

进程管理-电梯调度-设计方案报告

学号 1851197
姓名 周楷彬
指导老师 王冬青老师
上课时间 周三五六节/周五一二节
联系方式 email: 824999404@qq.com

目录

进程管理-电梯调度-设计方案报告

目录

项目需求

功能描述

开发环境

操作说明

系统分析

单部电梯内命令处理

多部电梯外命令处理

系统设计

界面设计

状态设计

类设计

系统实现

内命令处理

外命令处理

更新电梯状态

功能实现截屏展示

项目需求

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

- 每个电梯里面设置必要功能键：如数字键、关门键、开门键、上行键、下行键、报警键、当前电梯的楼层数、上升及下降状态等。
- 每层楼的每部电梯门口，应该有上行和下行按钮和当前电梯状态的数码显示器。
- 五部电梯门口的按钮是互联结的，即当一个电梯按钮按下去时，其他电梯的相应按钮也就同时点亮，表示也按下去了。
- 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。

开发环境

- 开发平台：Windows10

- 开发软件：

1. Unity3D 2019.2.17f1

2. Visual Studio Code 1.45.0

3. .Net Standard 2.0

- **开发语言:** C#

- **主要引用:**

1. UnityEngine

2. System.Collections.Generic

3. TMPro

操作说明

注: 该电梯调度程序采用3D游戏引擎开发, 运行环境要求较高, 若运行不了, 请使用网页版:<http://106.54.28.81:1455/Elevator/index.html>

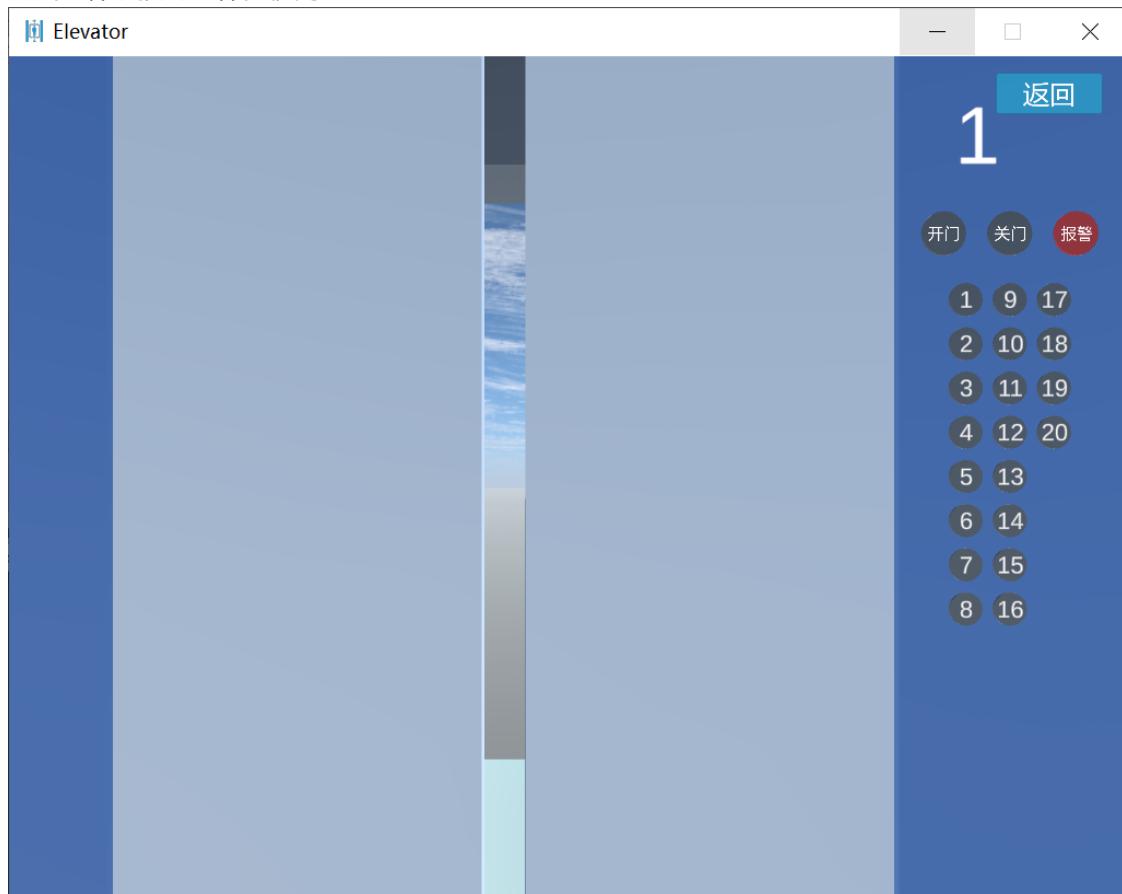
- 解压 `Elevator.zip`, 双击文件夹内的 `Elevator.exe`, 进入电梯调度程序, 如下图:



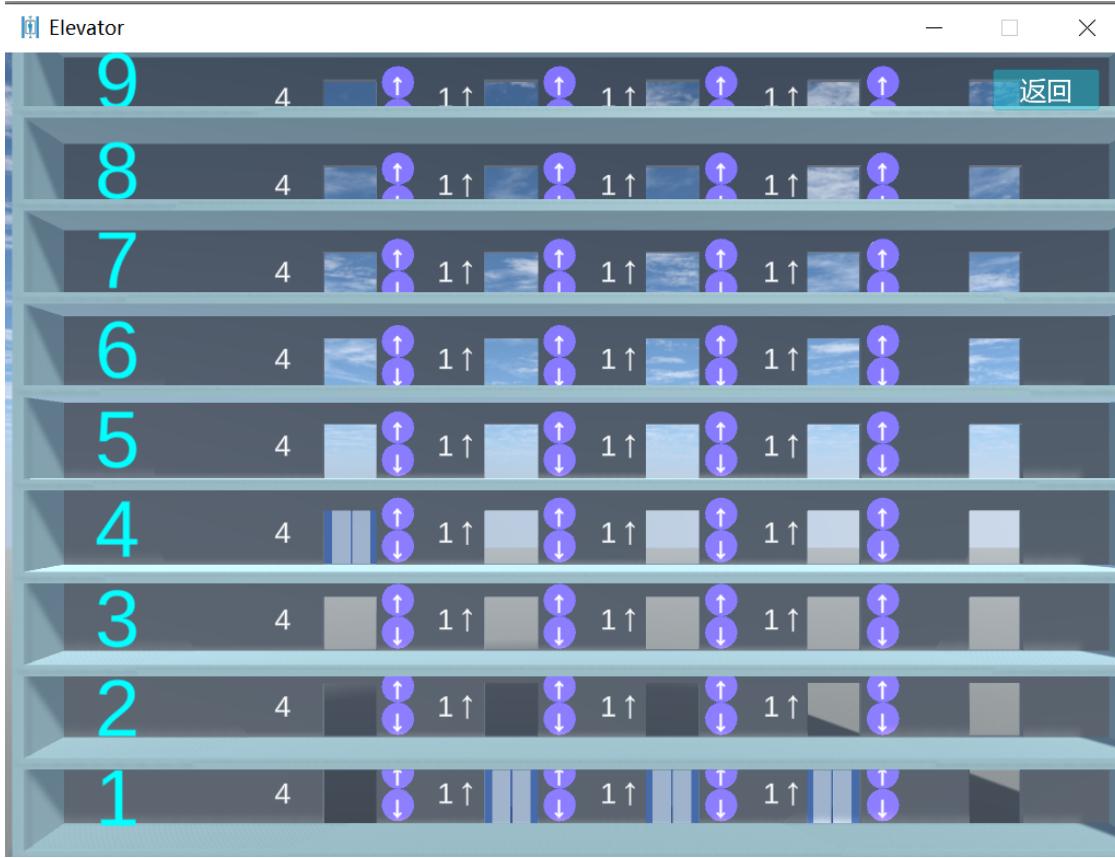
- 选择楼层数和电梯数后，点击**开始**按钮，进入电梯模拟系统



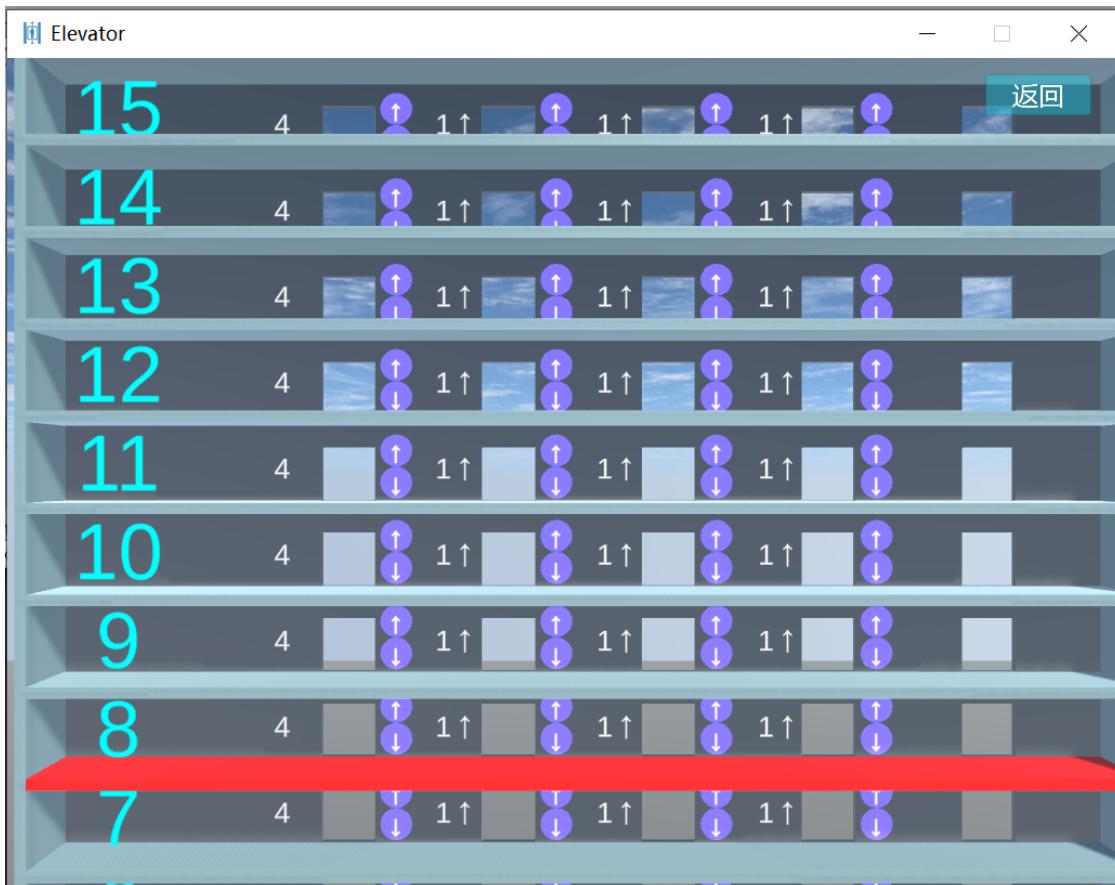
- 点击电梯切换到电梯内视角



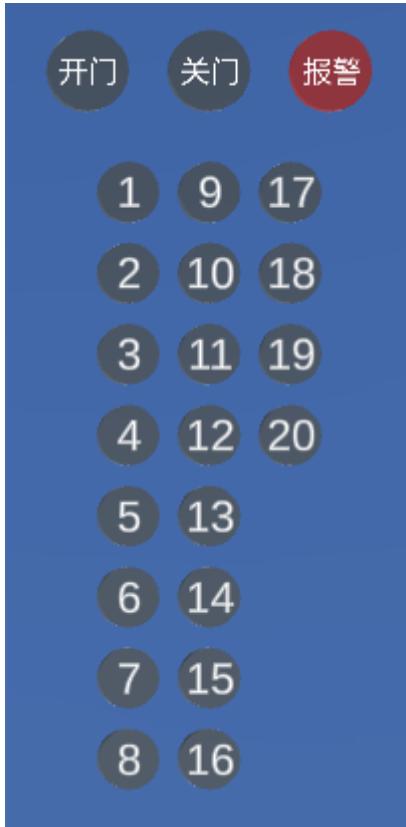
- 点击楼层切换到楼层视角



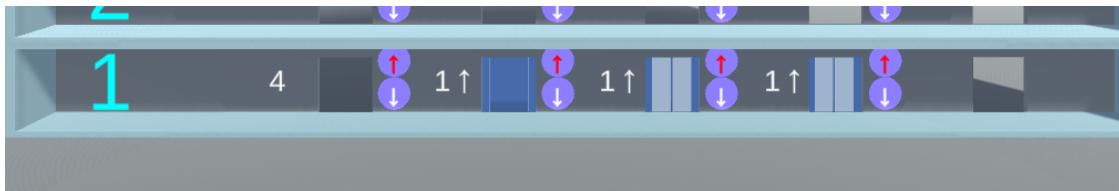
- 按**鼠标右键**或者点击右上方的**返回**按钮返回到全局视角
- 在楼层视角下，滚动鼠标滚轮或点击其它楼层上下移动摄像机



- 点击每部电梯的功能键(开门/关门, 报警器), 进行单部电梯内命令处理模拟



- 点击每层楼每部电梯门口的上/下行按钮, 进行多部电梯外命令处理模拟



系统分析

单部电梯内命令处理

- 内部事件:

- 用户点击楼层按钮
- 用户点击开/关门按钮
- 用户点击报警器

- 预期响应:

- 若按键楼层与当前楼层相同 ⇒ 该电梯开门(并等待用户自行关闭)

■ 若是在电梯运行过程瞬间点击该楼层, 则忽略该命令

- 若按键楼层与当前楼层不同:

- 若电梯为静止状态 ⇒ 将该楼层加入消息队列中

- 若电梯忙碌:

- 若电梯正在上行且按键楼层大于电梯此时楼层 ⇒ 将该楼层加入消息队列中
- 若电梯正在上行且按键楼层小于电梯此时楼层 ⇒ 将该楼层加入不顺路消息队列中
- 若电梯正在下行且按键楼层小于电梯此时楼层 ⇒ 将该楼层加入消息队列中
- 若电梯正在下行且按键楼层大于电梯此时楼层 ⇒ 将该楼层加入不顺路消息队列中

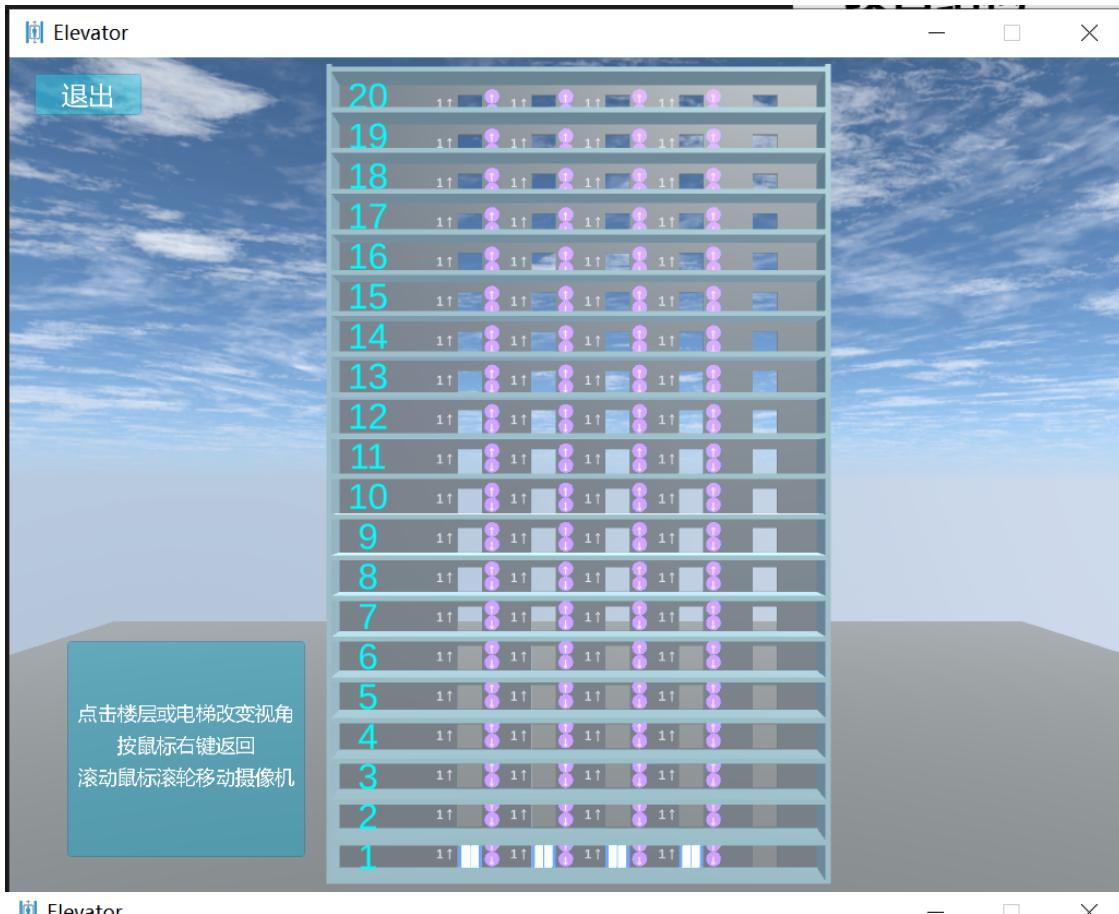
多部电梯外命令处理

- **外部事件:**
 - 用户点击任意楼层任意电梯门口的上下行按钮
- **预期响应:**
 1. 若该电梯可用，且处于该楼层，且该电梯静止⇒该电梯开门(并等待用户自行关闭)
 2. 若不满足上述条件，筛选可用的电梯 (没禁用的电梯)
 3. 计算可用电梯的可调度性:
 1. 向上顺路:
若某电梯正在向上运动并且用户选择楼层大于该电梯当前楼层 ⇒ "可调度性"定义为(用户楼层 - 该电梯当前楼层)
 2. 向下顺路:
若某电梯正在向下运动并且用户选择楼层小于该电梯当前楼层 ⇒ "可调度性"定义为(该电梯当前楼层 - 用户楼层)
 3. 某电梯静止:
若某电梯当前处于静止状态 ⇒ "可调度性"定义为(|该电梯当前楼层 - 用户楼层|)
 4. 选择可调度性最好(即值最小)的作为最佳电梯
 1. 若最佳电梯当前就在用户选择的楼层 ⇒ 该电梯开门(并等待用户自行关闭)
 2. 否则 ⇒ 将用户楼层加入该最佳电梯的消息队列中

系统设计

界面设计

1. 整体设计



2. 楼层模型:

1. 地板 Cube, GameObject, MonoBehaviour
 2. 背景墙 Cube, GameObject, MonoBehaviour
 3. 左/右侧墙 Cube, GameObject, MonoBehaviour
3. 楼层数字模型: TextMeshPro, GameObject

4. 电梯状态模型: TextMeshPro,GameObject

5. 电梯门口按钮: TextMeshPro,Cylinder,GameObject,MonoBehaviour

6. 电梯模型:

1. 四壁 Cube,GameObject,MonoBehaviour
2. 电梯门 Cube,GameObject,Animator
3. 状态和楼层 TextMeshPro,GameObject
4. 楼层按钮模型 TextMeshPro,Cylinder,GameObject,MonoBehaviour
5. 开/关门键和报警键 TextMeshPro,Cylinder,GameObject,MonoBehaviour

7. 人物模型: GameObject,Animator

8. UI: TextMeshProUGUI,Button

状态设计

1. 电梯状态(ElevatorState)

- STANDSTILL 静止状态
- RUNNING_UP 电梯上行状态
- RUNNING_DOWN 电梯下行状态

2. 门和动画状态(DoorState)

- OPEN 开门状态
- CLOSED 关门状态
- READYSTART 电梯即将启动
- READYSTOP 电梯即将停止
- NOPE 空动画

3. 用户操作(UserCommand)

- OPEN 开门
- CLOSE 关门
- GOUP 用户要上行
- GODOWN 用户要下行

类设计

1. 主要类:

1. StartScene 类: 开始界面

```
int elevatorCount=4;           //电梯数量
int floorCount=20;             //楼层数量
TextMeshProUGUI eleLabel;       //电梯数量标签
TextMeshProUGUI floorLabel;     //楼层数量标签

public void Quit();            //退出按钮点击事件
public void ChangeScene();     //开始按钮点击事件
public void eleAdd();          //增加电梯数量
public void eleMinus();        //减少电梯数量
public void floorAdd();        //增加楼层数量
public void floorMinus();      //减少楼层数量
```

2. Common 类: 用于定义状态枚举类型和公共变量

```
public enum ElevatorState    //电梯状态
{
```

```

        STANDSTILL,RUNNING_UP,RUNNING_DOWN,           //静止, 上行, 下行
    };
    public enum DoorState
    {
        OPEN,CLOSED,                                //开门, 关门
        NOPE,READYSTART,READYSTOP                 //空动画, 即将启动, 即将停止
    }
    public enum UserCommand          //用户操作
    {
        GOUP,GODOWN,                            //上行, 下行
        OPEN,CLOSE                               //开门, 关门
    };
    public int floorCount=20;                  //楼层数量
    public int elevatorCount=4;                //电梯数量

```

3. SingleElevator 类: 用于单部电梯的控制

```

public int ID;                           //电梯编号
public bool eleEnabled=true;            //电梯是否可用
public int currentFloor=1;              //当前楼层
public ElevatorState elevatorState=ElevatorState.STANDSTILL; //电梯状态
public DoorState doorState=DoorState.CLOSED; //门和动画状态
List<int> messageQueue=new List<int>(), //内部消息队列, 顺路
messageQueue_reverse=new List<int>(); //不顺路消息队列
DisplayState displayState;             //电梯楼层和状态显示类
NumberButtonsManager numberButtonsManager; //楼层按钮管理类
MultiElevator multiElevator;          //多电梯管理类
public OtherButton warnButton,openButton,closeButton; //开/关门按钮和报警器
public void doorControl(UserCommand command); //开关门动画
public void humanAnim(bool isIn);          //人物进出电梯动画
public void warnControl();                //警报键点击事件
public void eleMove(int dest);            //内命令电梯移动
void updateEleState();                  //处理消息队列, 更新电梯状态

```

4. MultiElevator 类: 用于多部电梯的调度

```

int elevatorCount=4;                  //电梯数
int floorCount=20;                   //楼层数
List<SingleElevator> elevatorList=new List<SingleElevator>(); //电梯数组
List<List<GameObject>> globalButtonList=new List<List<GameObject>>(); //电梯门口按钮数组
public void dispatch(int eleID,int floor,UserCommand command); //外命令电梯调度
public void enableButton(int floor); //启用某一楼层所有电梯门口按钮

```

2. 辅助类:

- 管理类:

1. `FloorManager` 类: 开始时创建楼层模型, 根据楼层数和电梯数调整模型大小和位置, 并用于更新每个楼层电梯状态
 2. `NumberButtonsManager` 类: 开始时根据楼层数创建电梯内部的楼层按钮, 并用于更新楼层按钮状态
- 按钮类:
 1. `GlobalButton` 类: 控制电梯门口上/下行按钮
 2. `NumberButton` 类: 控制电梯内楼层按钮
 3. `OtherButton` 类: 控制电梯内开/关门按钮和报警器
 - 其他类:
 1. `ChangeColor` 类: 鼠标指向电梯时改变颜色
 2. `ChangeView` 类: 用于切换视角
 3. `DisplayState` 类: 电梯内状态的显示
 4. `Floorview` 类: 鼠标指向楼层时改变颜色, 点击楼层时切换视角
 5. `HumanDestroy` 类: 人物动画播放完后销毁人物模型

系统实现

内命令处理

1. 报警器

- 用户点击某报警器 ⇒ 更改其颜色(响应用户) ⇒ 调用该电梯的 `singleElevator` 类的 `warnControl` 函数处理事件
- 该电梯状态设置为禁用 ⇒ 电梯关门 ⇒ 设置该电梯各部件禁用并停止所有动画 ⇒ 停用该类的内调度定时器

```
public void warnControl() //警报键
{
    eleEnabled=false; //禁用电梯
    for(int i=1;i<=floorCount;i++)
    {
        numberButtonsManager.setButtonState(i,false); //禁用楼层按钮
    }
    doorAnimator.SetTrigger("Close"); //关门动画
    doorState=DoorState.NOPE; //动画变为空状态
    elevetorState=ElevetorState.STANDSTILL; //电梯变为静止状态
    displayState.setDisabled(); //禁用电梯状态显示
    warnButton.setEnabled(false); //禁用警报键
    openButton.setEnabled(false); //禁用开门键
    closeButton.setEnabled(false); //禁用关门键
    StopAllCoroutines(); //停用定时器
}
```

2. 楼层按钮

- 用户点击某个楼层按键 ⇒ 获取楼层信息 ⇒ 改变按钮文字颜色(点击状态) ⇒ 调用该电梯的 `SingleElevator` 类的 `eleMove` 函数处理事件
- 如果按键楼层大于当前楼层:
 - 电梯处于 `STANDSTILL` 状态 ⇒ 将目标楼层加入 **消息队列**
 - 电梯正在 `RUNNING_UP` 状态 ⇒ 将目标楼层加入 **消息队列并排序**
 - 电梯正在 `RUNNING_DOWN` 状态 ⇒ 将目标楼层加入 **不顺路消息队列并排序**

- 禁用该楼层按钮
- 如果按键楼层小于当前楼层:
 - 电梯处于 STANDSTILL 状态 ⇒ 将目标楼层加入 消息队列
 - 电梯正在 RUNNING_DOWN 状态 ⇒ 将目标楼层加入 消息队列 并反向排序
 - 电梯正在 RUNNING_UP 状态 ⇒ 将目标楼层加入 不顺路消息队列 并反向排序
 - 禁用该楼层按钮
- 如果按键就为当前楼层:
 - 电梯处于 STANDSTILL 状态 ⇒ 打开门(并等待用户自行关闭)

```
//楼层按钮点击事件
void OnMouseDown()
{
    if(buttonEnabled)
    {
        text.color=Color.red;
        elevatorScript.eleMove(Number);
    }

}
```

```
//内命令电梯移动
public void eleMove(int dest)
{
    if(currentFloor<dest)          //按键大于当前楼层
    {
        if(elevatorState==ElevatorState.STANDSTILL)      //电梯处于静止状态
        {
            if(!messageQueue.Contains(dest))           //防止重复添加
            {
                messageQueue.Add(dest);                  //将目标楼层加入消息
                队列
            }
        }
        else if(elevatorState==ElevatorState.RUNNING_UP)   //电梯正在向上运行
        {
            if(!messageQueue.Contains(dest))
            {
                messageQueue.Add(dest);                  //将目标楼层加入顺路消息
                队列并排序
                messageQueue.Sort();
            }
        }
    }
    else if(elevatorState==ElevatorState.RUNNING_DOWN) //电梯正在向下运行
    {
        if(!messageQueue_reverse.Contains(dest))
        {
            messageQueue_reverse.Add(dest);             //将目标楼层加入不顺路消息
            队列并排序
            messageQueue_reverse.Sort();
        }
    }
    numberButtonsManager.setButtonState(dest, false);     //禁用目标楼层按
    钮
}
```

```

    }
    else if(currentFloor>dest)          //按键小于当前楼层
    {
        if(elevatorState==ElevatorState.STANDSTILL)      //电梯处于静止状态
        {
            if(!messageQueue.Contains(dest))
            {
                messageQueue.Add(dest);                      //将目标楼层加入消息队列
            }
        }
        else if(elevatorState==ElevatorState.RUNNING_DOWN) //电梯正在向下运行
        {
            if(!messageQueue.Contains(dest))
            {
                messageQueue.Add(dest);                      //将目标楼层加入顺路消息队列并反向排序
                messageQueue.Sort();
                messageQueue.Reverse();
            }
        }
        else if(elevatorState==ElevatorState.RUNNING_UP)   //电梯正在向上运行
        {
            if(!messageQueue_reverse.Contains(dest))
            {
                messageQueue_reverse.Add(dest);             //将目标楼层加入不顺路消息队列并反向排序
                messageQueue_reverse.Sort();
                messageQueue_reverse.Reverse();
            }
        }
        numberButtonsManager.setButtonState(dest, false);   //禁用目标楼层按钮
    }
    else                               //按键为当前楼层
    {
        if(elevatorState==ElevatorState.STANDSTILL)      //电梯静止，打开门
        {
            doorState=DoorState.OPEN;
            doorAnimator.SetTrigger("Open");              //开门动画
        }
    }
}

```

3. 开/关门

- 用户点击开/关门按钮 ⇒ 获取按键信息 ⇒ 调用该电梯的 `singleElevator` 类的 `eleMove` 函数处理事件
- 如果用户要开门:
 - 如果当前门是 `CLOSED` 或者 `NOPE` 状态并且电梯处于 `STANDSTILL` 状态 ⇒ 门的状态改为 `OPEN` ⇒ 电梯状态更新为禁用 ⇒ 播放开门动画
- 如果用户要关门:

- 如果当前门是 OPEN 或者 NOPE 状态并且电梯处于 STANDSTILL 状态 \Rightarrow 门的状态改为 CLOSED \Rightarrow 电梯状态更新为允许使用 \Rightarrow 将电梯门前的上下行按钮恢复 \Rightarrow 播放关门动画

```
//开关门事件
public void doorControl(UserCommand command)
{
    if(command==UserCommand.OPEN)          //用户要开门
    {
        if((doorState==DoorState.CLOSED||doorState==DoorState.NOPE)
        &&elevatorState==ElevatorState.STANDSTILL)      //如果门是关闭状态且电梯静止
        {
            doorState=DoorState.OPEN;                  //门状态更新为打开
            eleEnabled=false;                         //电梯不可用
            doorAnimator.SetTrigger("Open");           //开门动画
        }
    }
    else if(command==UserCommand.CLOSE)       //用户要关门
    {
        if((doorState==DoorState.OPEN||doorState==DoorState.NOPE)
        &&elevatorState==ElevatorState.STANDSTILL)      //如果门是关闭状态且电梯静止
        {
            doorState=DoorState.CLOSED;                //门状态更新为关闭
            eleEnabled=true;                          //电梯可用
            doorAnimator.SetTrigger("Close");           //关门动画
            multiElevator.enableButton(currentFloor);   //启用当前楼层电梯外按钮
        }
    }
}
```

外命令处理

1. 点击电梯门口上下行按钮

- 五部电梯门口的按钮是互联结的 \Rightarrow 用户点击了某一楼层某一电梯门口按钮 \Rightarrow 获取电梯编号 \Rightarrow 获取楼层信息 \Rightarrow 调用 MultiElevator 类的 dispatch 函数进行电梯调度

2. 电梯调度

- 若该电梯可用，且处于该楼层，且该电梯静止 \Rightarrow 该电梯开门(并等待用户自行关闭)
- 若不满足上述条件，筛选可用的电梯(没禁用的电梯) \Rightarrow 计算每部可用电梯的“可调度性” \Rightarrow 选择可调度性最好(值最小)的电梯作为最佳电梯
- 如果最佳电梯就在用户选择的楼层 \Rightarrow 该电梯开门(并等待用户自行关闭)
- 否则 \Rightarrow 加入该最佳电梯的消息队列 \Rightarrow 将用户的目标楼层按钮设定为点击状态 \Rightarrow 将该楼层所有上行或下行按钮都设置为点击状态(改变文字颜色) \Rightarrow 设置为禁用状态
- 可调度性:
 - 向上顺路: 若某电梯正在 RUNNING_UP 状态并且用户选择楼层大于该电梯当前楼层 \Rightarrow “可调度性”定义为(用户楼层 - 该电梯当前楼层)
 - 向下顺路: 若某电梯正在 RUNNING_DOWN 状态并且用户选择楼层小于该电梯当前楼层 \Rightarrow “可调度性”定义为(该电梯当前楼层 - 用户楼层)
 - 某电梯静止: 若某电梯当前处于 STANDSTILL 状态 \Rightarrow “可调度性”定义为(|该电梯当前楼层 - 用户楼层|)

```

//电梯调度
public void dispatch(int eleID,int floor,UserCommand command)
{
    if(elevatorList[eleID].enabled) //如果该电梯在当前楼层，且处于
静止状态，则直接开门
    {
        singleElevator elevator=elevatorList[eleID];

        if(elevator.elevatorState==ElevatorState.STANDSTILL&&elevator.currentFloor-
=floor)
        {
            elevator.doorControl(UserCommand.OPEN);
            return;
        }
    }
    List<SingleElevator> EnabledList=new List<SingleElevator>();
    foreach(SingleElevator elevator in elevatorList) //筛选可用
电梯
    {
        if(elevator.eleEnabled)
        {
            EnabledList.Add(elevator);
        }
    }
    int bestIndex=0;
    List<int> dist=new List<int>(); //可使用电梯距离用户的距离
    for(int i=0;i<EnabledList.Count;i++)
    {
        if(EnabledList[i].elevatorState==ElevatorState.RUNNING_UP&&
command==UserCommand.GOUP&&
floor>EnabledList[i].currentFloor) //向上顺路
        {
            dist.Add(floor-EnabledList[i].currentFloor);
        }
        else if(EnabledList[i].elevatorState==ElevatorState.RUNNING_DOWN&&
command==UserCommand.GODOWN&&
floor<EnabledList[i].currentFloor) //向下顺路
        {
            dist.Add(EnabledList[i].currentFloor-floor);
        }
        else if(EnabledList[i].elevatorState==ElevatorState.STANDSTILL)//电梯静
止
        {
            dist.Add(Mathf.Abs(EnabledList[i].currentFloor-floor));
        }
        else
        {
            dist.Add(100);
        }
        if(i!=bestIndex)
        {
            if(dist[i]<dist[bestIndex]) //寻找最短距离电梯
            {
                bestIndex=i;
            }
        }
    }
}

```

```

if(dist[bestIndex]==0) //最佳电梯在当前楼层
{
    EnabledList[bestIndex].doorControl(UserCommand.OPEN); //打开门等待用户关闭
}
else
{
    EnabledList[bestIndex].eleMove(floor); //加入最佳电梯消息队列
    if(command==UserCommand.GROUP) //禁用该楼层所有上/下行按钮
    {
        //禁用上行按钮
        foreach(GameObject globalButton in globalButtonList[floor-1])
        {
            globalButton.transform.GetChild(0).GetComponent<GlobalButton>().setEnabled(false);
        }
    }
    else
    {
        //禁用下行按钮
        foreach(GameObject globalButton in globalButtonList[floor-1])
        {
            globalButton.transform.GetChild(1).GetComponent<GlobalButton>().setEnabled(false);
        }
    }
}
}

```

更新电梯状态

1. 定时器

- 单部电梯内采用定时器(1s调用一次方法更新电梯状态)

```
//创建定时器，1s更新一次电梯状态
InvokeRepeating("updateEleState", 1, 1);
```

2. 更新电梯状态

- 电梯的消息队列不为空
 - 如果电梯门是打开的 ⇒ 等待电梯关门
 - 电梯处于 STANDSTILL 状态 ⇒ 播放开门动画 ⇒ 播放人物进门动画 ⇒ 根据即将运行的方向更新电梯状态 ⇒ 人物进门动画播放完后，动画变为 READYSTART 状态
 - 动画处于 READYSTART 状态 ⇒ 播放关门动画 ⇒ 动画变为 NOPE 状态
 - 动画处于就绪 READYSTOP 状态 ⇒ 从消息队列删除该楼层 ⇒ 启用该电梯的该楼层按钮 ⇒ 启用该楼层电梯门口上/下行按钮 ⇒ 播放关门动画 ⇒ 动画变为 NOPE 状态 ⇒ 电梯变为 STANDSTILL 状态
 - 除此之外 ⇒ 从消息队列中获取第一个目标楼层

- 向上运动: 当前楼层小于目标楼层 \Rightarrow 电梯状态变为 RUNNING_UP \Rightarrow 当前楼层加一
- 向下运动: 当前楼层大于目标楼层 \Rightarrow 电梯状态变为 RUNNING_DOWN \Rightarrow 当前楼层减一
- 电梯到达目的地: 播放开门动画 \Rightarrow 播放人物出电梯动画 \Rightarrow 动画变为 READYSTOP 状态
- 更新电梯内状态和当前楼层的显示
- 某个电梯的消息队列为空 & 不顺路消息队列不为空 \Rightarrow 交换两个队列
- 如果电梯处于 STANDSTILL 状态 \Rightarrow 启用报警器
- 否则, 电梯处于运行状态 \Rightarrow 禁用报警器

```

void updateElevatorState() //处理消息队列, 更新电梯状态
{
    if(!eleEnabled) return; //电梯已被禁用
    if(messageQueue.Count!=0)
    {
        if(doorState==DoorState.OPEN) return; //如果电梯门开着, 等待关门
        if(elevatorState==ElevatorState.STANDSTILL) //电梯静止
        {
            doorState=DoorState.OPEN; //门状态更新为打开
            doorAnimator.SetTrigger("Open"); //开门动画
            humanAnim(true);
            if(currentFloor<messageQueue[0]) //根据即将运行方向更新
电梯状态
            {
                elevatorState=ElevatorState.RUNNING_UP;
            }
            else if(currentFloor>messageQueue[0])
            {
                elevatorState=ElevatorState.RUNNING_DOWN;
            }
            // doorState=DoorState.READYSTART; //就绪启动状态, 人物动画
播放完后设置
        }
        else if(doorState==DoorState.READYSTART) //处于就绪启动状态
        {
            doorAnimator.SetTrigger("Close"); //关门动画
            doorState=DoorState.NOPE; //动画变为空状态
        }
        else if(doorState==DoorState.READYSTOP) //处于就绪停止状态
        {
            int dest=messageQueue[0];
            messageQueue.RemoveAt(0); //从消息队列删除目标楼层
            numberButtonsManager.setButtonState(dest,true); //启用目标
楼层按钮
            multiElevator.enableButton(dest); //启用目标楼层
电梯外按钮
            //TODO: 判断之前上行还是下行, 只启用一个键
            doorAnimator.SetTrigger("Close"); //关门动画
            doorState=DoorState.NOPE; //动画变为空状态
            elevatorState=ElevatorState.STANDSTILL; //电梯变为静止状态
        }
        else //电梯处于移动状态
        {
            if(!isArrived) return; //还没到达下一楼层
            int destFloor=messageQueue[0]; //获取第一个目标楼层
        }
    }
}

```

```

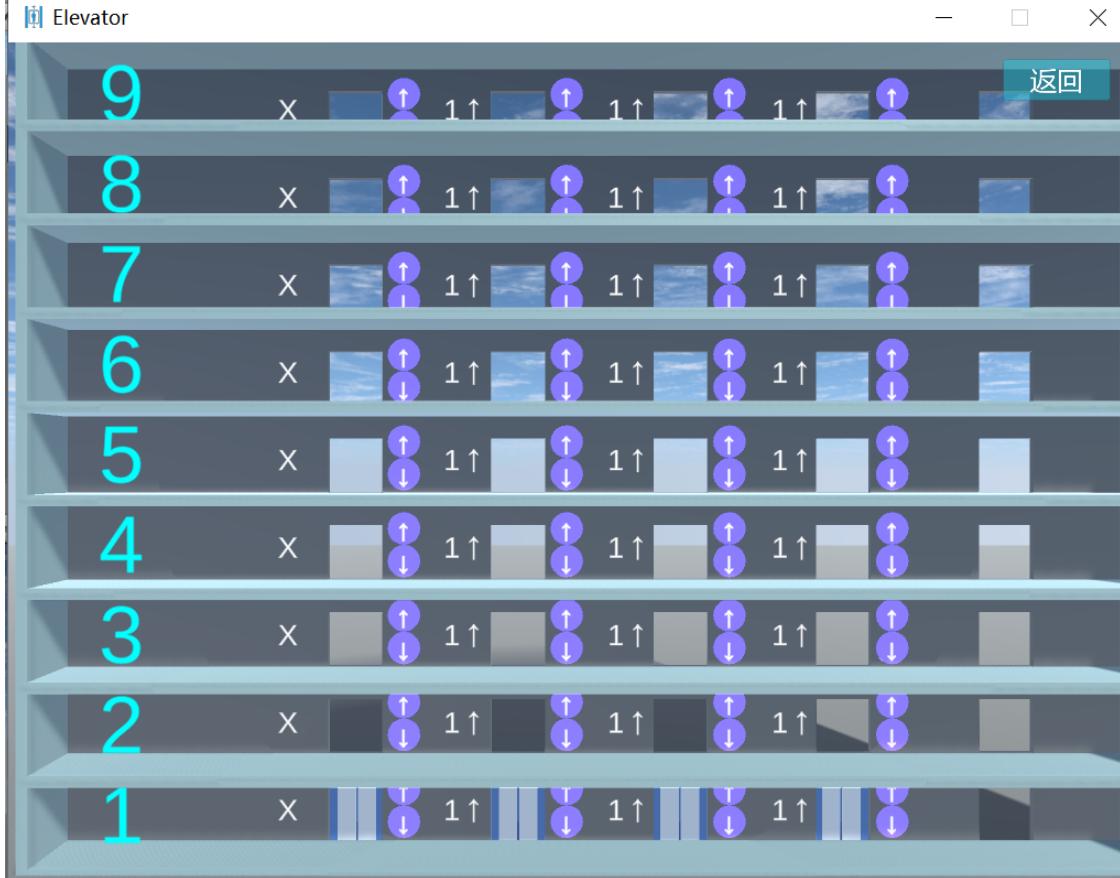
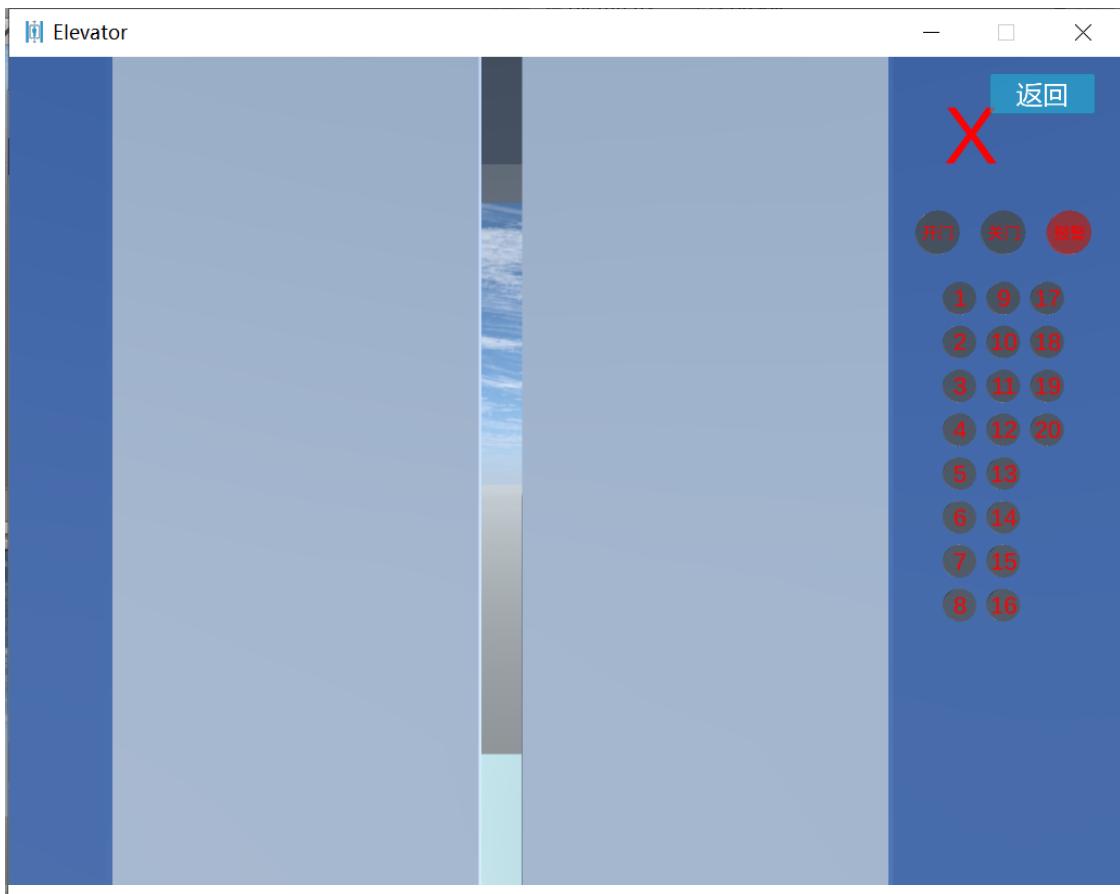
        if(currentFloor<destFloor)           //向上运动
        {
            elevatorState=ElevatorState.RUNNING_UP;
            // currentFloor++;
            nextFloor=currentFloor+1;          //下一楼层为当前楼层加一
            isArrived=false;
        }
        else if(currentFloor>destFloor)       //向下运动
        {
            elevatorState=ElevatorState.RUNNING_DOWN;
            // currentFloor--;
            nextFloor=currentFloor-1;          //下一楼层为当前楼层减一
            isArrived=false;
        }
        else                                //到达目标楼层
        {
            doorState=DoorState.OPEN;
            doorAnimator.SetTrigger("Open"); //开门动画
            humanAnim(false);
            // doorState=DoorState.READYSTOP; //就绪停止状态，人物动画播
放完后设置
        }
    }
    displayState.setFloor(currentFloor);      //在电梯内显示当前楼层
    displayState.setState(elevatorState);      //在电梯内显示当前状态
}
else if(messageQueue_reverse.Count!=0)      //如果顺路消息队列为空，不顺路消
息队列不为空
{
    messageQueue=new List<int>(messageQueue_reverse.ToArray()); //交换
两个队列
    messageQueue_reverse.Clear();
}

if(elevatorState==ElevatorState.STANDSTILL)   //电梯在运行过程中禁止点击报
警键
{
    warnButton.setEnabled(true);
}
else
{
    warnButton.setEnabled(false);
}
}

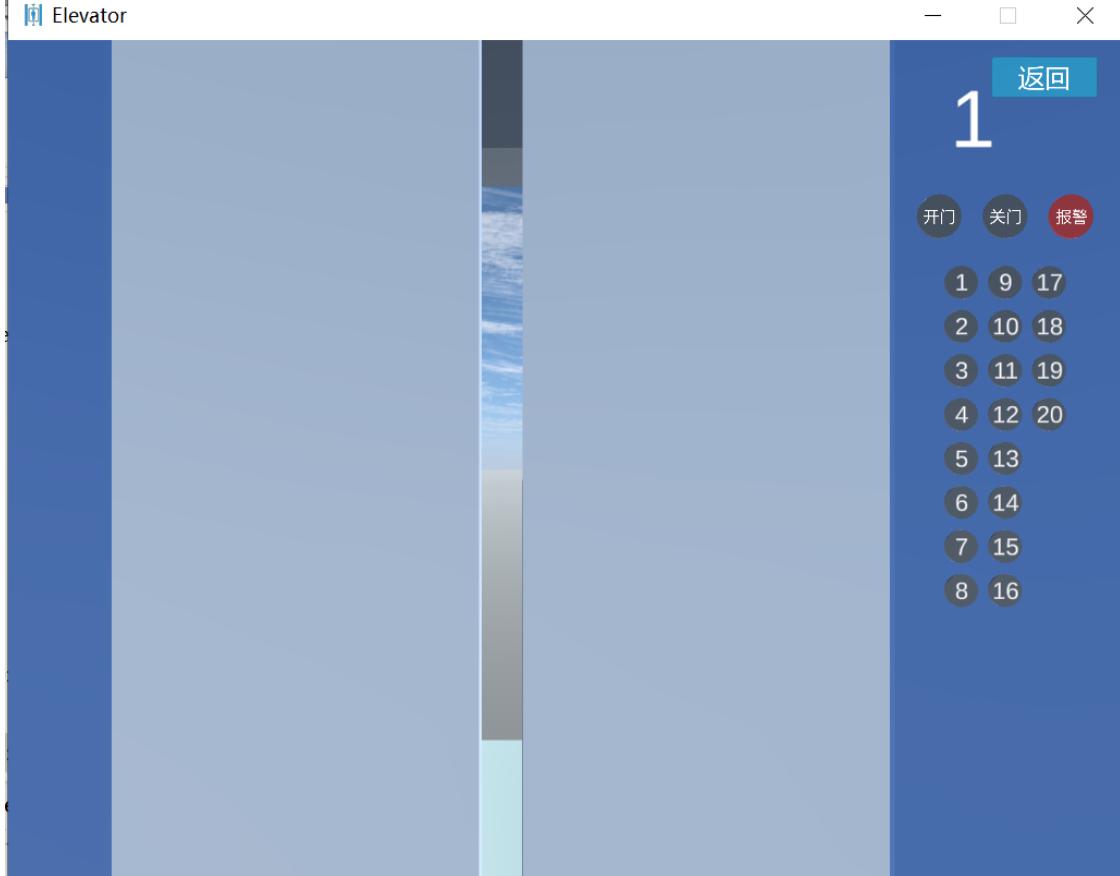
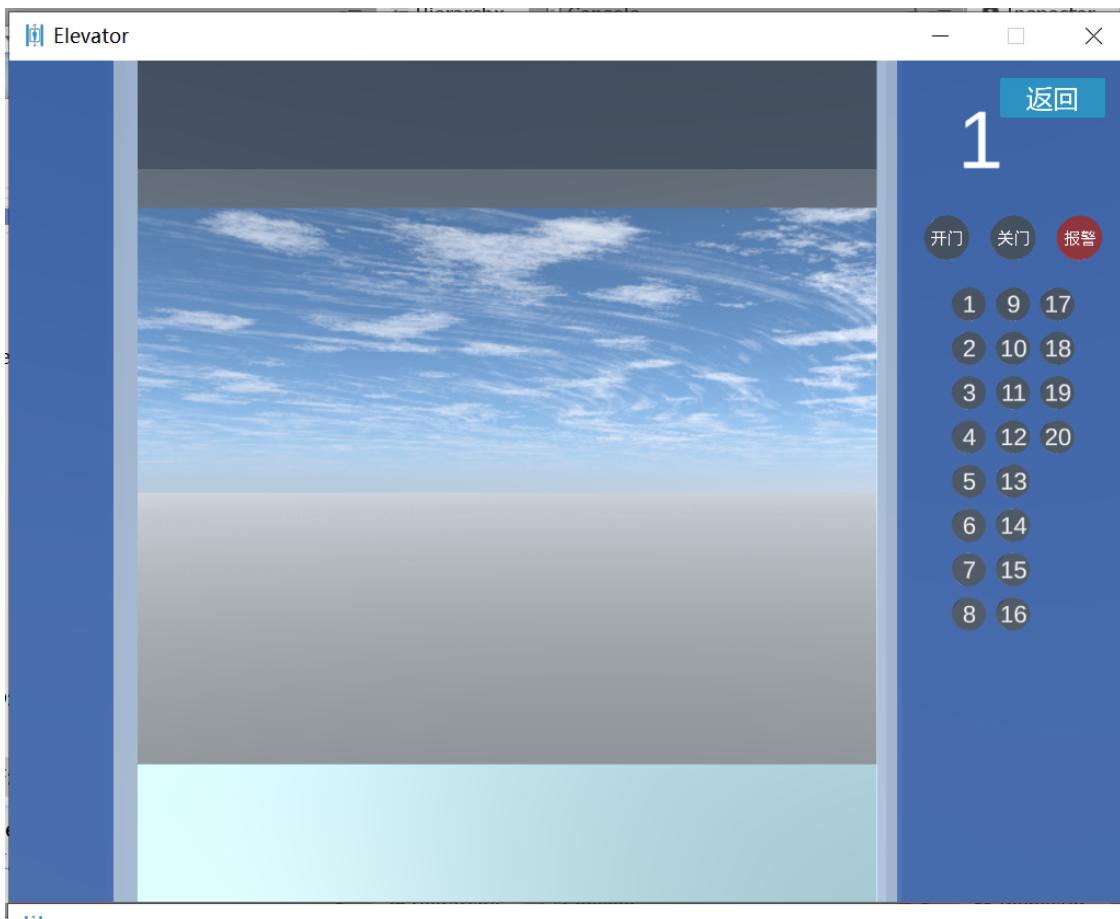
```

功能实现截屏展示

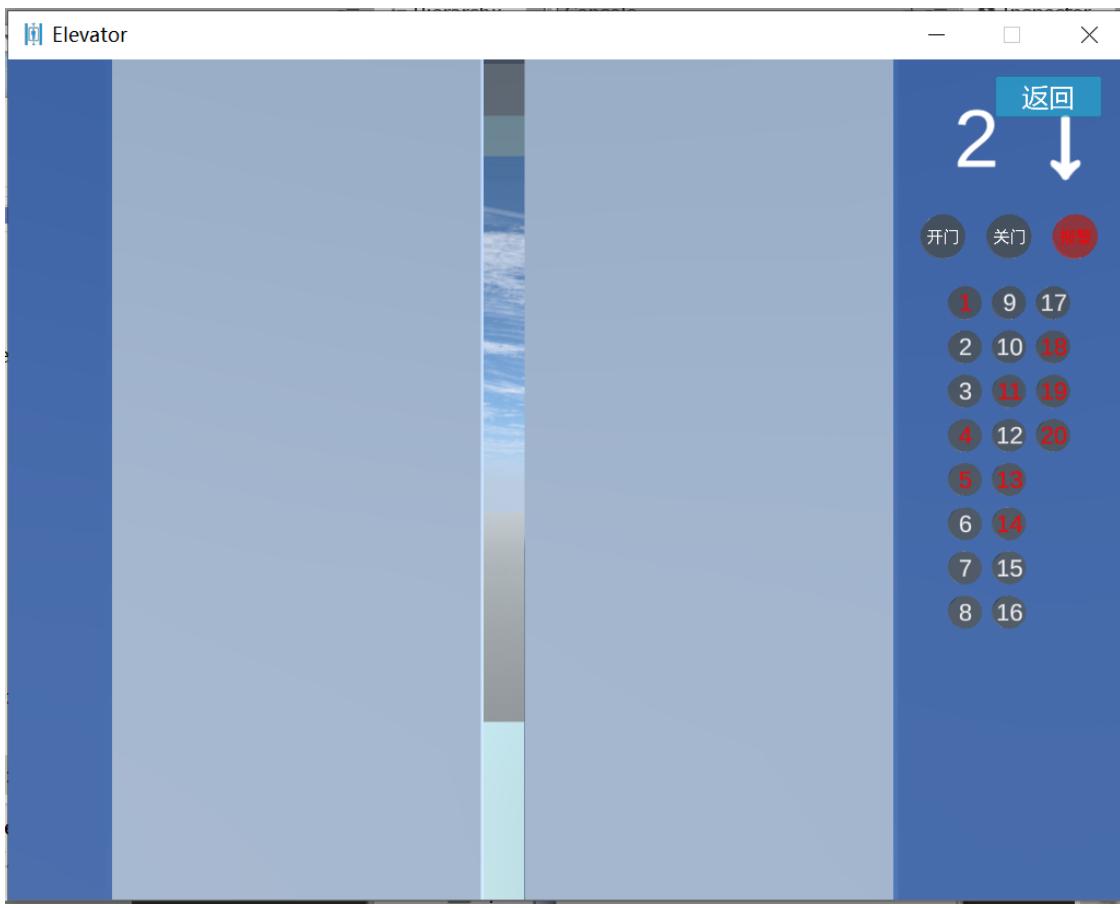
1. 报警器功能展示



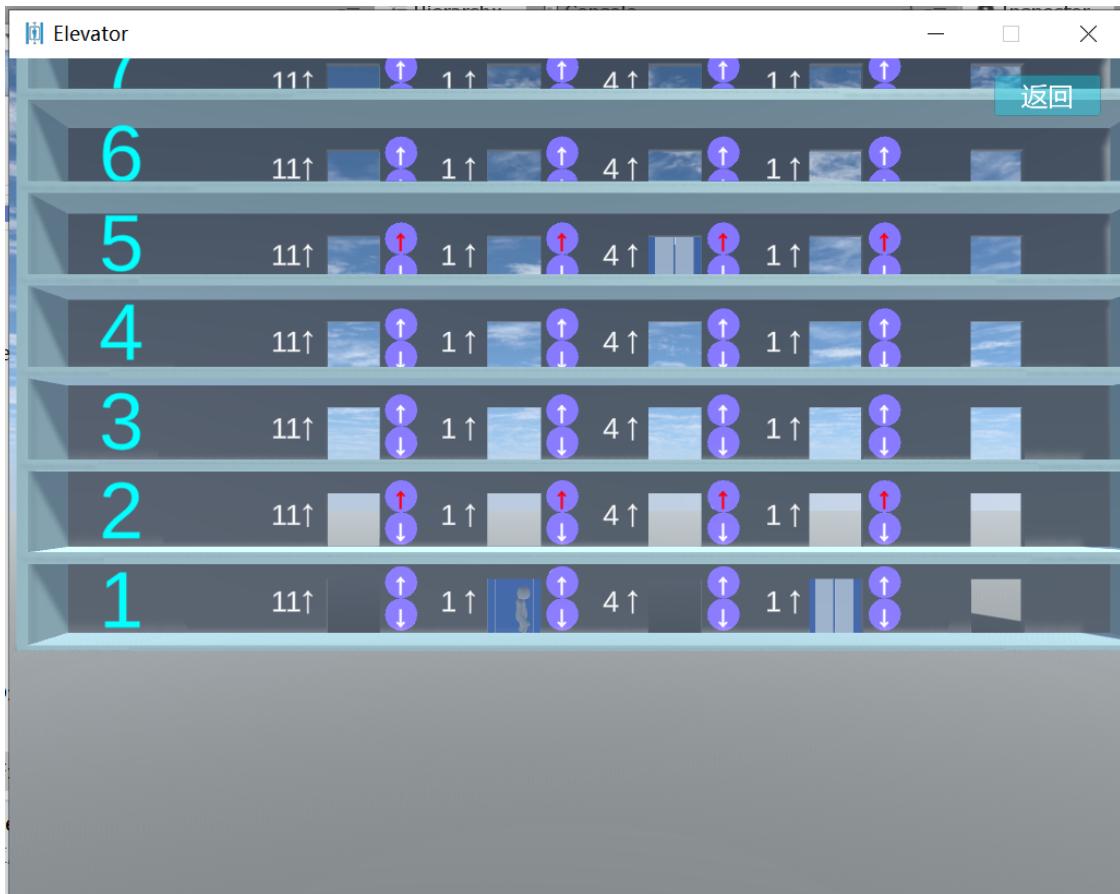
2. 开/关门功能展示



3. 内部指令处理



4. 外部指令处理与电梯调度



5. 上下行按钮联结

