**电梯调度系统**

操作系统第一次作业

指导老师：王冬青

学院/专业：软件学院/软件工程

# 一、项目分析

## 项目名称

电梯调度系统

## 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。

## 功能需求

1、每个电梯里面设置必要功能键：如数字键、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。

3、五部电梯门口的按钮是互联结的，即一个楼层外部按钮被按下后，五台电梯都有可能来响应该请求。

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。

## 4、开发环境

**○Windows10 家庭中文版**

**○C#.NET**

**○Unity 2019.4.21f1c1**

**○屏幕分辨率要求：1920\*1080**

# 二、项目设计

## 1、界面设计

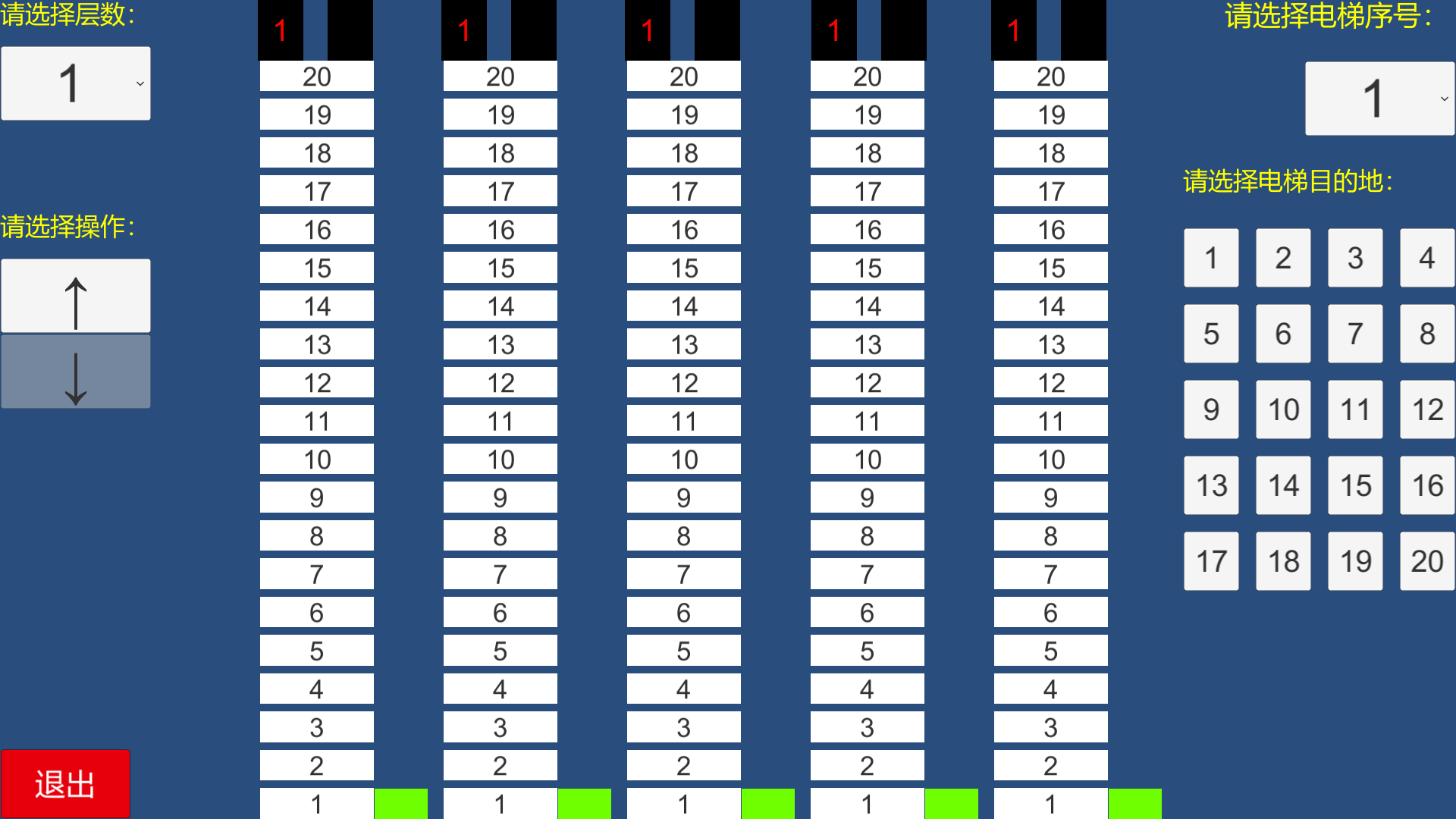
界面中央是并行排列的五部电梯，每一部电梯会有一块板子上下滑动来表示电梯的上下移动。每部电梯上方有两个数码管，来实时显示电梯的状态和当前楼层。

界面左上方是一个下拉选择框，用于选择楼层，来模拟用户处在1到20楼的任意一层楼。选择框下方是两个按钮，分别是上行和下行按钮，用户通过按下按钮来发出乘坐电梯的请求。在按钮已经按下或不可用时，会变为灰色。

界面右上方同样为一个下拉选择框，用于选择当前电梯。选择框下方为20个按钮，用于选择乘坐当前电梯前往20层楼中的某一层。在按钮已经按下时，会变为灰色。

界面左下角为一个退出按钮，点击退出按钮或按下esc键可以退出系统。

**界面截图：**



## 2、调度算法设计

#### 2.1、总体调度概述

本项目采取扫描法来分配任务，即在外部请求出现后，找到未被调度的电梯或运行方向和请求方向相同的电梯进行分配，若出现未被分配的任务则在之后出现可用电梯再分配。同时考虑到用户在电梯内部按下按钮后会为电梯增加任务，可能会影响原有的任务分配，因此增加了放弃任务并重分配的机制。对于单个电梯，只需要按前进方向依次停顿即可。于是，本项目的调度主要分为两个部分：外部调度、内部调度。

#### 2.2、电梯内部调度

对于每一台电梯都会收到来自内部按钮的请求和调度器分配的楼层按钮请求，所以电梯需要一个算法来管理属于这部电梯的请求任务，计算运行路径。本项目采用LOOK算法来完成电梯内部任务的调度。

2.2.1、算法原理

LOOK算法是扫描算法的一种改进。扫描算法(SCAN)是一种按照楼层顺序依次服务请求的算法，它让电梯在最底层和最顶层之间连续往返运行，在运行过程中响应处在于电梯运行方向相同的各楼层上的请求。扫描算法较好地解决了电梯移动的问题，在这个算法中，每个电梯响应乘客请求使乘客获得服务的次序是由其发出请求的乘客的位置与当前电梯位置之间的距离来决定的，所有的与电梯运行方向相同的乘客的请求在一次电梯向上运行或向下运行的过程中完成，免去了电梯频繁的来回移动。

对LOOK算法而言，电梯同样在最底层和最顶层之间运行。但当LOOK算法发现电梯所移动的方向上不再有请求时立即改变运行方向，而扫描算法则需要移动到最底层或者最顶层时才改变运行方向。

2.2.2、具体实现

一个楼层的请求一共有四种情况：无请求、上行、下行、内部请求（即电梯内部按下去往该楼层的按钮）。

本项目由于允许放弃任务重新分配，所以将内外请求分别处理。对内部请求使用一个数组public bool[] buttons = new bool[20]来判断是否按键被按下，对外部请求使用一个二维数组public bool[,] taskStop = new bool[2, 20]来判断是否分配上行、下行任务，既没有按钮被按下，也没有分配相应任务时即为无请求。

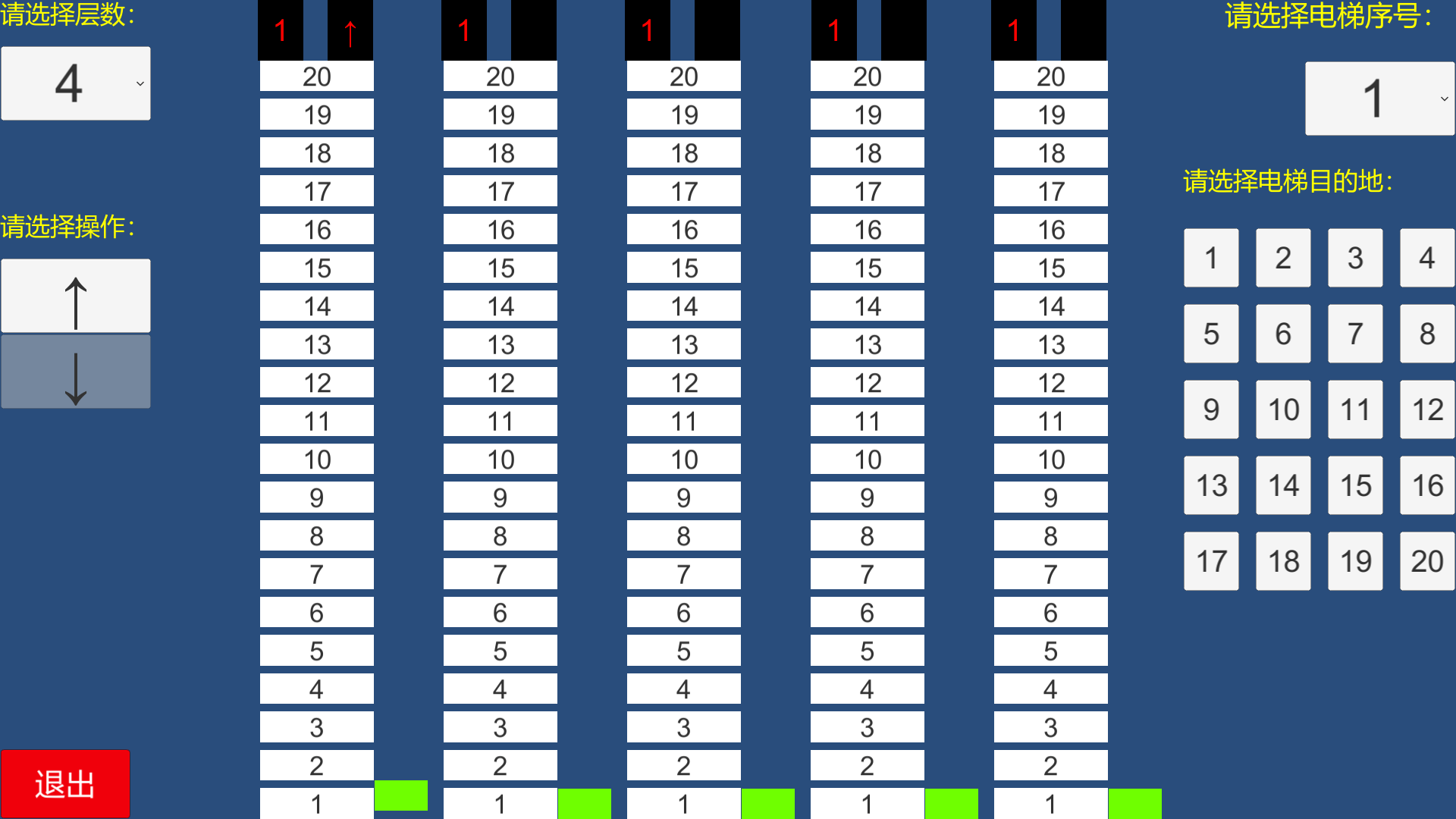
电梯运行时，会保证自己只执行和当前运行方向一致的任务，到达最高（低）层后再检测是否有运行方向相反的任务。运行方向由枚举类型（Order）order保存。order只会在电梯运行到最高（低）层和order==Order.Free状态下收到新任务改变。

电梯的运行状态分为三种：向上，向下，空闲。项目中设置了一个枚举类型（Direction）state来记录当前状态。在Update函数中，将会根据当前的state来进行电梯的向上、向下移动和改变层数。处于向上、向下状态的电梯，每次到达目标楼层都会判断是否到达最高（低）层，并停留一段时间，若到达最高（低）层则更新order。处于停留状态的电梯，若order处于自由状态则保持不动，否则将会在计时器结束后继续按当前order方向执行任务（state = (Direction)order）。

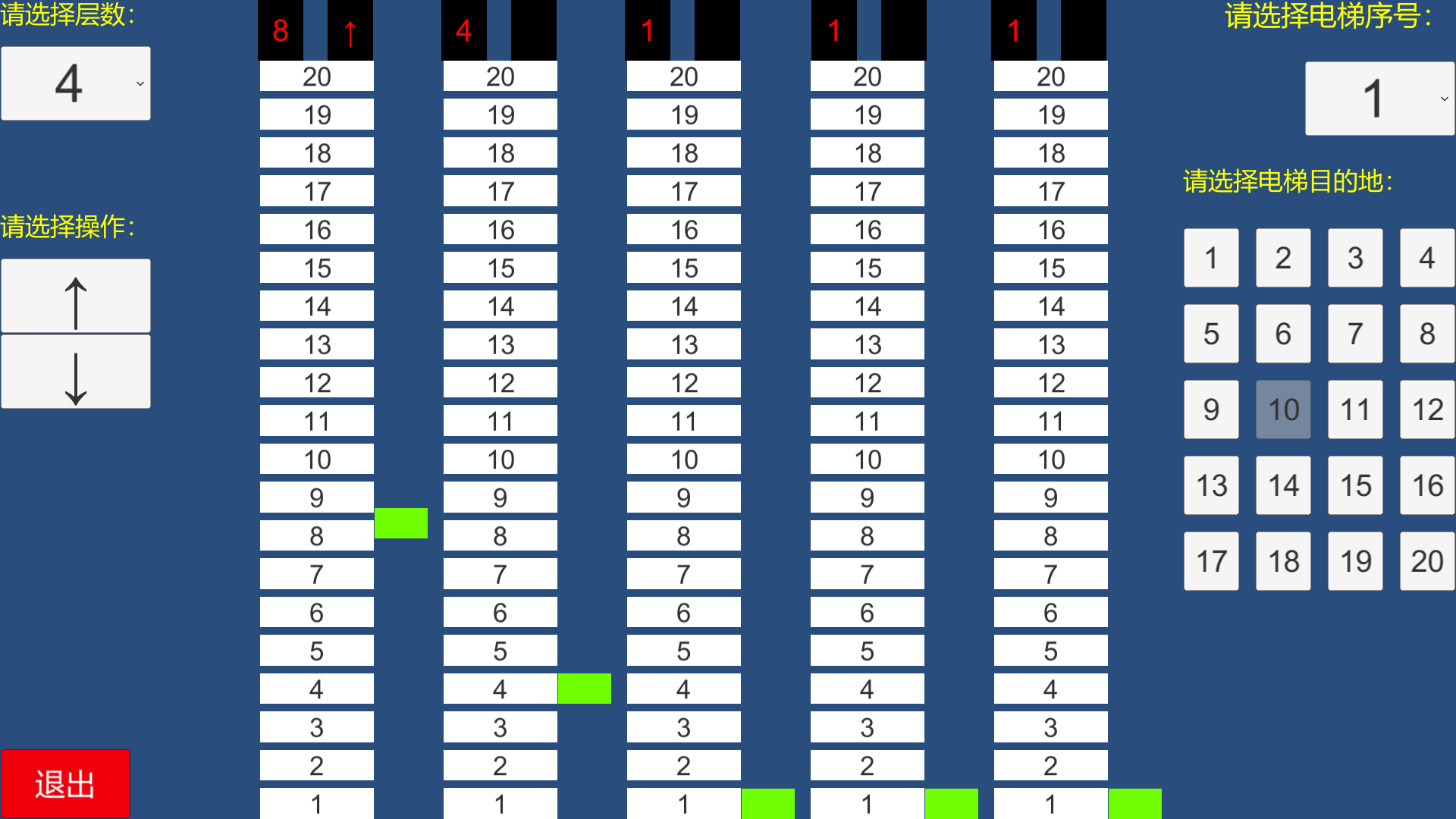
电梯运行时，在Update函数中，每当改变层数，都会与buttons和taskStop来比较，若有相应任务则会停止，state=Direction.Stop，并将所有方向相同的任务执行完毕、更新运行方向，否则继续沿当前方向运行。

当电梯内按钮被按下时，可能会与外部调度冲突（调度时间过长），因此有放弃任务并再分配的相应函数。具体思路为：当电梯被分配了一个上行任务后，若电梯处于下行状态（由自由状态从高层向低层调度），则lowLevel一定不为空，此时若按下比lowLevel更低层数的按钮，会使得任务发生冲突，总体调度时间大幅增加。因此将外部任务放弃，更新lowLevel，沿当前方向继续移动至低层。

反之思路相同，举例为：电梯1收到4层的下行请求并响应。



此时电梯内用户按下10层按钮，电梯放弃外部调度命令，并前往10层，此时电梯2被调度响应被放弃的任务。



这样避免了响应10层的时间过长和响应4层的时间过长，对于更为极端的层数效果尤其明显。

## 2.3、外部调度

2.3.1、总体思路概括

对于每一个外部请求（即楼层的上下按钮）而言，当这种请求出现时，不应该对应于某一个特定的电梯，而是应该由联合调度系统通过计算来规划出更可能是最好选择的电梯来执行外部请求（让此电梯对该请求进行独占）。

通过对日常生活的调查我们发现，电梯的联合调度主要遵循以下几个原则：

1、时间最短原则：尽量让乘客等待电梯的时间最短；

2、平均载客原则：尽量让乘客平均分配到各个电梯中，以提升总体载客量；

3、电力节省原则：尽量不让电梯做往返运动。

在本项目中，主要考虑时间最短原则，同时只考虑合理的电梯调度，将任务分配到合理的电梯，没有合适电梯则在电梯状态更新之后再分配，从而保障电梯内外用户的等待时间最短，且和当前运行方向一致的任务优先被执行。

2.3.2、任务分配

为了保证时间最短和顺路两个基本原则，我们分配任务时选择电梯有一定筛选。

对于所有任务，只有满足以下条件之一的电梯可调度：

1.电梯处于自由状态且未开门（timer<=0）。

2.电梯处于自由状态，开门，但收到的是当前层的指令（distance[i]==0）。

3.电梯未被调度或被调度方向与新调度方向一致，并且向上运行的电梯只接受比自己层数高的向上调度命令，向下运行的电梯只接受比自己层数低的向下调度命令。

示例：

电梯1在4层自由状态且未开门，它可以被任意任务调度。

电梯2在6层自由状态且开门，它只能被6层的任务调度以开门。

电梯3在8层向上运行且未被调度，它只能被9层及以上的上行任务调度，而不能被5层的上行任务调度。

电梯4在6层向上运行，且已被9层的下行任务调度，它不能被其他任何任务调度。

除此之外，还要计算每个电梯与任务的目标层的距离，在可调度的电梯中选择距离最小的电梯，节省时间。

若没有可调度的电梯，函数会返回null，并等待电梯状态更新后重新调度。

# 三、类设计

## 3.1、总述

本项目采用Unity开发，文件主要是C#脚本文件与Unity文件。

脚本文件共包含三个脚本类：ElevatorButton，Elevator和GameManager，还有一个实体类UDTask，以下将分别做具体介绍。同时Elevator.cs文件中还包含了两个枚举类型的定义，由于定义较为简单，这里直接给出。

public enum Direction

{

Stop,//停止

Up, //正在向上

Down//正在向下

};

public enum Order

{

Free, //自由状态

Up, //向上运行

Down//向下运行

};

可视化与可交互的控件在Unity文件中，此处不做具体介绍。

## 3.2、UDTask类（外部上下调度任务类）

成员变量：

public readonly Direction dir;//用于储存任务的运行方向

public readonly int taskLevel = 1;//用于储存任务的层数

成员函数：

public UDTask(Direction direction,int taskFloor)

//构造函数

## 3.3、GameManager类

成员变量：

public static GameManager Instance;//标明实体，方便外部调用

public Dropdown floor;//选择层数的下拉框，获取选择的为哪一层

public Dropdown number;//选择电梯的下拉框，获取选择的为哪部电梯

public Button upButton;//向下的按钮

public Button downButton;//向上的按钮

public Button exitButton;//退出系统的按钮

public List<UDTask> mainTasks = new List<UDTask>();//储存所有未调度的任务

public List<UDTask> dispachedTasks = new List<UDTask>();//储存所有已调度的任务

public bool[,] buttonInteractable = new bool[2, 20];//储存当前上、下按钮是否可交互

成员函数：

void Start()

//初始化并添加按钮事件

void Update()

//每帧更新上下按钮的可交互性

private void Awake()

//用于得到实体Instance

private void AddUDTask(Direction direction)

//此处添加外部调度任务

public void MoveTask(int Index)

//将任务从mainTasks移动至dispachedTasks

public void TaskFinished(UDTask task, Elevator elevator)

//任务已完成，移除任务

public void TaskRestart(UDTask task)

//任务被放弃，重新开始任务

## 3.4、ElevatorButton类

成员变量：

public static ElevatorButton Instance;//标明实体，方便外部调用

public Button[] levels;//储存所有楼层按钮

public Elevator[] elevators; //储存所有电梯

public int elevatorNum = 0; //当前选择的电梯

public List<Elevator> UpElevators = new List<Elevator>();//储存所有被外部上行任务调度的电梯

public List<Elevator> DownElevators = new List<Elevator>();//储存所有被外部下行任务调度的电梯

成员函数：

void Start()

//初始化

void Update()

//每帧对mainTask的任务进行一次调度尝试，并更新当前选中的电梯和更新UI中电梯按钮是否被按下

private void Awake()

//用于得到实体Instance

private void AddListener(Button button, int Index)

//为按钮添加监听器，方便循环中调用

private void AddMoveTask(int taskLevel)

//为电梯内部调度添加任务

private void AddMoveTask(int taskLevel, Elevator ele)

//为电梯外部调度添加任务

private Elevator GetAvailavleElevator(int Index)

//使用外部调度算法，获取当前可调度的最优电梯

## 3.5、Elevator类

成员变量：

static public float speed = 20.0f;//控制运动速度的变量

public Direction state = Direction.Stop; //表示当前电梯运行状态的变量

public Order order = Order.Free; //表示当前电梯运行方向的变量

public int level = 1; //电梯所在层数

public int? highLevel = null; //电梯上行任务的最高层

public int? lowLevel = null; //电梯下行任务的最低层

public bool[] buttons = new bool[20]; //电梯的20个楼层按钮是否被按下

public float timer = 0.0f; //计时器，模拟开关门

public bool[,] taskStop = new bool[2, 20]; //用于记录是否被分配外部上行、下行任务

public Image EleFloor; //用于显示所在楼层

public Image EleStatus; //用于显示运行状态

成员函数：

private void Update()

//每帧依照当前运行状态、运行方向执行任务

private void Start()

//初始化

public int GetLevel(bool isUp)

//得到当前的楼层，由于向上向下的楼层计算方式不完全一致，因此需要参数

public void GiveUpTask(bool isUp)

//放弃任务，为了节省遍历时间，因此需要参数

# 四、总结

通过本次实验，一方面，我体会了多线程调度算法的设计与实现，在操作系统中，每个电梯就如同一个线程，每个线程都会申请资源，对电梯的调度就是在模拟对线程及其资源的调度。同时，电梯放弃任务重新分配任务的过程也是线程间通信的使用。另一方面，我对C#语法和Unity引擎的使用有了更多的了解，提升了用 C#解决问题的能力。

# 五提交说明及注意事项

本项目包含包含四个文件（夹），第一个是这份项目报告，第二是打包好的项目可执行文件（Elevator\_exe），第三为完整项目文件（Elevator），包含Unity配置文件和C#源码，可以在**Unity 2019.4.21f1c1**查看Unity文件，C#代码部分可用VS查看，第四为单独的该项目调度系统源码（Elevator\_codes），仅包含.cs文件。

本项目打包后的可执行文件请用Windows64位系统运行，并且要求屏幕分辨率为1920\*1080，以保证电梯的显示位置与实际位置一致，若屏幕分辨率不正确，则会导致电梯显示位置与实际位置不符，但不影响正常调度。