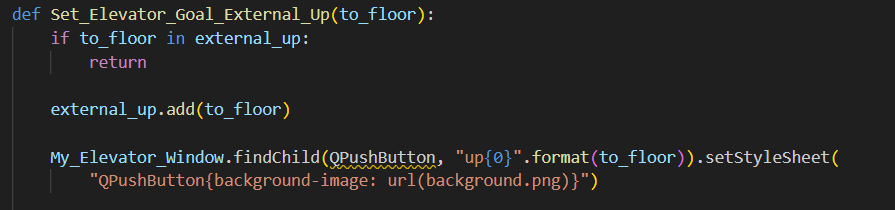
电梯外部按下上楼的按钮后，调用 Set\_Elevator\_Goal\_External\_Up 函数，根据电梯调度算法选择响应该请求的电梯并且更新该电梯的任务列表。

函数接受一个参数：to\_floor 表示外部上楼请求所在的楼层。

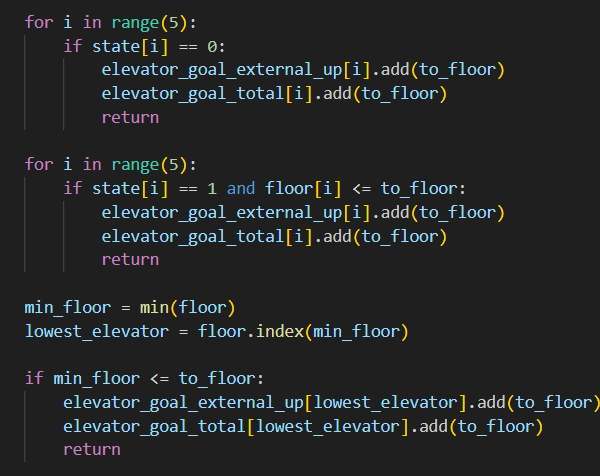
① 检查是否已经存在相同的外部上楼请求，如果存在则直接返回。

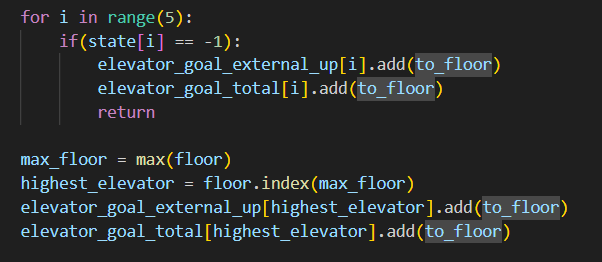
➁ 将外部上楼请求楼层添加到外部上楼请求集合 external\_up 中。

③ 使用 findChild 方法找到对应楼层的上行按钮，并将按钮的样式修改为表示该楼层有外部上楼请求的状态。



④ 根据前面讨论过的电梯调度算法，会选择一个电梯来响应该外部上楼请求。





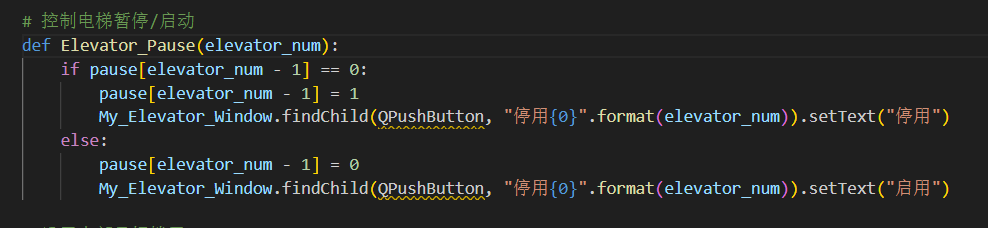
#### Set\_Elevator\_Goal\_External\_Down 函数

电梯外部按下下楼的按钮后，调用 Set\_Elevator\_Goal\_External\_Down 函数，根据电梯调度算法选择响应该请求的电梯并且更新该电梯的任务列表。

函数内容与 Set\_Elevator\_Goal\_External\_Up 函数一致，此处不再赘述。

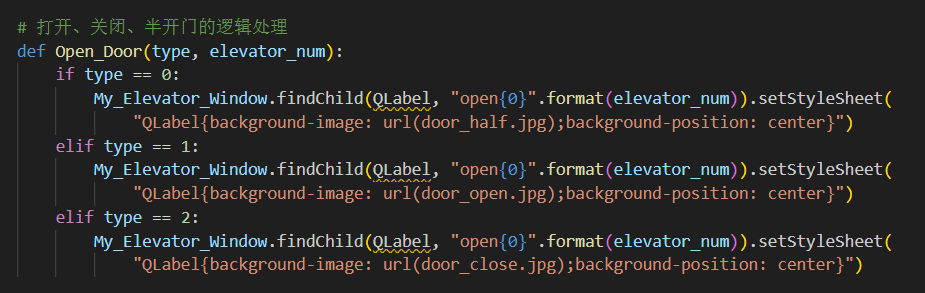
#### Elevator\_Pause 函数

当图形界面的”Pause”或”Start”按钮被按下后，会调用 Elevator\_Pause 函数来更新电梯的故障情况。

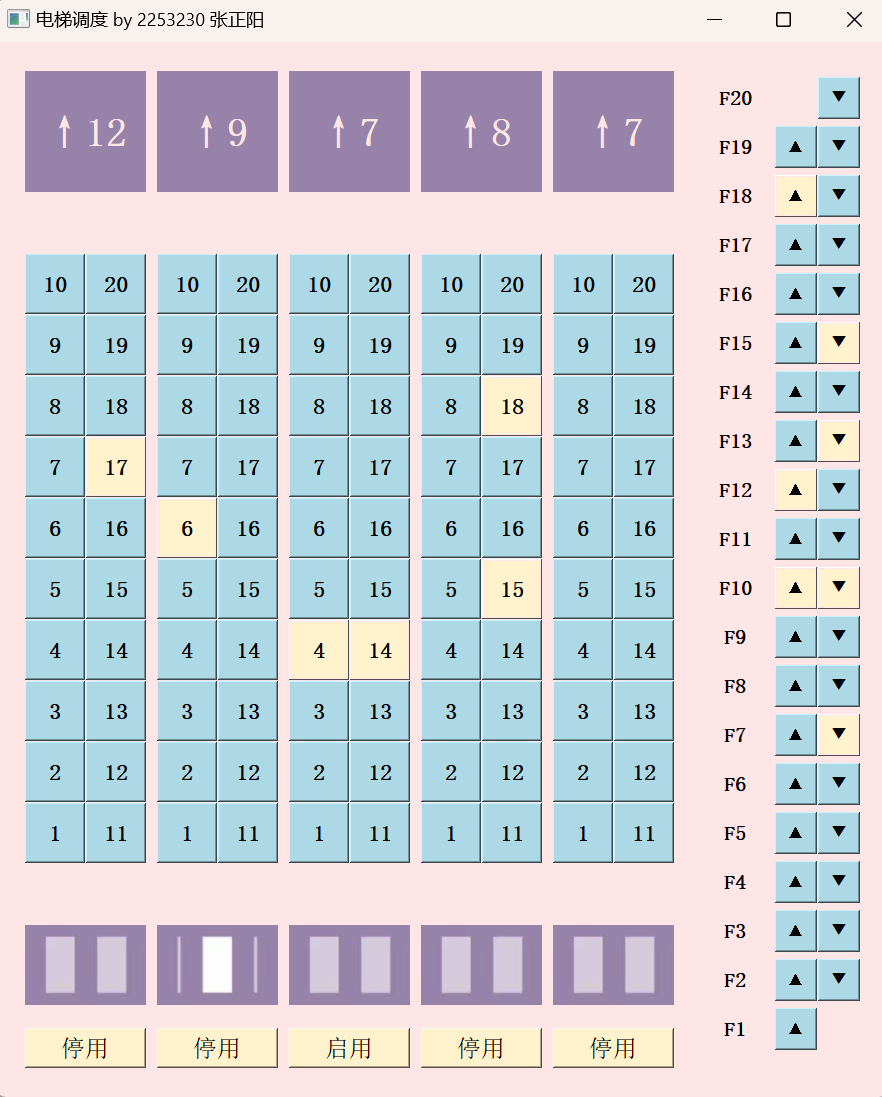


#### Open\_Door 函数

该函数根据不同的参数显示电梯门不同的状态图片，以实现电梯门开关的动画效果。



## 运行界面展示

****

## 项目总结与心得

拿到这个项目后，我的第一步是调研不同的电梯调度算法、各种语言以及框架的实现。考虑到 PyQt5 将 Qt 的功能与 Python 的灵活性和易用性结合在一起，为开发者提供了一种创建功能丰富、跨平台的 GUI 应用程序的便捷方式，我最终决定以 Python 为编程语言，并以 PyQt5 库为框架来完成。对于我这样没有写过 Python、没有用过任何框架写过前后端的人来说，这是一项极具挑战性的任务，因为我要从零开始学习。

在这个项目的推动下，我在几天的时间内入门了 Python 和 PyQt5，并且完成了项目。虽然还存在许多不足和待改进之处，但我对自己的成果感到满意。

在完成项目的过程中，我遇到了一个最困扰我最久的问题，那就是界面中图标闪烁的问题。在尝试解决这个问题的过程中，我走了很多弯路，但也收获了很多。

一开始我认为是多线程同时修改图形界面导致的冲突，所以尝试在修改页面之前加锁。在反复修改的过程中，我对锁这一机制有了更深入的理解。

最终解决问题的方法是：在线程执行期间使用信号与槽机制与主线程进行通信，通过发送信号在主线程中修改图形界面，来避免在多个线程中同时修改界面的情况，从而解决了图标闪烁甚至消失的问题。通过这个过程，我认识到在多线程编程中，应该尽量避免在多个线程中同时修改 GUI。

此项目的多线程编程实践加深了我对线程这一概念的理解，也让我觉得自己不再是空谈理论。总的来说，我收获颇丰。