项目说明文档

操作系统课程设计

——电梯调度系统

作 者 姓 名： 李德涛

学 号： 1852141

指 导 教 师： 王冬青

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

目 录

[1 简介 1](#_Toc495668153)

[2 设计 2](#_Toc495668156)

[2.1 楼层呼叫算法 2](#_Toc495668157)

[2.2 电梯运行算法 2](#_Toc495668158)

[3 实现 3](#_Toc495668161)

[4 测试 5](#_Toc495668186)

# 1 简介

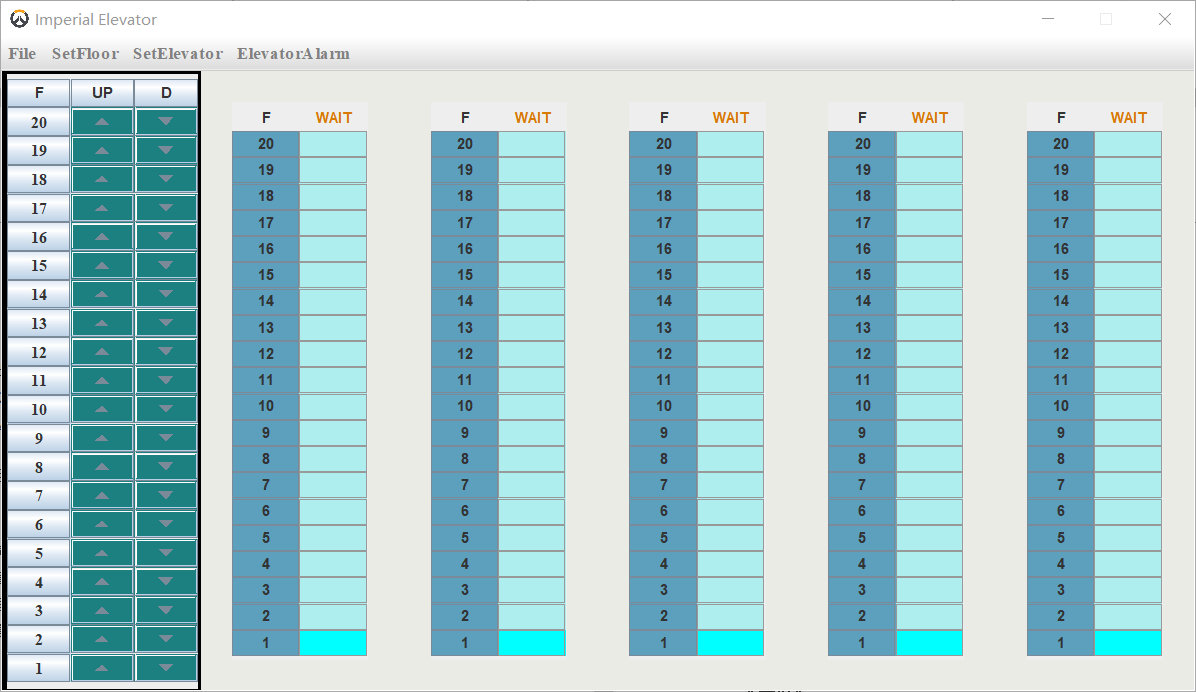
开发环境：jdk-14.0.1\_windows-x64

开发平台：Eclipse

开发语言：JAVA

项目名称：电梯调度系统Imperial Elevator

操作界面：



操作说明：

在安装有JAVA JDK包的情况下，直接打开imperialElevator.jar文件即可。

菜单栏有四类操作：File中可退出程序；SetFloor中可自定义楼层数（预设有20到25六种可选）；SetElevator中可自定义电梯数（预设有3到6四种可选）；ElevatorAlarm中可使用电梯报警功能。

主界面由两部分组成，左侧表示每层楼电梯门口的外部按钮，包括上行（UP）、下行（D）两种功能。点击后按钮变绿，表示系统已接受响应。右侧依次显示各部电梯的详细情况，最上面的橘色字体代表电梯此时的状态，共有WAIT等待、UP上行、DOWN下行、OPEN开门中、CLOSE关门中五种不同状态；下面左侧为电梯内部表示目的地的按钮，右侧为电梯此时位置，深蓝色表明电梯处于下行，浅蓝色则为上行。

# 2 设计

电梯调度系统的核心算法包括以下两部分：

## 2.1 楼层呼叫算法

外部需求主要分为两种：

外部需求为上行需求时，优先找当前正处于上行状态且该电梯的当前楼层在外部需求楼层之下的电梯，在有多部电梯满足要求的情况下取距离最近的一个电梯接收该外部需求；如果当前没有上行电梯则寻找静止电梯和在需求楼层以下的下行电梯中距离最近的电梯，若没有找到匹配结果则下次在进行分配。

外部需求为下行需求时，优先找当前正处于下行状态且该电梯的当前楼层在外部需求楼层之上的电梯，在有多部电梯满足要求的情况下取距离最近的一个电梯接收该外部需求；如果当前没有上行电梯则寻找静止电梯和在需求楼层以上的上行电梯中距离最近的电梯，若没有找到匹配结果则下次在进行分配。与上行需求相似。

2.2 电梯运行算法

# 按照楼层的顺序依次服务需求，使得电梯在最底层与最顶层之间往返运行，没有需求时保持静止。保存电梯所在楼面与目的地楼面的差值。电梯响应需求是互联结的。

# 3 实现

源代码由三个文件组成：

ElevatorMain.java 含有main函数，初始化界面，作为程序入口。

ElevatorFrame.java 初始化主程序框架，包括菜单栏、程序界面，设置UI及属性，生成N个代表电梯的线程，并担任调度电梯的角色。

ElevatorThread.java 初始化电梯界面的布局，完成单个电梯线程上下行的工作。

**private ElevatorThread[] eleThread;**

上面一行代码代表电梯的线程数组。

程序通过电梯数eleNum生成所需的线程数，如下：

**for** (**int** i = 0; i < *eleNum*; i++)

{

ElevatorThread list = **new** ElevatorThread();

cp.add(list);

list.getThread().start();

eleThread[i] = list;

}

同时也为自定义电梯数量提供了一个方便的接口。

满足电梯外部需求时：

寻找响应上行指令最近的电梯：

**for** (**int** j = 0; j < eleThread.length; j++)

{

**if** (eleThread[j].isAbort()

|| (eleThread[j].isUp() && floor >= eleThread[j].getCurPos()))

{

**int** temp = Math.*abs*(floor - eleThread[j].getCurPos());

**if** (temp < distance)

{

whichList = j;

distance = Math.*abs*(floor - eleThread[j].getCurPos());

}

}

}

寻找响应下行指令最近的电梯：

**for** (**int** j = 0; j < eleThread.length; j++)

{

**if** (eleThread[j].isAbort()

|| (eleThread[j].isDown() && floor <= eleThread[j].getCurPos()))

{

**int** temp = Math.*abs*(floor - eleThread[j].getCurPos());

**if** (temp < distance)

{

whichList = j;

distance = Math.*abs*(floor - eleThread[j].getCurPos());

}

}

}

电梯内部使用的监听类：

**for** (**int** i = 0; i < numButton.length; i++)

{

**if** (e.getSource() == numButton[i])

{

numState[i] = **true**;

numButton[i].setBackground(numColor1);

**if** (direction == ABORT)

{

tarPos = i;

}

**if** (direction == UP)

{

tarPos = getMaxPressedNum();

}

**if** (direction == DN)

{

tarPos = getMinPressedNum();

}

}

}

当电梯到达目标楼层时，执行

Thread.sleep(100);

并在线程睡眠期间更改电梯状态，模拟开门、关门、等待的状态，以向上运行为例：

dispState.setText("UP");

Thread.*sleep*(600);

listButton[i].setBackground(floorColor1);

**if** (i > oldPos)

{

listButton[i - 1].setBackground(floorColor0);

}

**if** (numState[i])

{

dispState.setText("OPEN");

Thread.*sleep*(1000);

dispState.setText("CLOSE");

numButton[i].setBackground(numColor0);

Thread.*sleep*(1000);

}

curPos = i;

# 4 测试

## 功能测试

