项目说明文档

处理机管理项目

——电梯调度

作 者 姓 名： 毛凌骏

学 号： 2053058

指 导 教 师： 张慧娟

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

目录

[1 需求 1](#_Toc9930)

[1.1 项目背景 1](#_Toc13130)

[1.2 功能分析 1](#_Toc7701)

[2 设计 2](#_Toc22402)

[2.1 界面设计 2](#_Toc32189)

[2.1.1 电梯内部视角 2](#_Toc5046)

[2.1.2 楼层外部视角 3](#_Toc30526)

[2.2 算法设计 4](#_Toc24057)

[2.2.1 计算优先级 4](#_Toc12483)

[2.2.2 分发请求 5](#_Toc12965)

[2.2.3 更新电梯状态 5](#_Toc20649)

[2.2.4 故障处理 5](#_Toc5198)

[3 实现 6](#_Toc5693)

[3.1 可视化界面实现 6](#_Toc30728)

[3.2 调度算法实现 6](#_Toc29010)

[3.2.1 优先级计算 6](#_Toc26913)

[3.2.2 电梯调度 7](#_Toc22434)

[3.2.3 电梯状态更新 7](#_Toc18309)

[3.2.3 电梯损坏与修理 7](#_Toc19991)

[4 测试 8](#_Toc24162)

[4.1 电梯调度测试 8](#_Toc3433)

[4.2 开关门按钮测试 10](#_Toc15561)

[4.3 报警按钮测试 12](#_Toc9407)

# 1 需求

## 1.1 项目背景

在大型多层建筑中，电梯是不可或缺的设施之一。由于对电梯的请求需要随时得到响应，电梯的调度和运行成为了一个具有挑战性的问题。在传统的电梯系统中，通常是按照楼层顺序依次响应请求，导致电梯在高峰期运行效率低下。因此，设计一个高效的电梯调度算法成为了一个重要的研究方向。

为了解决此类问题，本项目模拟了一个五部电梯相互联结的系统，每个电梯都能够响应楼层按钮和内部按钮的请求。当某个楼层按钮按下时，系统将自动调用优先级最高的电梯前往该楼层。可以发现，电梯调度算法与处理机调度算法有大量相似之处，具体如下：

1. 都需要进行任务分配和调度，以最优化完成任务或处理任务。
2. 任务或请求都需要按照一定的顺序或优先级来进行处理或完成。
3. 都需要考虑资源的利用效率和时间的优化，以提高系统的整体性能。
4. 在任务或请求量大的情况下，都需要进行优化算法的设计，以提高任务或请求的处理效率和系统的吞吐量。
5. 都需要进行状态的监测和调整，以保证系统的正常运行和资源的充分利用。
6. 在多任务或多请求的情况下，都需要进行并发控制和资源的分配与协调，以保证系统的稳定性和安全性。

基于电梯调度算法和处理机调度算法的相似性，该项目对于计算机处理机调度问题也有很大的意义。

## 1.2 功能分析

【基本任务】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。

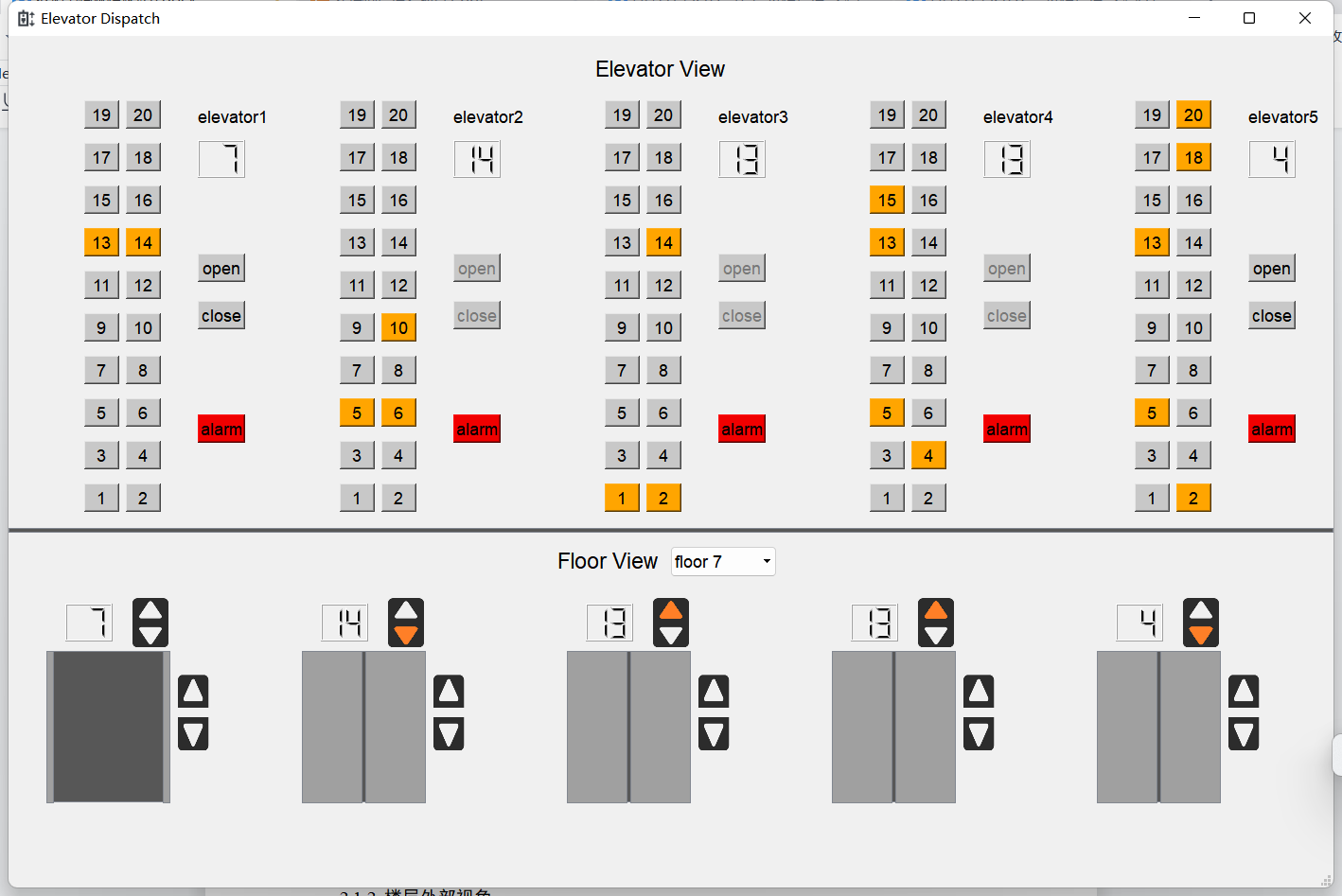
【功能描述】电梯应有一些按键，如：数字键、关门键、开门键、上行键、下行键、报警键等；有数码显示器指示当前电梯状态；每层楼、每部电梯门口，有上行、下行按钮、数码显示；五部电梯相互联结，即当一个电梯按钮按下去时，其它电梯相应按钮同时点亮，表示也按下去了。

【电梯调度算法】所有电梯初始状态都在第一层；每个电梯没有相应请求情况下，则应该在原地保持不动；电梯调度算法自行设计。

# 2 设计

## 2.1 界面设计

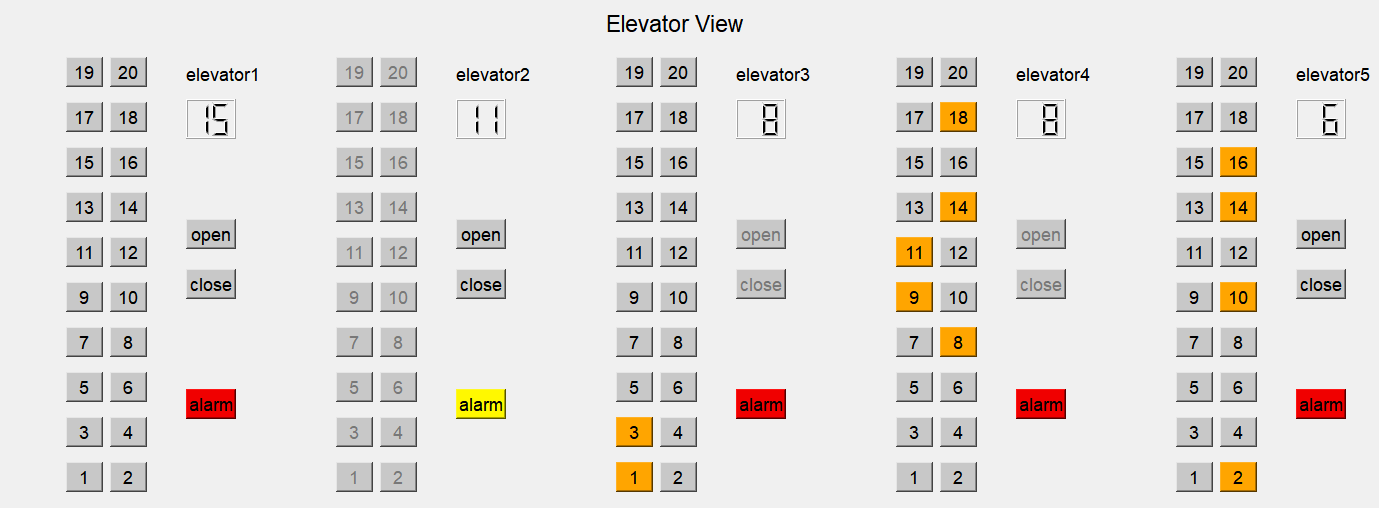
本项目采用python为主要开发语言，使用PyQt5图形库实现可视化界面。为了清晰表现电梯调度的过程，本项目将整个画面分割为上下两个部分，分别表示电梯内部视角与楼层外部视角。



### 2.1.1 电梯内部视角

该界面模拟电梯的内部场景，实时显示每个电梯的状态与所在楼层。包含的控件有楼层按钮、LED显示屏、开关门按钮和警报按钮。

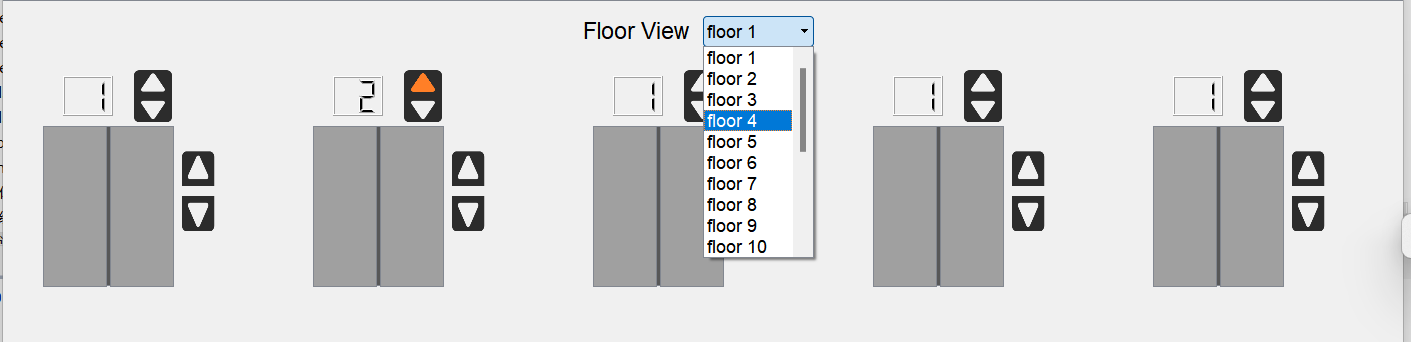
当按下某一楼层按钮后，该按钮会点亮，直至该层到达后熄灭，电梯移动过程中LED显示屏实时显示电梯所在楼层数。当电梯正在上下移动时，电梯的开关门按钮将被禁用以保证乘客安全，直至电梯停下时才会重新激活。电梯警报按钮按下后意味着该电梯已经损坏，为了简化场景，系统设定在一段固定时间后损坏的电梯将被维修并重新恢复正常。

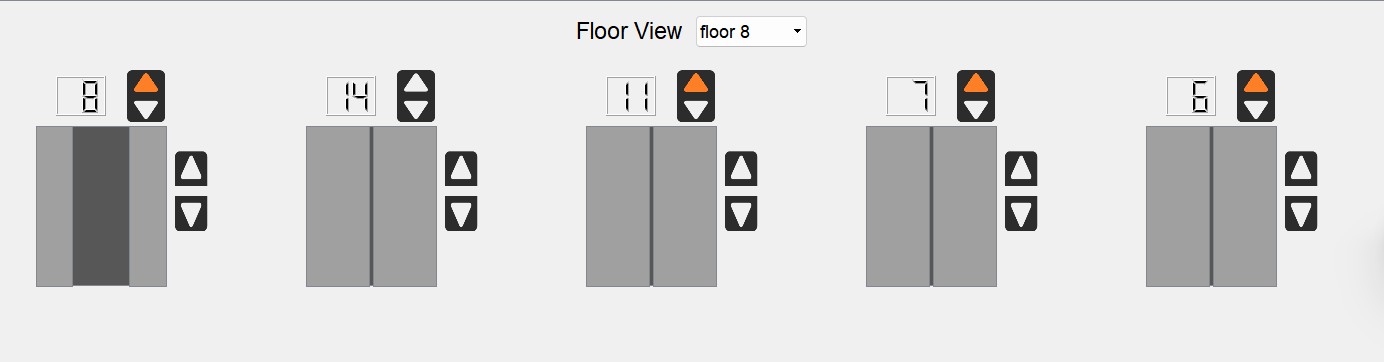


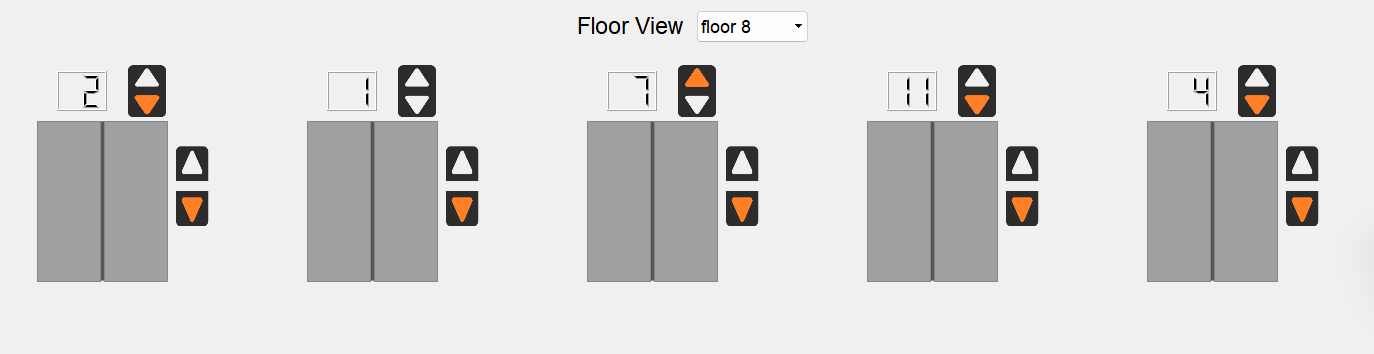
### 2.1.2 楼层外部视角

该界面模拟楼层外部场景，用户可以选择楼层，选择后该界面将展示该楼层的状态，包括是否有电梯开关门、每个电梯所在楼层以及上下情况等。该界面包含控件有电梯楼层LED显示屏、电梯上下行状态显示屏、上下按钮、电梯门与电梯。

当电梯上行时电梯状态显示屏会显示向上的箭头，下行则显示向下的箭头。对于该楼层的上下按钮，五个按钮相互联结，即当一个电梯按钮按下去时，其它电梯相应按钮同时点亮，当电梯到达即请求满足后五个按钮均会熄灭。当某个电梯到达该楼层后，画面中的电梯门将会打开，用户也可使用电梯内部的开关门按钮人工操作。







## 2.2 算法设计

本项目通过电梯和控制器类实现电梯调度。其中Elevator类表示每部电梯，包含其状态、楼层、目标等信息。Controller类管理所有的电梯并负责电梯的调度，负责计算优先级、调度电梯等操作。本电梯调度算法的设计思路如下：

### 2.2.1 计算优先级

为了选择最适合响应请求的电梯，需要计算每部电梯的优先级。这里根据电梯的运动方向、距离请求楼层的距离以及电梯是否损坏等因素来计算优先级。损坏的电梯具有最低优先级。

在计算电梯优先级时，我们考虑了以下因素来为每部电梯分配一个优先级得分，并分别赋予不同的得分值：

|  |  |  |
| --- | --- | --- |
| 优先级得分 | 条件 | 说明 |
| -1 | 电梯损坏 | 电梯暂时无法响应请求。 |
| -2 | 电梯运动方向与请求方向相反 | 将请求楼层加入电梯的待实现目标数组prior。 |
| -4 | 同一楼层已有相反方向的请求 | 该电梯已经被安排在相反方向的请求中，考虑其他电梯来响应当前请求。 |
| 20-N | 距离请求楼层的距离为N层 (N  为正整数) | 距离越近，优先级得分越高。得分最大值为20，根据距离的远近递减。 |

在这个优先级计算方法中，我们引入了两个数组：目标数组goal和待实现目标数组prior。目标数组goal存储了电梯需要按顺序到达的楼层，而待实现目标数组prior存储了电梯反方向的所有目标，即完成目标数组中的任务后需要到达的楼层。

当一个新的请求到达时，我们根据以上四个因素计算所有电梯的优先级得分。首先，检查电梯是否损坏，如果损坏则得分为-1。然后，根据电梯的运动方向、距离请求楼层的距离以及电梯目标楼层队列来调整优先级得分。最后，选择优先级得分最高的电梯来响应请求。

这种优先级计算方法充分考虑了电梯的运动状态、楼层距离以及目标楼层队列，从而使得调度算法能够在保证响应速度的同时，实现电梯资源的合理分配和高效利用。同时，引入目标数组goal和待实现目标数组prior使得调度算法可以灵活地处理不同方向的请求，提高了系统的调度效率。

### 2.2.2 分发请求

当有楼层按钮被按下时，Controller类会计算每部电梯的优先级并选择最适合响应请求的电梯。分发请求的过程中，根据请求的方向（上/下）来决定哪些电梯具有较高优先级。

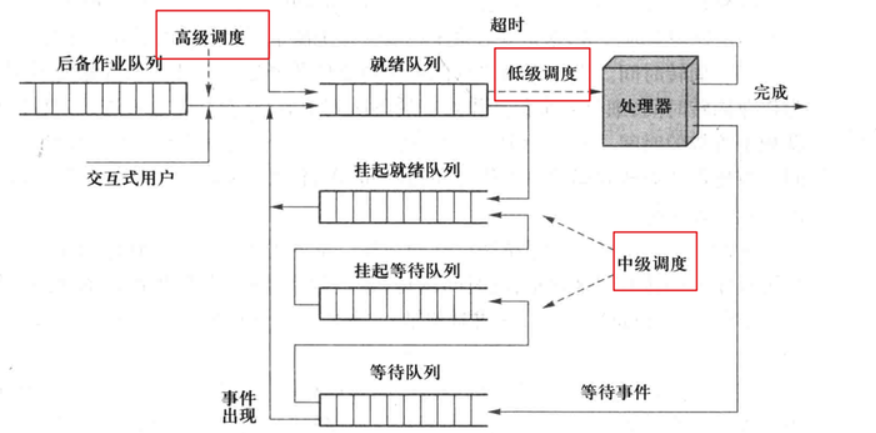
### 2.2.3 更新电梯状态

在每次调度周期中，Controller会更新每部电梯的状态。根据当前楼层、目标楼层和运动方向来决定电梯的下一步行动。同时，也会根据当前楼层是否有请求来执行相应的动画。

### 2.2.4 故障处理

如果电梯出现故障，控制器会将其标记为损坏并禁用相关按钮。经过一定时间后，控制器会自动修复损坏的电梯，使其重新参与调度。

本电梯调度算法可以类比为操作系统中的进程调度。在操作系统中，处理器需要根据优先级、资源需求等因素来选择合适的进程进行执行。电梯调度问题的电梯和控制器分别对应于计算机处理器和操作系统调度器，楼层按钮对应于不同的进程请求。通过合理的调度策略，可以实现资源的合理分配和高效利用，进而提高整个系统的性能。



# 3 实现

## 3.1 可视化界面实现

本项目使用PyQt5图形库实现可视化界面。通过窗口类设置应用程序的主界面，设置 UI 元素，并连接了适当的信号和槽以处理用户交互。实际的电梯调度逻辑由一个单独的Controller类处理，该类从dispatch模块导入。

## 3.2 调度算法实现

### 3.2.1 优先级计算

**def** calculate\_prior(self, move):

    scores = []

**for** i **in** range(5):

        score = 0

**if** self.elevators[i].damage == True:

            score = -1

**elif** (self.floor - self.elevators[i].floor) \* self.elevators[i].move < 0:

            score = -2

**else**:

            result = [0]

**for** j **in** self.elevators[i].goal:

                result.append(abs(j - self.floor))

            score = 20 - max(result)

**if** self.elevators[i].move == 0:

                score = 20 - abs(self.elevators[i].floor - self.floor)

            goal\_len = len(self.elevators[i].goal)

**if** goal\_len != 0:

**if** (self.elevators[i].goal[goal\_len - 1] - self.floor) \* move < 0:

                    score = -2

**if** move == 1:

                index = 0

**else**:

                index = 1

**if** i == self.elev.request[self.floor - 1][1 - index]:

                score = -4

        scores.append(score)

**return** scores

### 3.2.2 电梯调度

**def** dispatch(self, move):

    self.floor = self.elev.floor

    priors = self.calculate\_prior(move)

**for** i **in** range(5):

**if** move == 1:

            self.elev.pressed[self.floor - 1][0] = 1

**if** move == -1:

            self.elev.pressed[self.floor - 1][1] = 1

    max\_index = np.argmax(priors)

    max\_value = max(priors)

**if** move == 1:

        index = 0

**else**:

        index = 1

    self.elev.request[self.floor - 1][index] = max\_index

**if** max\_value == -1:

        QtWidgets.QMessageBox.critical(self.elev, "Error", "All elevators have been damaged!")

**elif** max\_value == -2:

**if** self.floor **not** **in** self.elevators[max\_index].prior:

            self.elevators[max\_index].prior.append(self.floor)

**else**:

**if** self.floor **not** **in** self.elevators[max\_index].goal:

            self.elevators[max\_index].goal.append(self.floor)

### 3.2.3 电梯状态更新

**def** update\_elevator\_position(self):

    self.floor = self.elev.floor

**for** i **in** range(5):

        self.elevators[i].moving()

**if** self.elev.pressed[self.floor - 1][0] == 1:

            self.elev.up\_button[i].setDown(True)

**else**:

            self.elev.up\_button[i].setDown(False)

**if** self.elev.pressed[self.floor - 1][1] == 1:

            self.elev.down\_button[i].setDown(True)

**else**:

            self.elev.down\_button[i].setDown(False)

### 3.2.3 电梯损坏与修理

**def** alarm(self, no):

    self.elev.alarm\_button[no].setStyleSheet("background-color: rgb(255, 248, 0);")

    self.elevators[no].damage = True

**for** i **in** range(20):

        button = self.elev.findChild(QtWidgets.QPushButton, "button " + str(no) + ' ' + str(i + 1))

        button.setEnabled(False)

    self.elev.open\_button[no].setEnabled(False)

    self.elev.close\_button[no].setEnabled(False)

    self.elev.up\_button[no].setEnabled(False)

    self.elev.down\_button[no].setEnabled(False)

    self.fix\_timer.timeout.connect(partial(self.fix, no = no))

    self.fix\_timer.start(20000)

**def** fix(self, no):

    self.elev.alarm\_button[no].setStyleSheet("background-color: rgb(240, 0, 0);")

    self.elevators[no].damage = False

**for** i **in** range(20):

        button = self.elev.findChild(QtWidgets.QPushButton, "button " + str(no) + ' ' + str(i + 1))

        button.setEnabled(True)

    self.elev.open\_button[no].setEnabled(True)

    self.elev.close\_button[no].setEnabled(True)

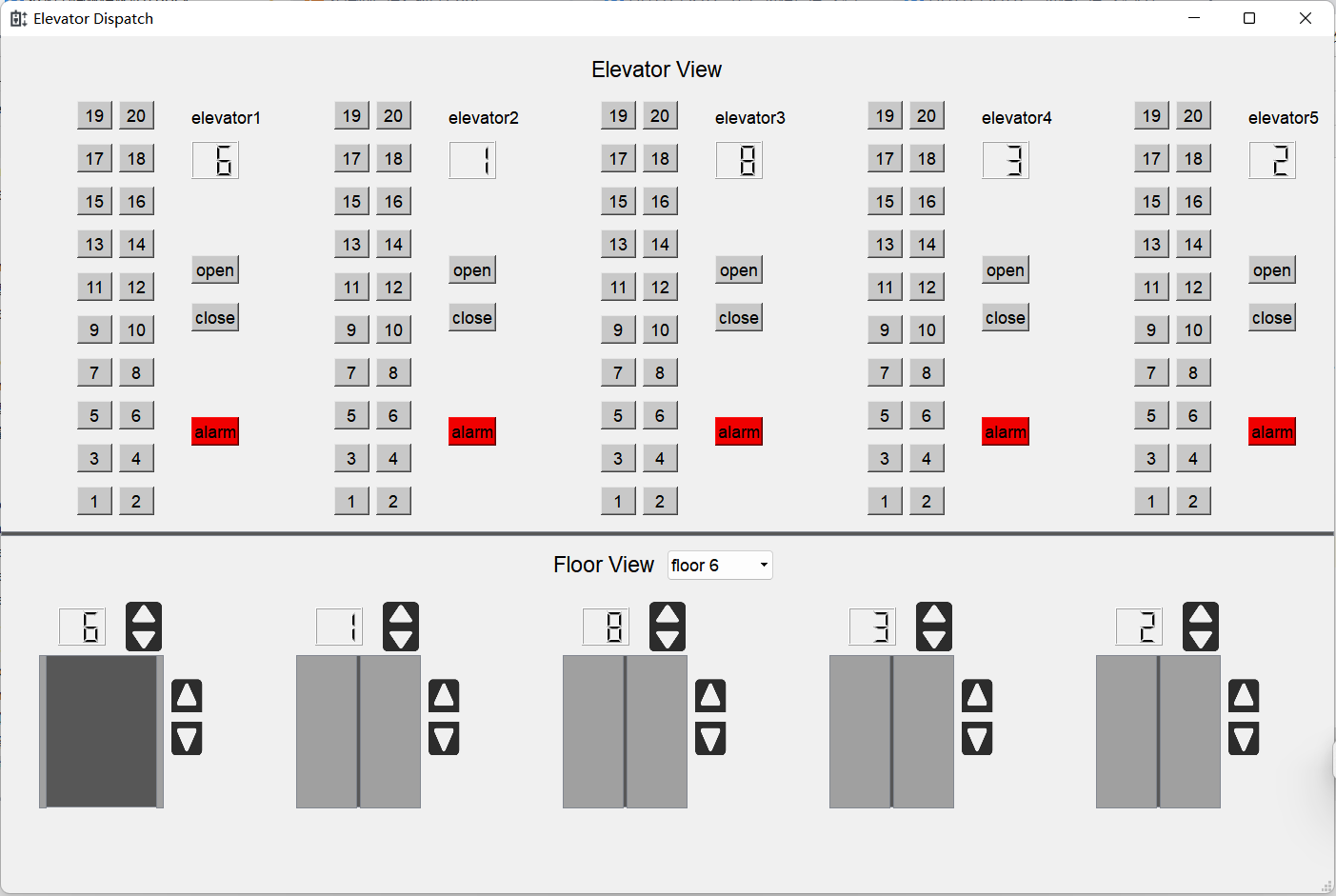
    self.elev.up\_button[no].setEnabled(True)

    self.elev.down\_button[no].setEnabled(True)

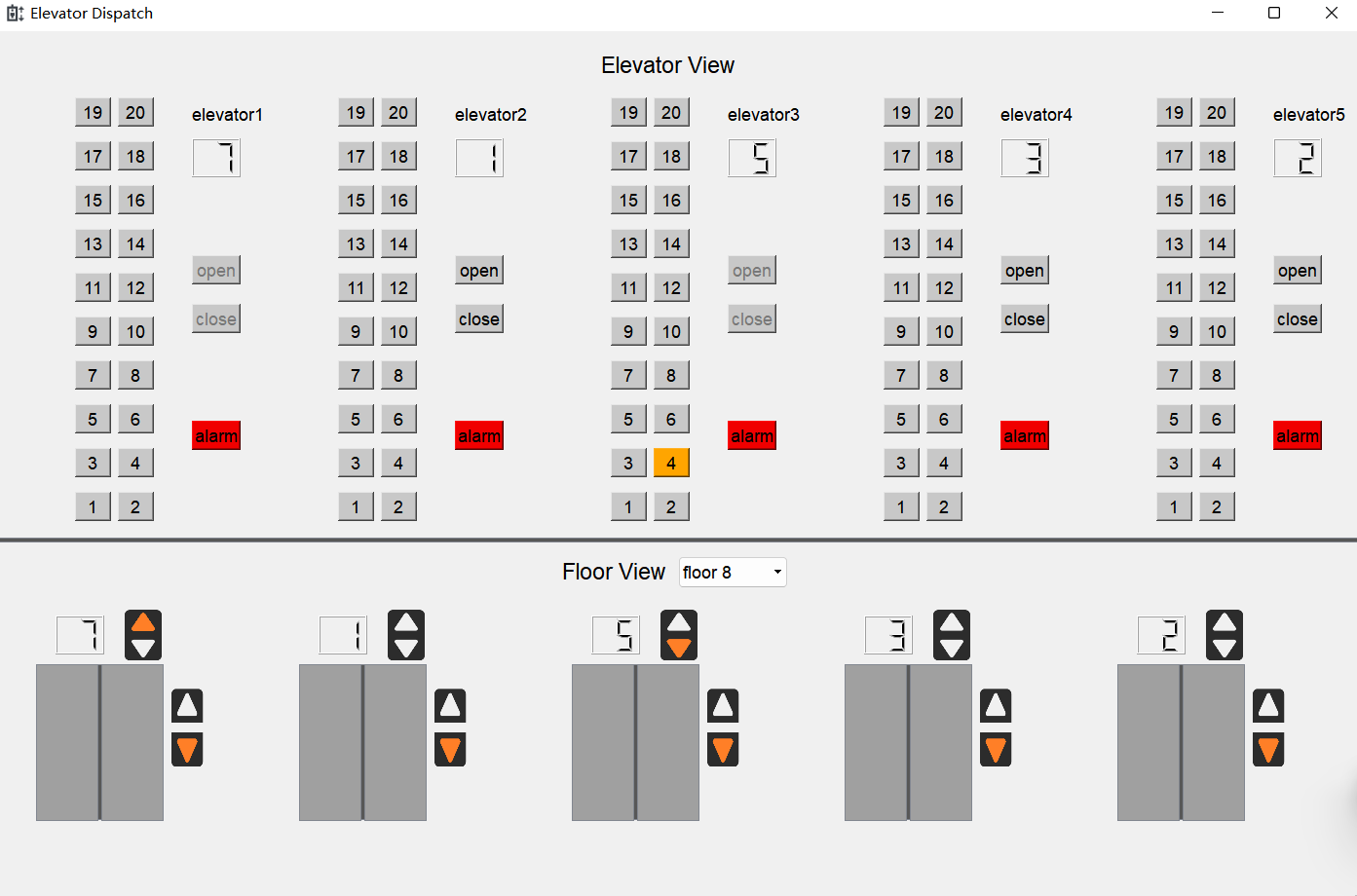
# 4 测试

## 4.1 电梯调度测试

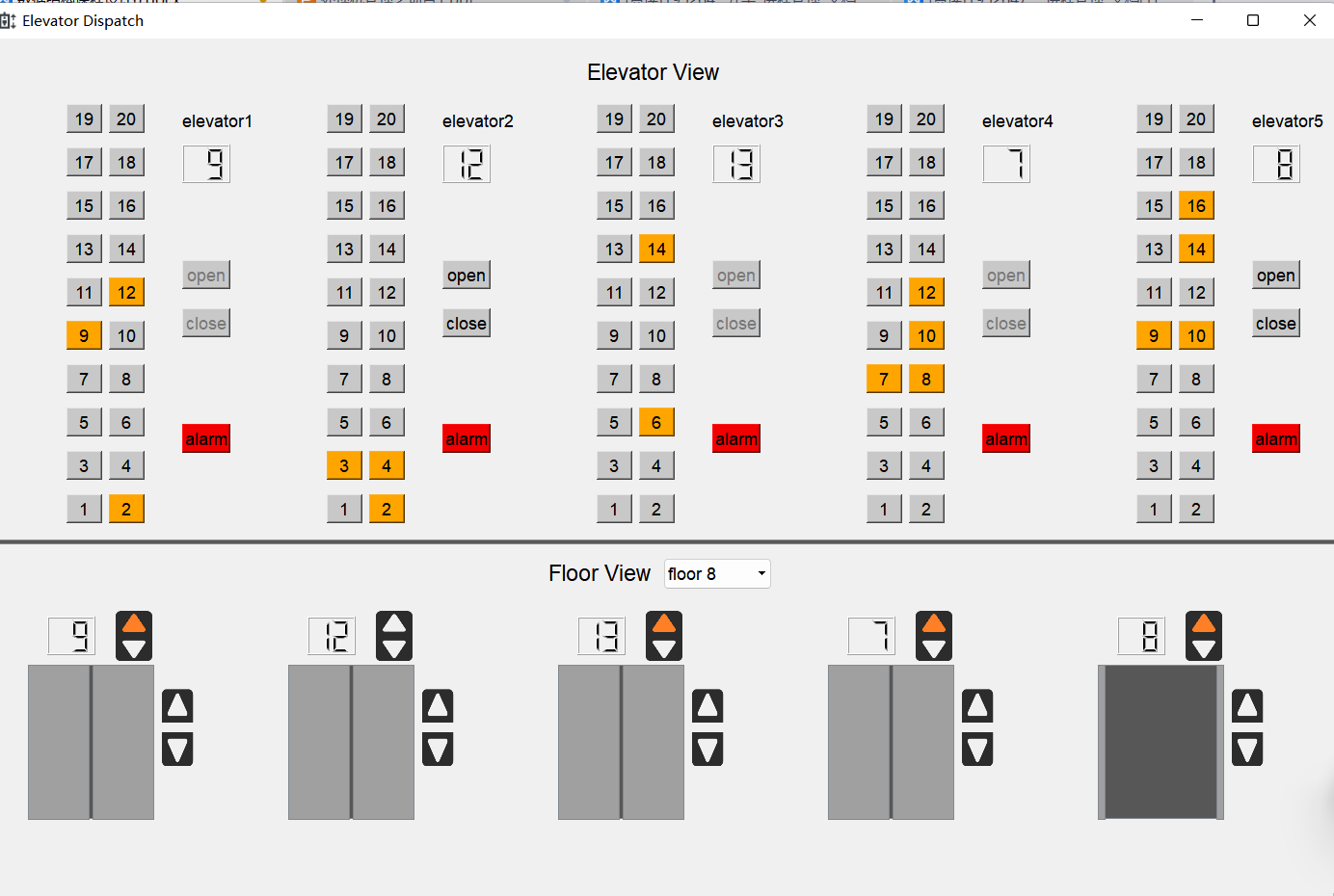
电梯内部按下6楼，在6楼停下：



在8楼按向下按钮，1号电梯前往8楼：

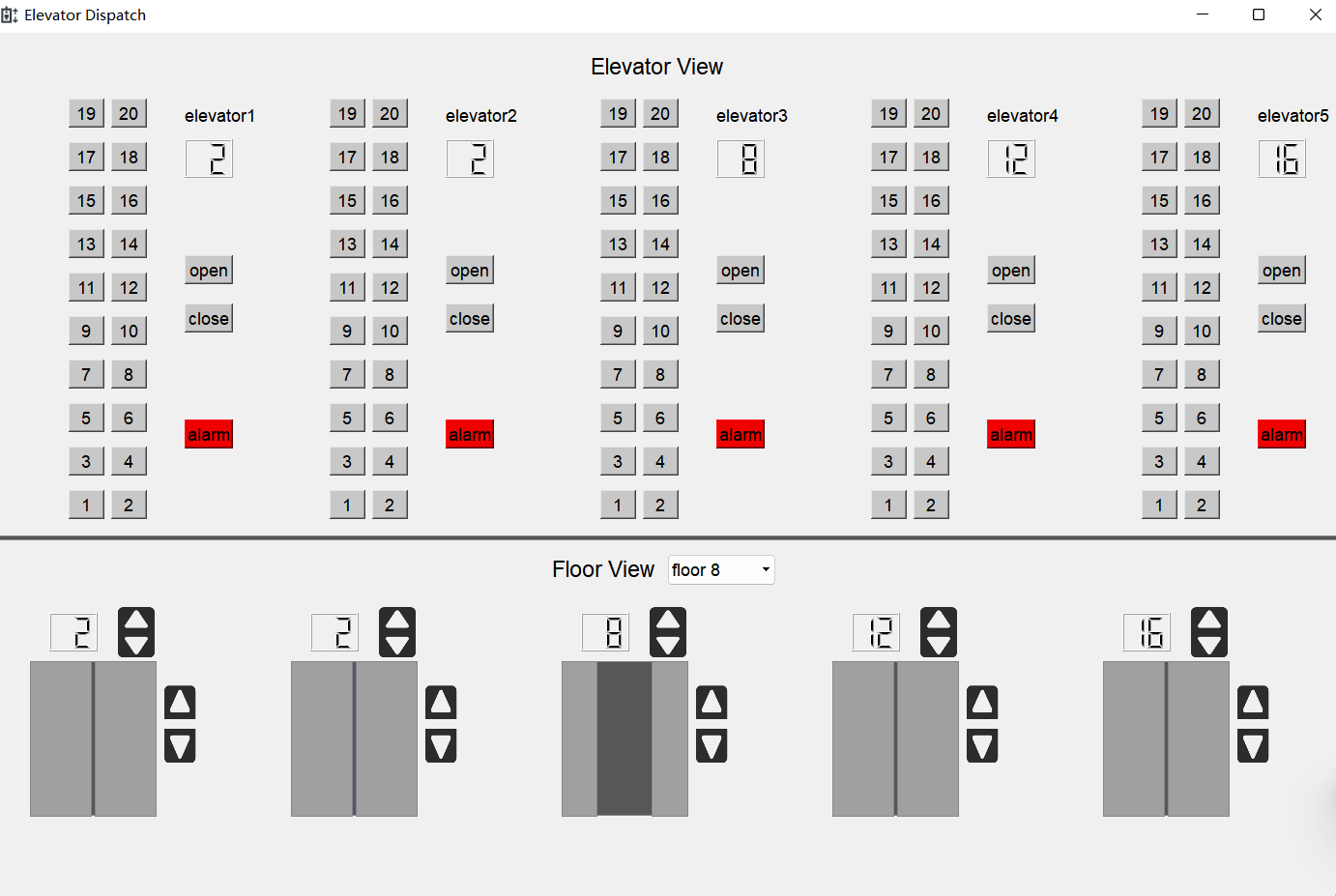


连续响应：

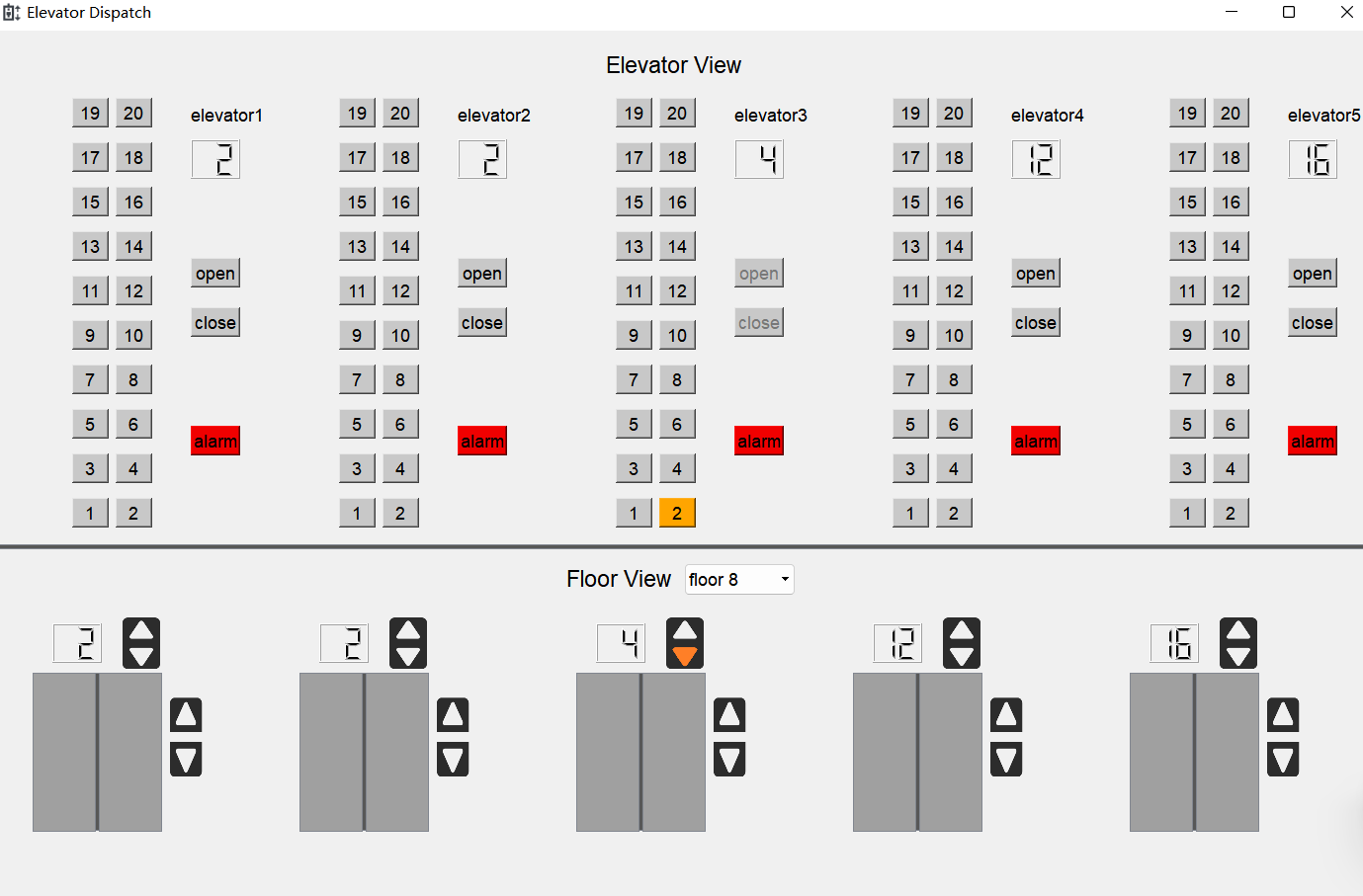


## 4.2 开关门按钮测试

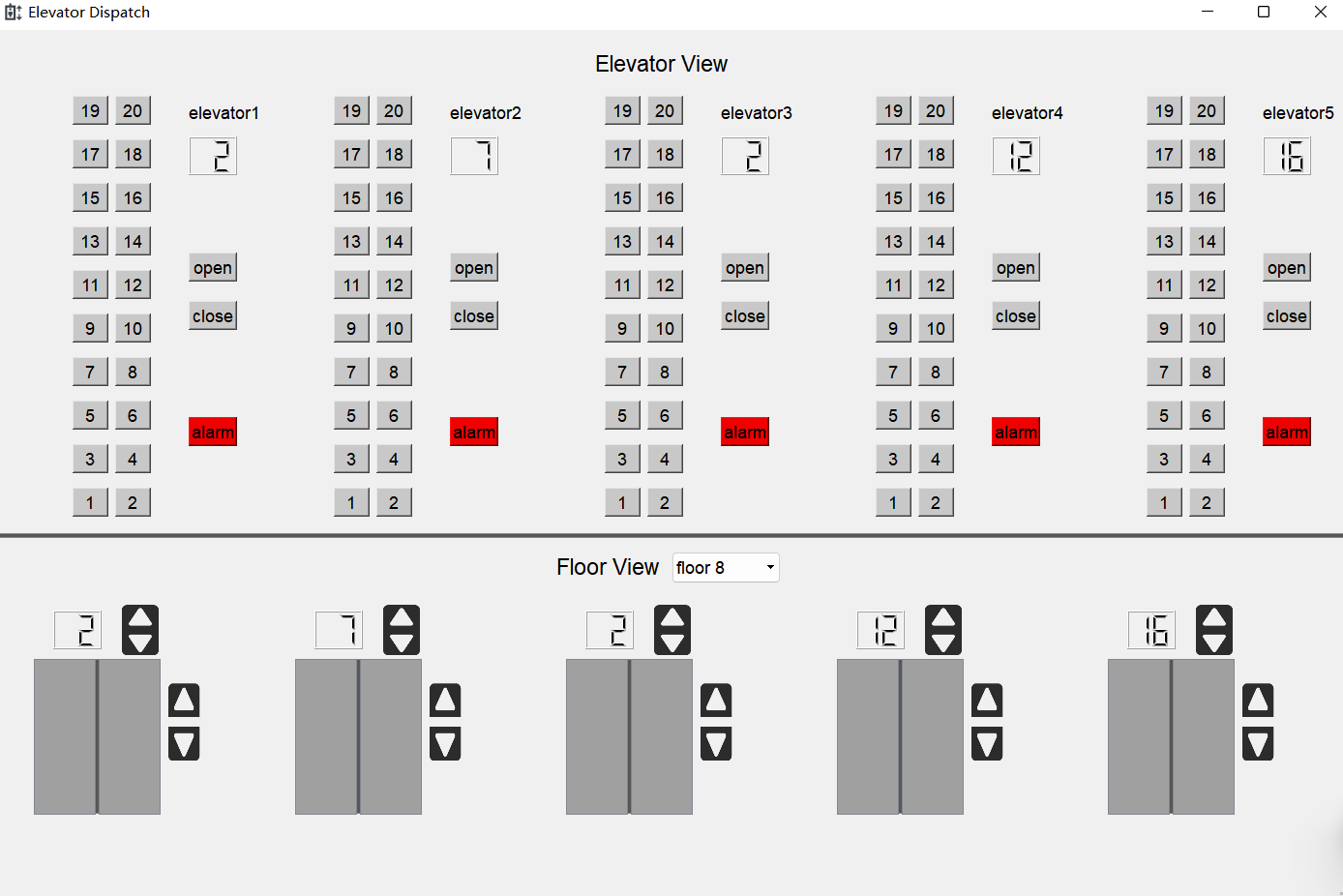
电梯在8楼，点击开门：



电梯行进中，开关门无效：

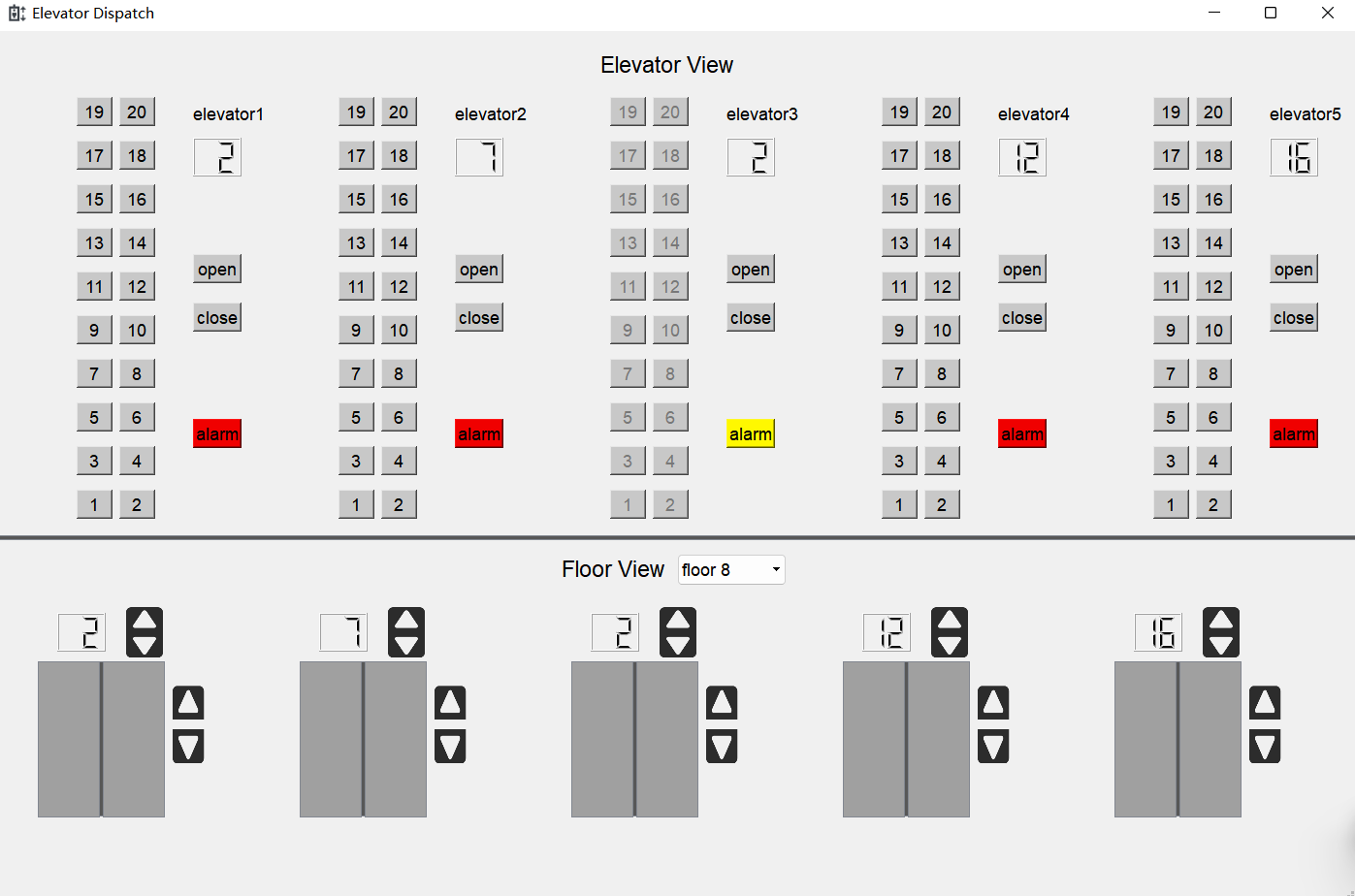


电梯在7楼，视角在8楼，点击开门，电梯在7楼停止，无开门动画：

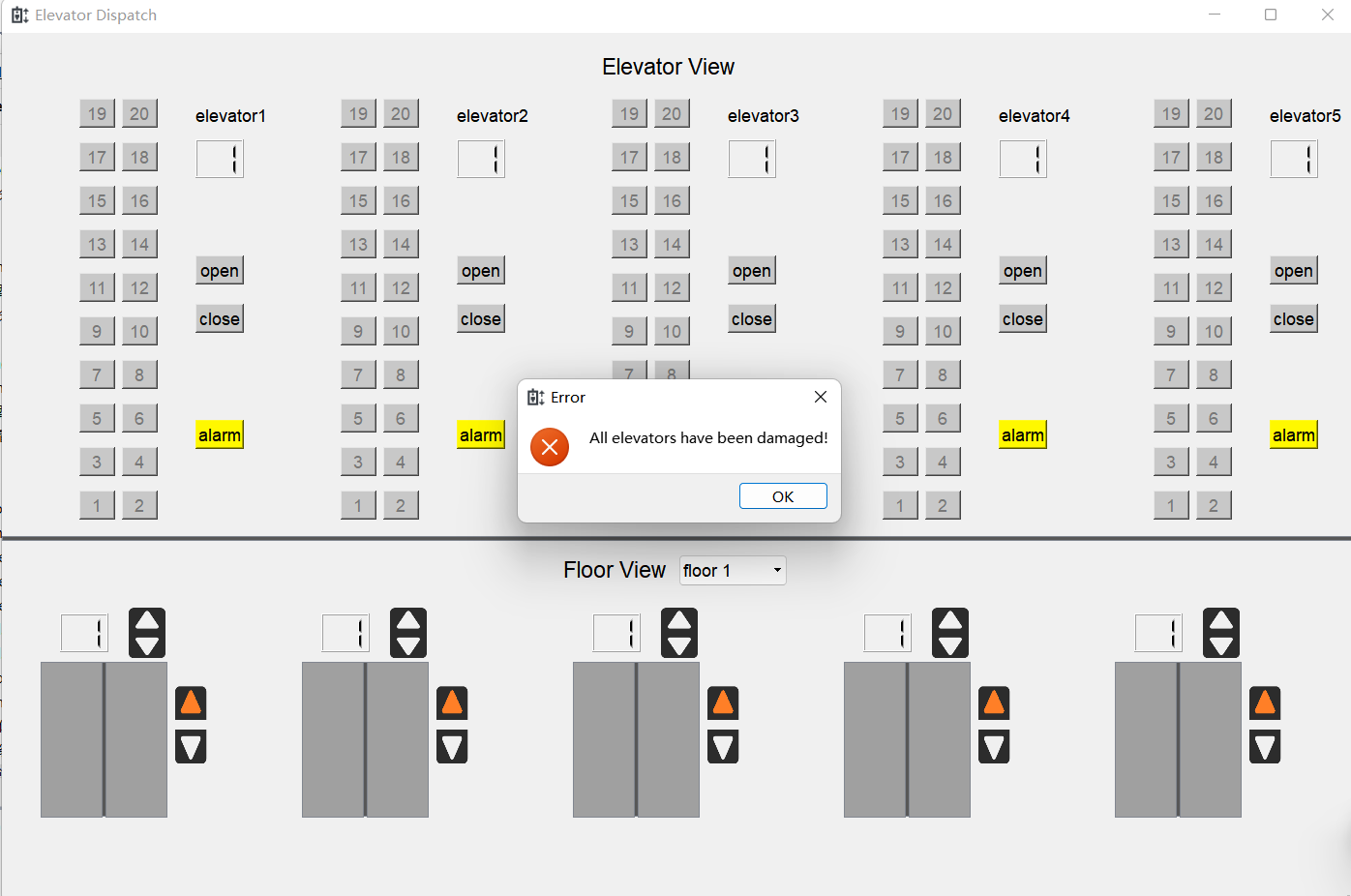


## 4.3 报警按钮测试

按下报警按钮，电梯其余按钮失效，30s后恢复：



所有电梯损坏：



## 4.4 按钮连结性测试

