

电梯调度

操作系统第一次作业

李文玥 1750803

项目背景

1. 20 层楼，每层楼可呼叫电梯，去往上行（或下行）方向楼层。
2. 5 部电梯，每部电梯内有 1~20 层楼的按钮，供电梯内乘客选择目的楼层。

开发/运行环境

开发语言：html、Javascript

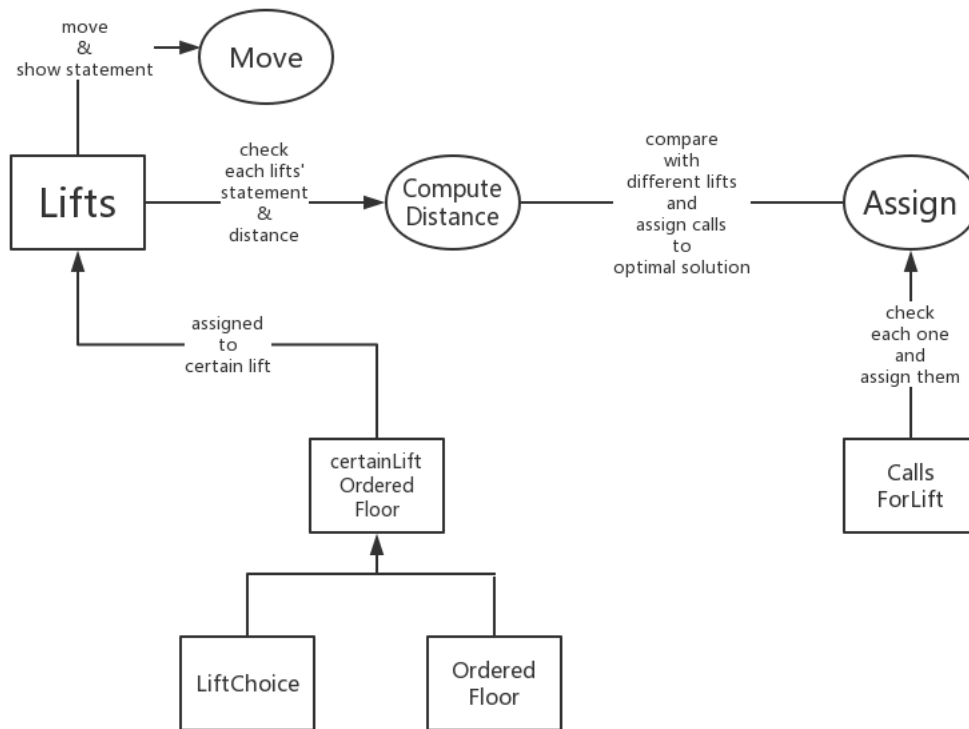
运行环境：Google Chrome

项目设计简述

1. 项目架构

- 界面
 - 电梯
 - 电梯选择按钮
 - 电梯楼层选择按钮
 - 电梯状态可视化显示
 - 楼层
 - 呼叫电梯按钮（上行）
 - 呼叫电梯按钮（下行）
- 调度算法
 - 内部调度算法：计算本部电梯是否可用
 - 电梯选择按钮
 - 电梯楼层选择按钮
 - 电梯状态可视化显示
 - 外部调度算法：将电梯请求分配给电梯
 - 电梯内部客人到达楼层请求
 - 电梯外部楼层呼叫电梯请求

2. 架构图解



3. 电梯状态转移说明

1. WAIT 状态由于上行需求，转为 UP 状态
2. WAIT 状态由于下行需求，转为 DOWN 状态
3. UP 状态由于到达某一下客楼层，转为 ARRIVE 状态
4. UP 状态由于到达某一上客楼层，转为 FETCH 状态
5. DOWN 状态由于到达某一下客楼层，转为 ARRIVE 状态
6. DOWN 状态由于到达某一上客楼层，转为 FETCH 状态
7. ARRIVE 状态由于上行需求，转为 UP 状态
8. ARRIVE 状态由于下行需求，转为 DOWN 状态
9. FETCH 状态由于上行需求，转为 UP 状态
10. FETCH 状态由于下行需求，转为 DOWN 状态
11. ARRIVE 状态由于当前无请求，转为 WAIT 状态
12. FETCH 状态由于当前无请求，转为 WAIT 状态
13. WAIT 状态由于当前楼层为下客楼层，转为 ARRIVE 状态（电梯内开门按钮）
14. WAIT 状态由于当前楼层为上客楼层，转为 FETCH 状态（电梯外开门按钮）

4. 外部调度算法说明

- 电梯内部客人到达楼层请求调度

1. 首先客人选中乘坐的电梯，刷新按钮设置（锁死已选电梯按钮，避免重复选择）
2. 选择电梯内楼层，刷新按钮设置（锁死已选楼层按钮，避免重复选择）
 - 电梯正在运动中，按键必须为当前方向的楼层，否则按钮不响应
 - 电梯处于等待状态，可选任意楼层
3. 将选中楼层压入当前电梯送客楼层数组中，并按大小顺序将数组中的楼层数排序
4. 到达楼层后，删除数组中的楼层，并刷新按钮设置

[代码]

```
changeLiftChoice=(lift_choice_id)=>{ //选中电梯
  lift_choice=lift_choice_id;
  reloadChoiceButton();
}

reloadChoiceButton=()=>{//刷新按钮
  for (let i = 0; i < 5; i++) {
    $("#choice-
buttons").children().eq(i).children().eq(0).attr("disabled", false);
  }
  $("#choice-
buttons").children().eq(lift_choice).children().eq(0).attr("disabled",
true);
  for (let i = 0; i < 20; i++) {
    $("#floor-buttons" + parseInt(i / 5)).children().eq(i %
5).children().eq(0).attr("disabled", false);
  }
  console.log("now we are taking elevator no.")
  console.log(lifts[lift_choice].receivedCallFloors);
  for (let i in lifts[lift_choice].receivedCallFloors) {
    let j = floor_num - lifts[lift_choice].receivedCallFloors[i];
    console.log("disabled:")
    console.log(j);
    $("#floor-buttons" + parseInt(j / 5)).children().eq(j %
5).children().eq(0).attr("disabled", true);
  }
}

pushReceivedCallFloors=(floor)=>{ //选中楼层进入当前电梯送客楼层数组
  lifts[lift_choice].receivedCallFloors.push(floor);
  lifts[lift_choice].receivedCallFloors.sort();
  reloadChoiceButton();
  console.log("call floor no.")
  console.log(lifts[lift_choice].receivedCallFloors);
}
```

```

}

addReceivedFloors=(floor)=>{ //选中楼层
  console.log("Where are we going?");
  if(lifts[lift_choice].state===State['WAIT'])
  {
    pushReceivedCallFloors(floor);
  }
  else if(lifts[lift_choice].state===State['UP']){
    if(lifts[lift_choice].currentFloor<floor){
      pushReceivedCallFloors(floor);
    }
    else{
      console.log("NO WAY!");
    }
  }
}
else if(lifts[lift_choice].state===State['DOWN']){
  if(lifts[lift_choice].currentFloor>floor){
    pushReceivedCallFloors(floor);
  }
  else{
    console.log("NO WAY!");
  }
}
}
};

```

- 电梯外部楼层呼叫电梯请求调度

1. 客人选择当前楼层上行或下行，刷新按钮设置（锁死已选楼层方向按钮，避免重复呼叫）
2. 将呼叫压入字典中
3. 循环字典，调度每一条请求，查看每一部电梯，调用电梯内部调度函数。比较电梯之间调度结果，设置请求分配状况，与分配的电梯

[代码]

```

assign=(key)=>{
  let bestIndex = -1;
  mindistance=1000;
  for (let liftIndex = 0; liftIndex < lifts.length; liftIndex++) {
    let lift = lifts[liftIndex];
    let call=callDict[key];
    let distance = lift.computeDistance(call); //内部调度算法
    console.log(distance)
    if (distance>-1) {
      if (bestIndex > -1) {
        if (distance<mindistance) {

```

```

        bestIndex = liftIndex;
        mindistance=distance;
        console.log("close");
    }
    console.log("far");
} else {
    bestIndex = liftIndex;
    mindistance=distance;
}
}
}
if (bestIndex > -1) {
    let bestLift = lifts[bestIndex];
    if (bestLift.targetCallKey > -1) {
        callDict[bestLift.targetCallKey].isAssigned = false; //一部
        电梯任何时候只有一个 targetCallKey
    }
    bestLift.targetCallKey = key; //把该请求加入到选中电梯的
    targetcallkey 中
    callDict[key].isAssigned = true;
}
}
}

```

5.内部调度算法说明

1. 计算当前楼层与目标楼层的距离
2. 根据电梯状态,
 - 1) 等待状态, 返回距离*0.5, 认为等待状态的电梯更应该去接客, 避免某部电梯过于繁忙
 - 2) 当前电梯运行方向与呼叫楼层方向相同, 楼层包含在当前电梯楼层与最终目标楼层区间内, 返回距离
 - 3) 其余情况均返回不可调度, 即-1

[代码]

```

computeDistance(call){
    let callFloor = call.floor;
    let distance=Math.abs(callFloor-this.currentFloor);
    if(this.state===State['WAIT']){
        return Math.abs(callFloor-this.currentFloor)*0.5; //等待状态
        电梯自动乘 0.5, 防止一辆电梯太过繁忙
    }
    else if (this.targetCallKey === -1) {//没有呼叫, 但有送客楼层
        let lastFloor=this.lastFloor();
    }
}

```

```

        if (this.state === 1 && callFloor < lastFloor && callFloor >
this.currentFloor) {
            return distance;
        }
        else if (this.state === -1 && callFloor >lastFloor &&
callFloor < this.currentFloor) {
            return distance;
        }
        else {
            return -1;    //不可被调度
        }
    }
    else{
        let currentCallFloor;
        if(!(this.receivedCallFloors.length)){//只有呼叫楼层
            currentCallFloor = callDict[this.targetCallKey].floor;
        }
        else{//呼叫楼层+送客楼层
            let lastFloor=this.lastFloor();
            let lastCallFloor=callDict[this.targetCallKey].floor;
            if(this.state===1){
                if(lastFloor>=lastCallFloor){
                    currentCallFloor=lastFloor;
                }
                else{
                    currentCallFloor=lastCallFloor;
                }
            }
            else if(this.state===-1){
                if(lastFloor<=lastCallFloor){
                    currentCallFloor=lastFloor;
                }
                else{
                    currentCallFloor=lastCallFloor;
                }
            }
        }
        if ((callFloor - currentCallFloor) * (currentCallFloor -
this.currentFloor) < 0) {
            return distance;
        }
        else {
            return -1;    //不可被调度
        }
    }

```





























```
    }  
};  
  
lastFloor(){  
    if(this.state===1){  
        let i=this.receivedCallFloors.length-1;  
        return this.receivedCallFloors[i];  
    }  
    if(this.state===-1){  
        return this.receivedCallFloors[0];  
    }  
};
```

6.类的设计

- 电梯类 (class Lift)
包含成员：电梯 id, 当前电梯楼层, 相应呼叫字典关键字, 目的楼层序列, 运行状态状态
包含函数：内部调度函数、电梯可视化运行函数。
- 呼叫类 (class Call)
包含成员：呼叫楼层, 呼叫方向, 指派状态

电梯使用

1. UI 设计

电梯0	电梯1	电梯2	电梯3	电梯4	按钮	
20	20	20	20	20		
19	19	19	19	19		
18	18	18	18	18		
17	17	17	17	17		
16	16	16	16	16		
15	15	15	15	15		
14	14	14	14	14		
13	13	13	13	13		
12	12	12	12	12		
11	11	11	11	11		
10	10	10	10	10		
9	9	9	9	9		
8	8	8	8	8		
7	7	7	7	7		
6	6	6	6	6		
5	5	5	5	5		
4	4	4	4	4		
3	3	3	3	3		
2	2	2	2	2		
1	1	1	1	1		
选择楼层	选择楼层	选择楼层	选择楼层	选择楼层		
20	19	18	17	16		
15	14	13	12	11		
10	9	8	7	6		
5	4	3	2	1		

（由于只有一个楼层数字面板，所以通过选择电梯（原则楼层按钮），来选中当前数字面板归属哪部电梯）

2. 功能操作

- 去往特定楼层
选中电梯->选择到达层->电梯可视化运动（通过颜色改变表征）->电梯到达（通过颜色表征）
- 呼叫电梯
选中楼层方向->电梯可视化运动->电梯接到乘客

3. 可视化颜色说明

