### **基于冬筑实际情况的电梯算法优化**

提交人：禹康

学号：2024150072

#### **Abstract**

基于冬筑每个电梯间在高峰期常常出现电梯间大量人员等电梯的情况，做了针对三台电梯的算法优化。参考了前人的算法，但是由于网上的参考资料大多基于一个电梯，因此做出改进：基于LOOK算法和群控电梯调度的思想，可以解决每层楼都太多人，16楼等到电梯的时候发现已经满员的情况。并尽可能让电梯用最短的时间把所有有需要的楼层中的所有人都运往1层。

1. **背景介绍 (Introduction)**

当前冬筑在用电梯高峰期（如下午两点，早上八点），冬筑16楼在等电梯时常常发现，等了好久的电梯已经满员了，但是还是上了16楼，既耽误了13楼、10楼的同学下楼，也没让16楼的同学上电梯，因此做了算法优化，确保最高楼的同学有电梯坐的同时，可以在未满员的情况下接纳低楼层的同学。

### **2. 相关研究 (Related Research)**

**1）先来先服务算法（FCFS，FCFS-First Come First Serve）**

**按照请求顺序决定电梯去往楼层的算法。**

例：一个在1楼的电梯。有3个人分别在2楼，3楼按下请求，则电梯会先去2楼，接到人之后处理2楼乘客的请求，再去3楼，接到人之后处理3楼乘客的请求。

**优点：每个请求平等，在电梯运行速度足够快的情况下，不存在有楼层经常等不到电梯的情况。**

**缺点：处理逻辑简单，效率低，并且不兼容多层楼去同一个目的地的情况，现在已经基本不使用该方法，更多是作为评判一个电梯的算法标准。**

例：一个在3楼的电梯，先有一个人在16楼按了请求，又有一个人在1楼按了请求，则电梯会先上16楼再去1楼。

**2）最短寻找楼层时间优先算法(SSTF，SSTF-Shortest Seek Time First)**

#### **按照电梯的当前楼层，比较请求楼层的距离，先处理距离近得楼层的请求。**

例：一个在3楼的电梯，先有一个人在16楼按了请求，又有一个人在1楼按了请求。若当前电梯仍在3楼，则会先处理1楼请求再处理16楼请求。

**优点：效率相较于FCFS有所提高，平均处理时间最短。**

**缺点：会发生一个楼层一直等不到电梯的情况。并且电梯会一直变方向，不利于乘客预测电梯运行情况。这样的设计不贴合当前电梯的按钮设计，即每个楼层都有向上或向下的按钮。**

例：有人在16楼发出请求，同时1，2，3楼都有人按下请求，则16楼的人的请求会被最后处理。

1. **扫描算法(SCAN)**

**在最顶层和最底层之间来回移动，中间有同方向的请求就接受，不接受异方向的请求。**

例：一个从1楼去往16楼的电梯，3，6楼都发出向上请求，接受；9楼发出向下请求，不接受，等下一班。

**优点：相比SSTF算法不会出现上述的一直等不到电梯的情况，处理请求的方差更小，更符合当前的电梯按钮的规则。**

**缺点：直到顶层或底层才停止（不是一个实时算法），无论电梯是否有人，耗电并且效率降低。**

例：1，2，3楼的乘客都去往10楼，全下了之后有人在1楼发出请求，但是电梯仍然前往16楼之后才会掉头前往1楼。

1. **LOOK 算法**

**基于SCAN算法的改进，是目前电梯普遍使用的算法，当当前的请求已经终止则不再继续运行直到得到下一个请求。**

例：1，2，3楼的乘客都去往10楼，全下了之后电梯将会在原楼层停止，如果此时有人在1楼发出请求，则会立刻前往1楼。

**优点：去除了SCAN算法的缺点，更贴合当前主流电梯的运行逻辑。**

**缺点：在启动后方向就不能改变，可能造成效率降低。**

例：3楼的电梯先得到16楼的人发出的请求，1s后1楼有人发出请求，但是此时电梯已经启动前往16楼，不会前往1楼。

### **3. 算法与解决方法 (Proposed Algorithm and Solution)**

### 当前冬筑也是使用LOOK算法，但是当电梯均在1楼，并且10,13,16楼同时发出请求时，电梯会逐次在10,13,16楼停下接客。但是10楼+13楼的乘客在高峰期很容易就让电梯直接满载，导致16楼乘客上不了电梯。

针对这个情况，如果使用“满载直接下楼”的解决方法，会导致16楼出现SSTF算法的饿死情况。

所以此处我们应该先去往请求楼层的最高层，然后向下运行，有其他请求乘客就接上，此时满载就直接下楼不停止。

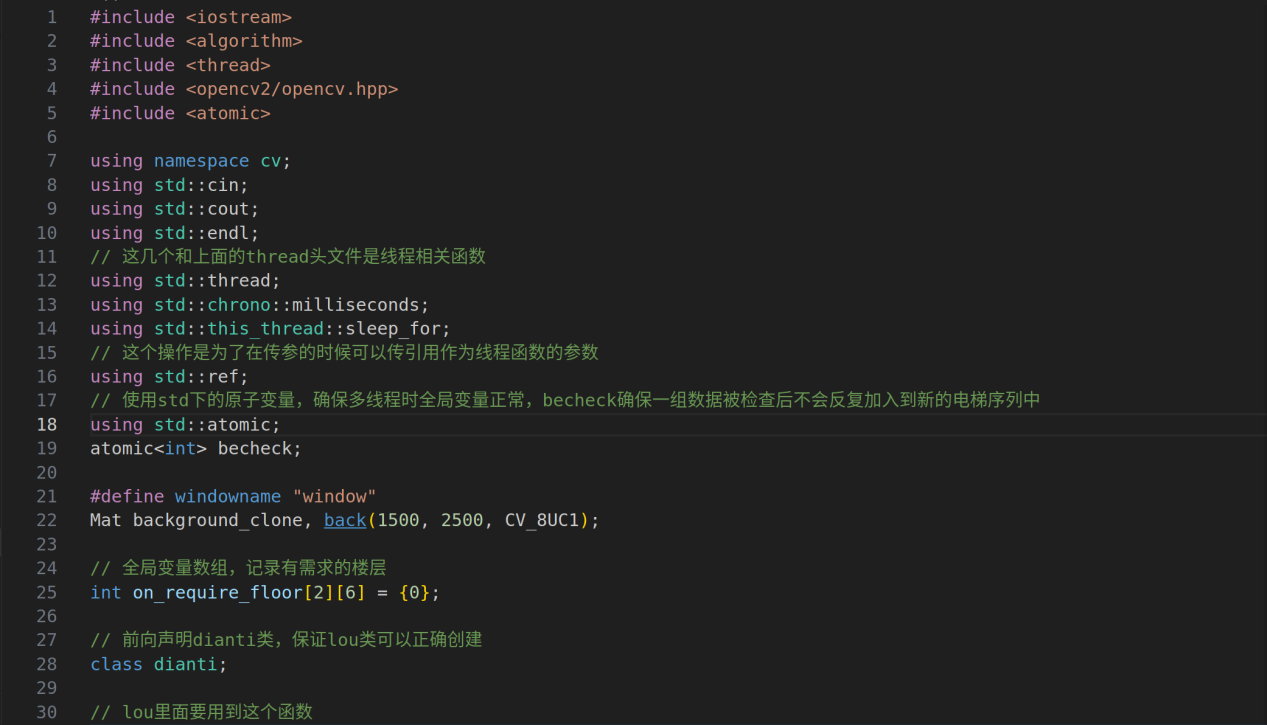
借鉴群控电梯调度算法的“下行潮”算法，此时所有楼层都是向下运行，应该在一部电梯确定去往16楼之后，后续的电梯直接前往13楼，再后续的电梯前往10楼，并在接到乘客后下楼。理论来说，应该让三部电梯分别去往16，13，10楼，此时三部电梯都可以向下兼容其他楼层的乘客。如果不能兼容，则需要在结束本次运行后根据队列中请求，处理最先请求的楼层。（本次代码中不考虑超载问题）

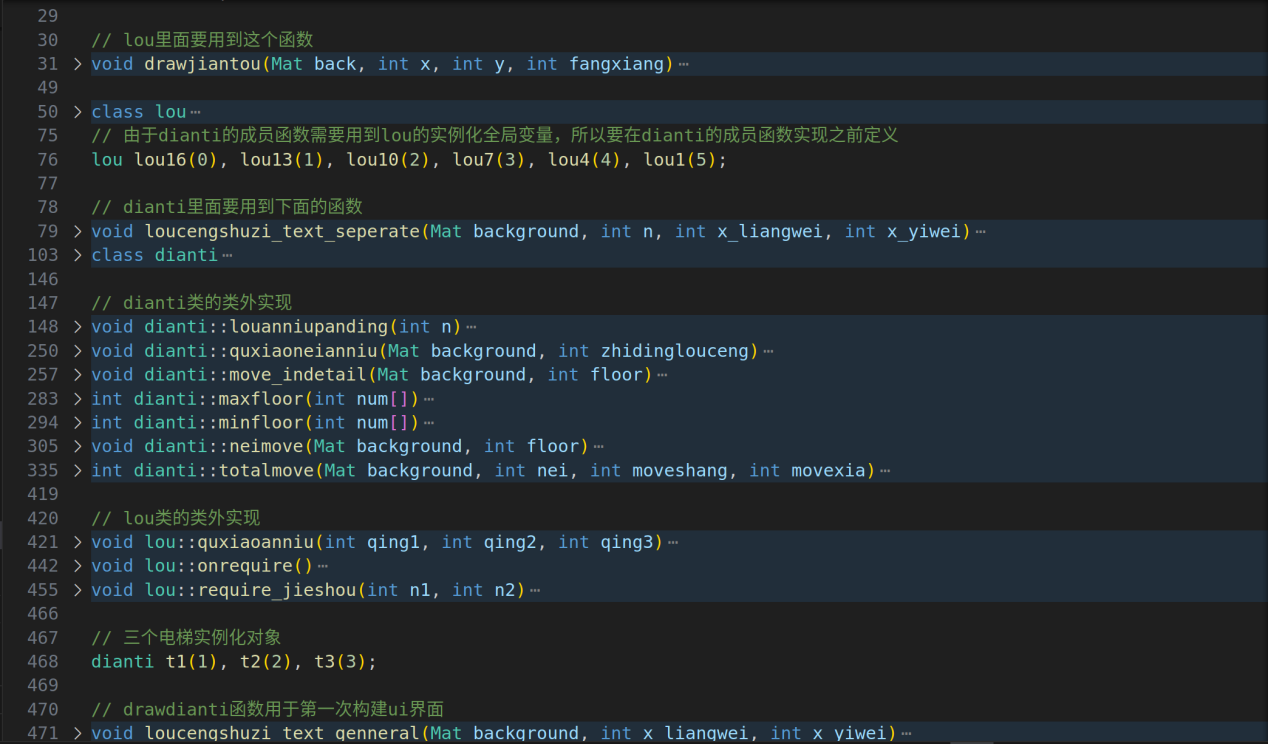
代码展示：

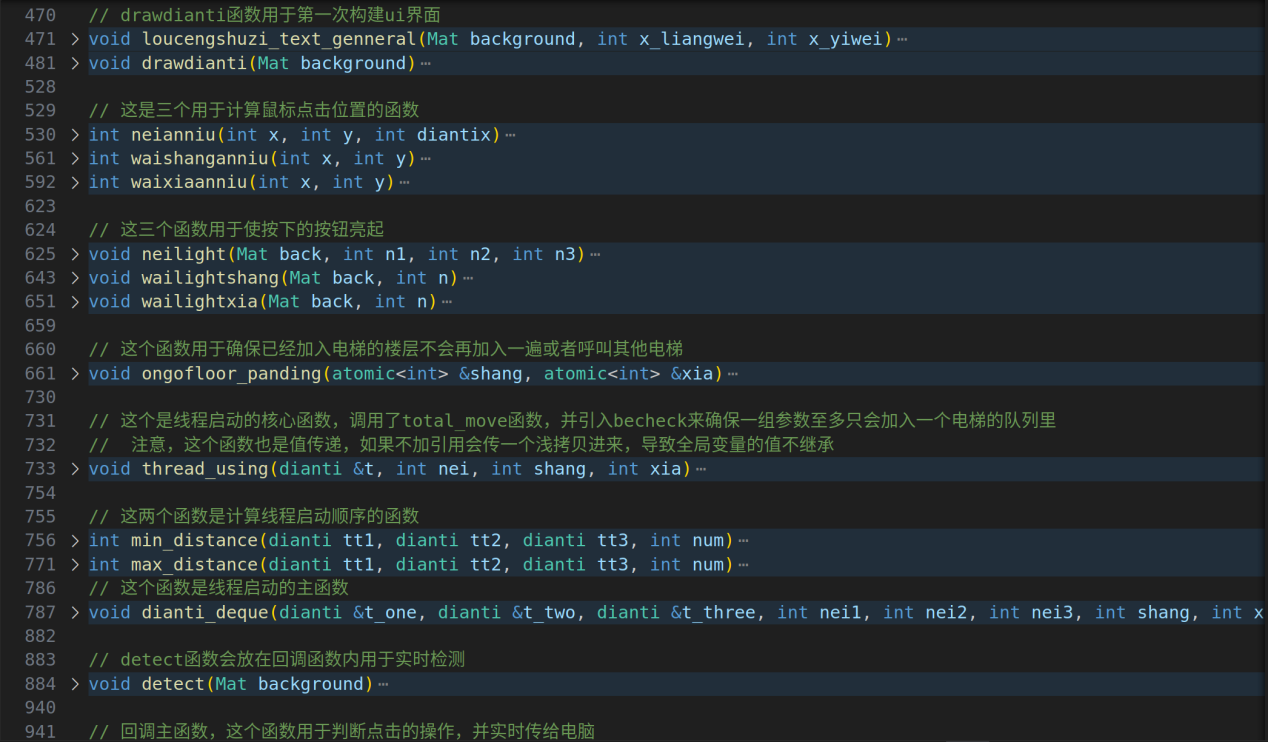
1. 函数声明——使用到的函数总览
2. 功能概况——讲解代码的运行功能
3. 代码详解——对每个功能如何实现进行具体解释

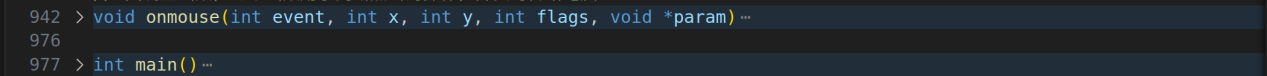
注：源代码在附件，由于可视化ui操作界面使用的是c++的opencv库，这是个动态库，所以直接生成exe会很大（因为要包含整个opencv的实现代码），所以本次作业仅提交源代码，成果演示见附件视屏

1）函数声明：



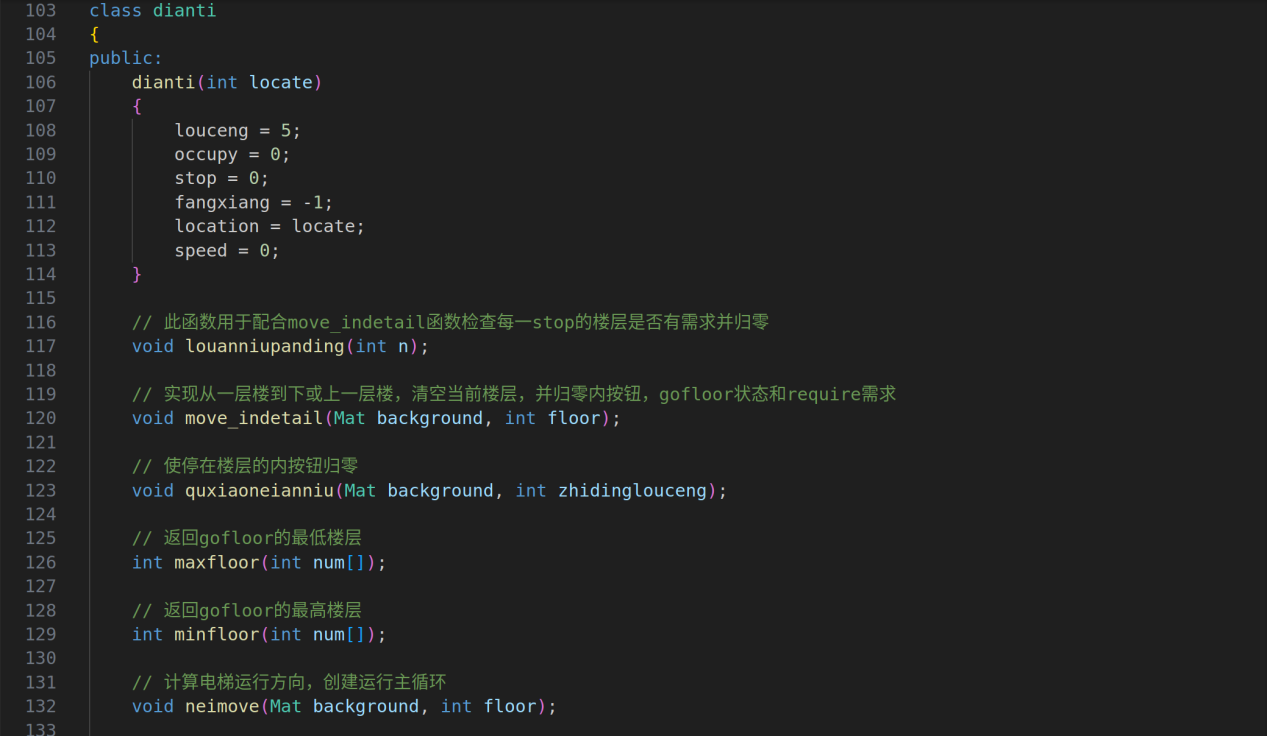


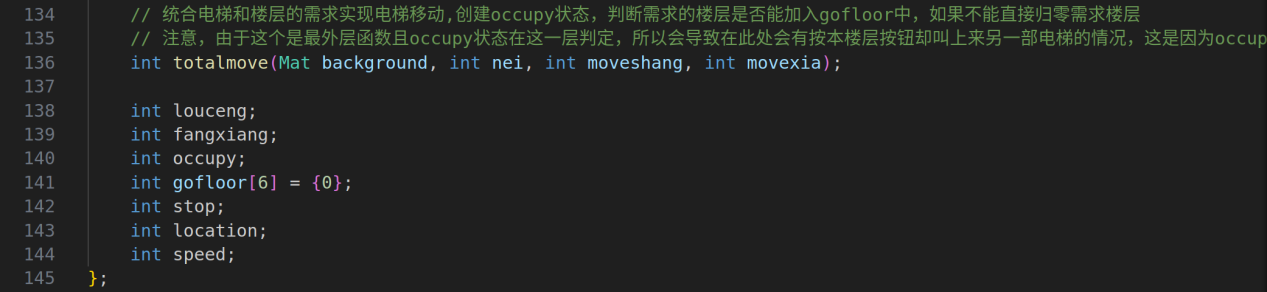


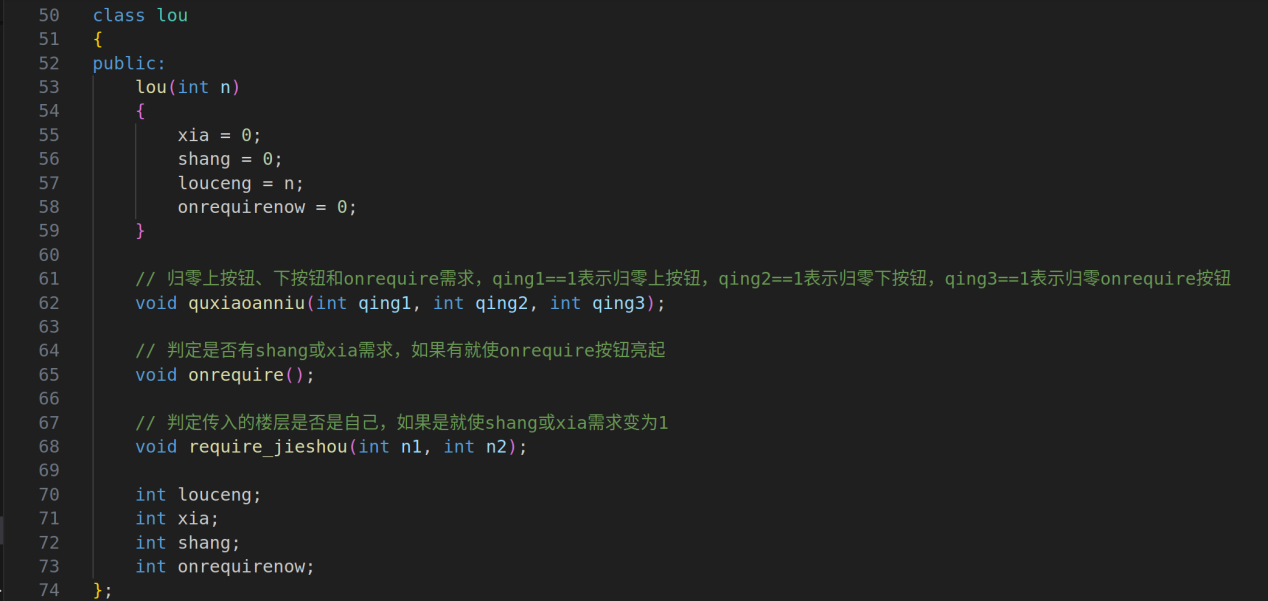


1. 功能概况

——创建两个类：dianti类和lou类，分别代表电梯和6个楼层，后续的操作将面向这几个对象进行。

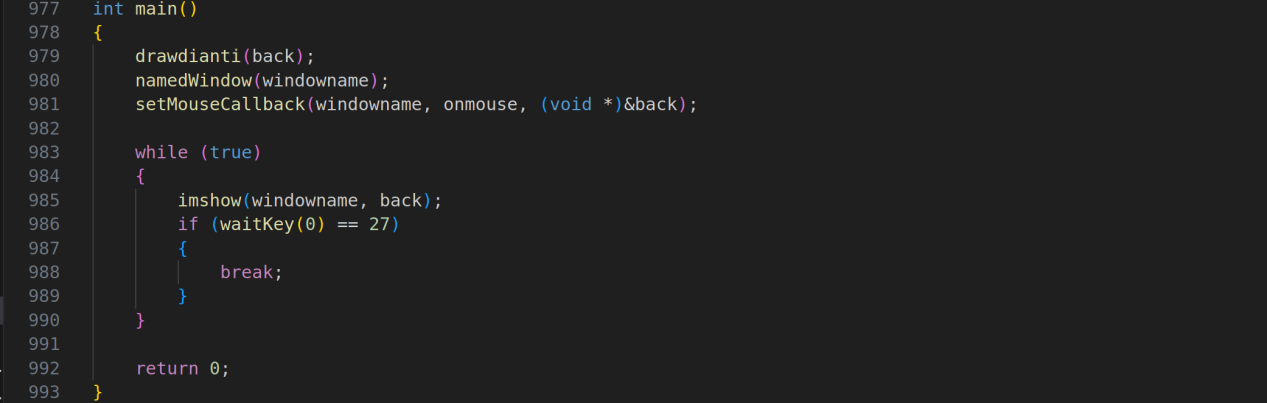






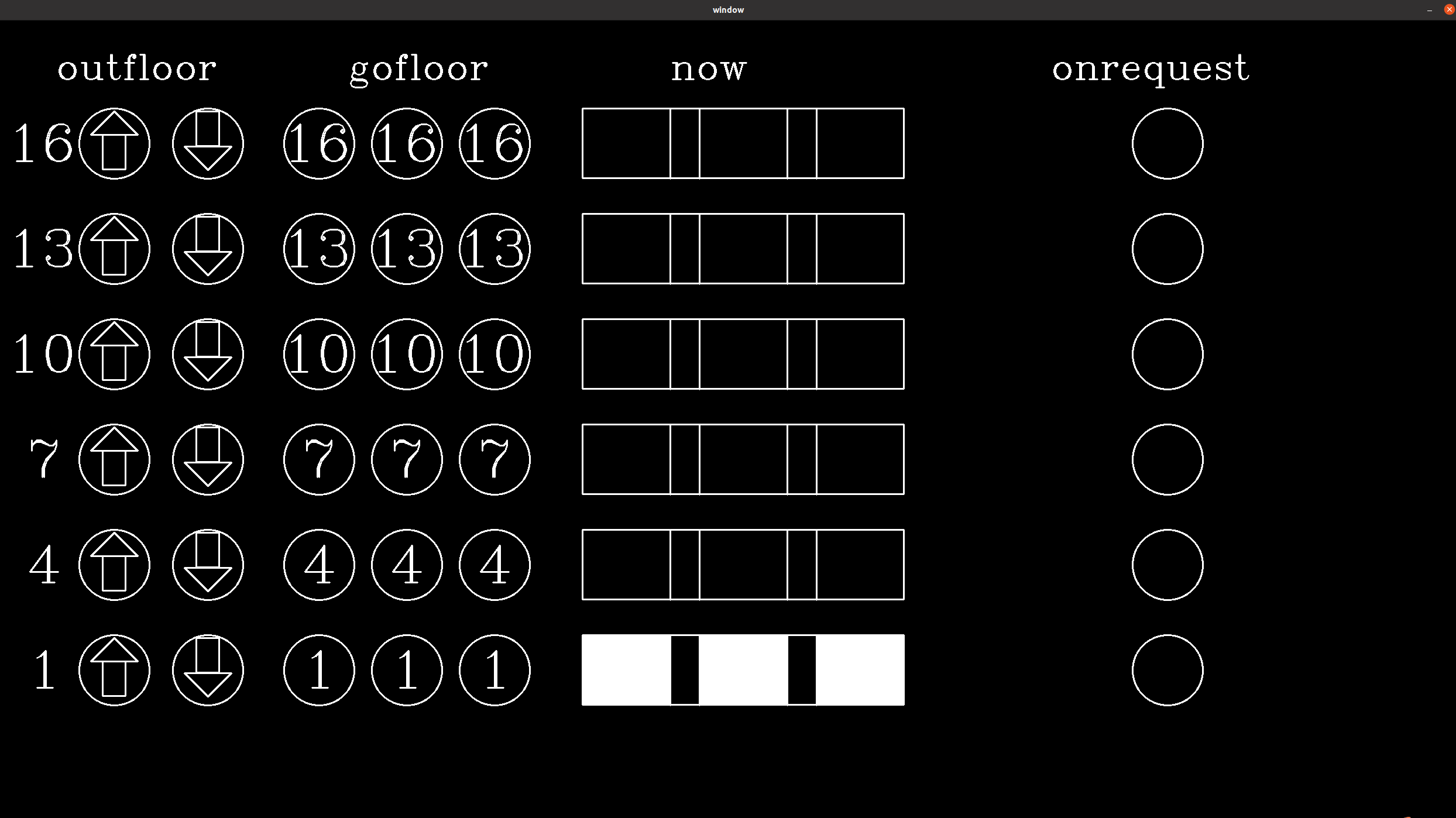
——运行过程：

1，创建电梯环境，回调函数，用于接收鼠标操作



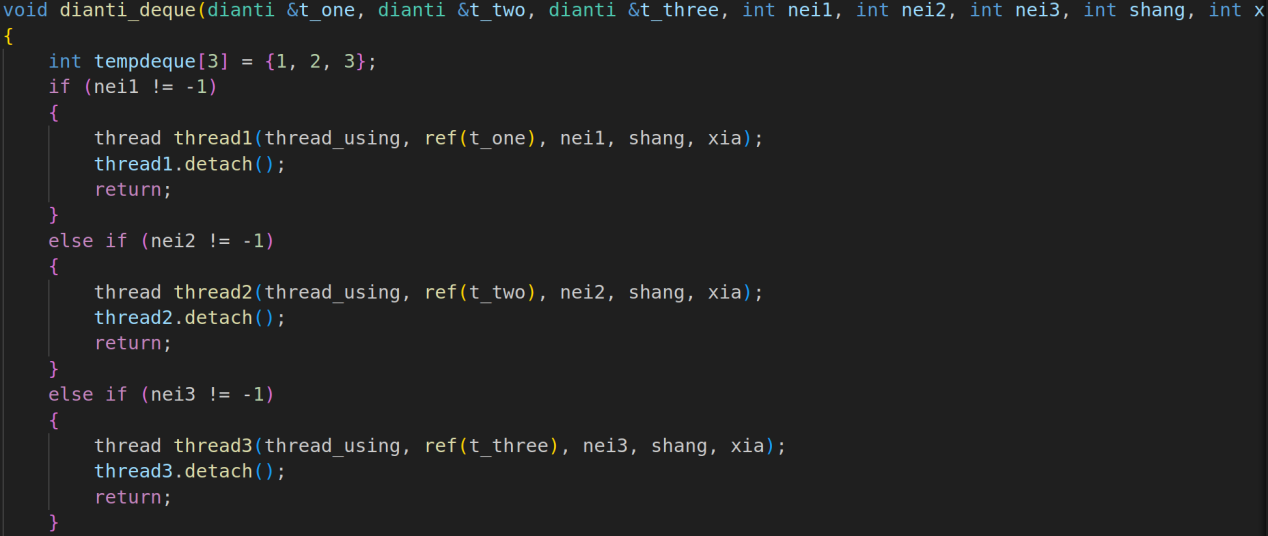
2，进入回调函数后，在鼠标点击时判断点击位置来判断是来自lou的操作还是dianti的操作

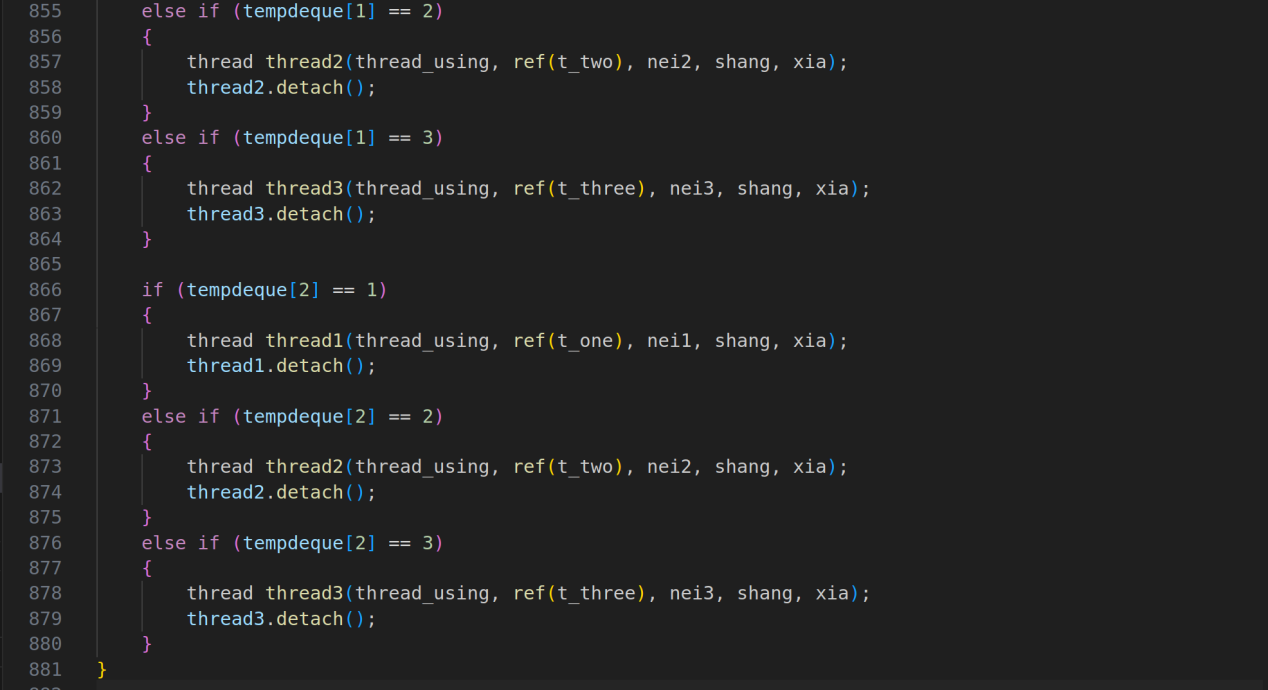
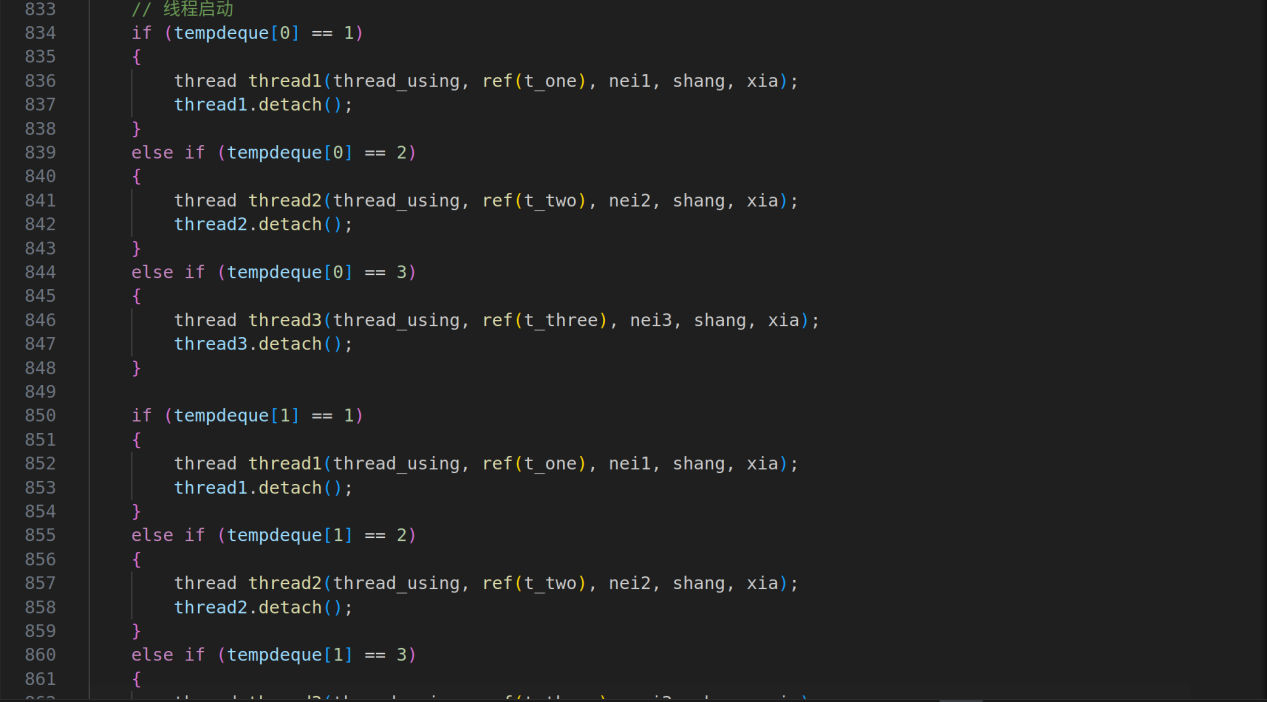
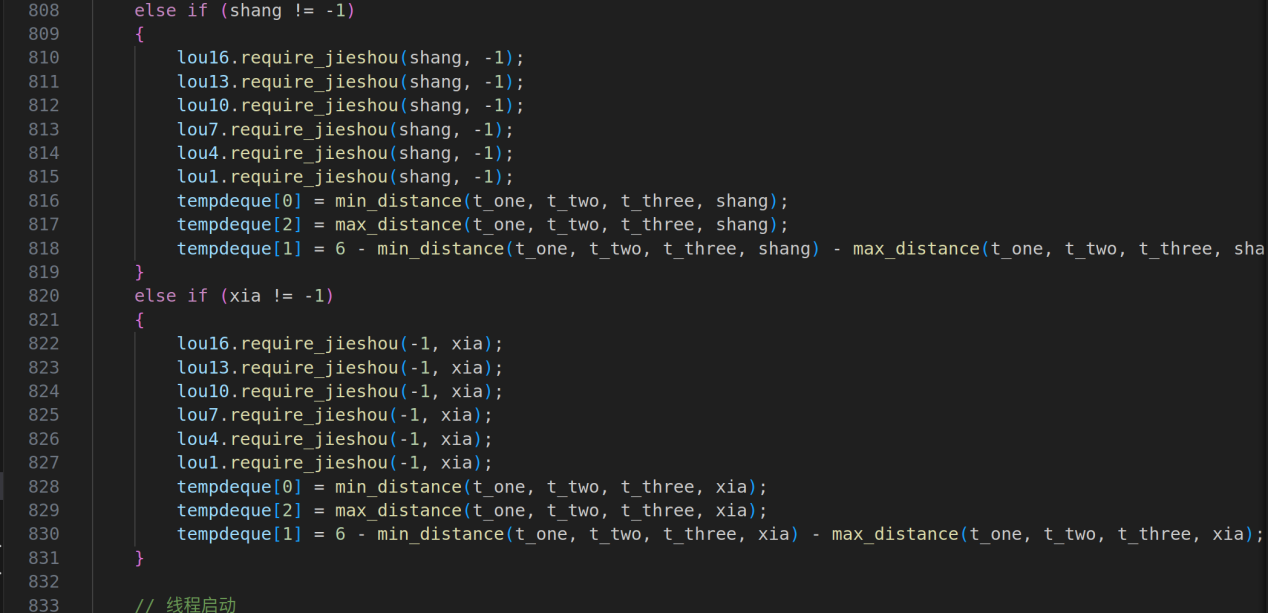
此行为模拟按按钮。（如图）outfloor两列表示每个楼的按钮，gofloor表示三台电梯内的按钮（此处为美观和lou类的整体性并未将1楼的向下按钮和16楼的向上按钮去掉），now三列表示电梯所在位置，onrequest表示某一楼有需求正在按按钮。



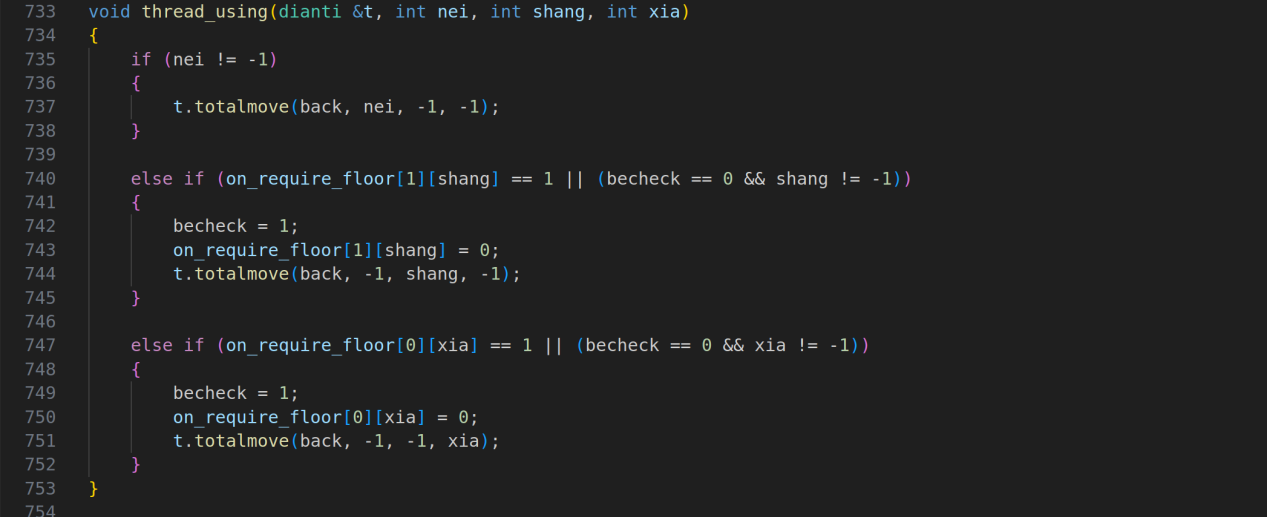
1. 当接收到鼠标操作的信号时，计算信号，然后将参数传递至dianti\_deque函数中，通过传入参数进行操作：如果是三台电梯的按钮，则只创建单个电梯的线程；如果是来自某个楼层的按钮，则按照电梯距离楼层的远近排序后分别创建三个电梯的线程。

此函数的功能在于分类传入变量进行不同情况下的线程操作，并且根据楼层情况给电梯排序。





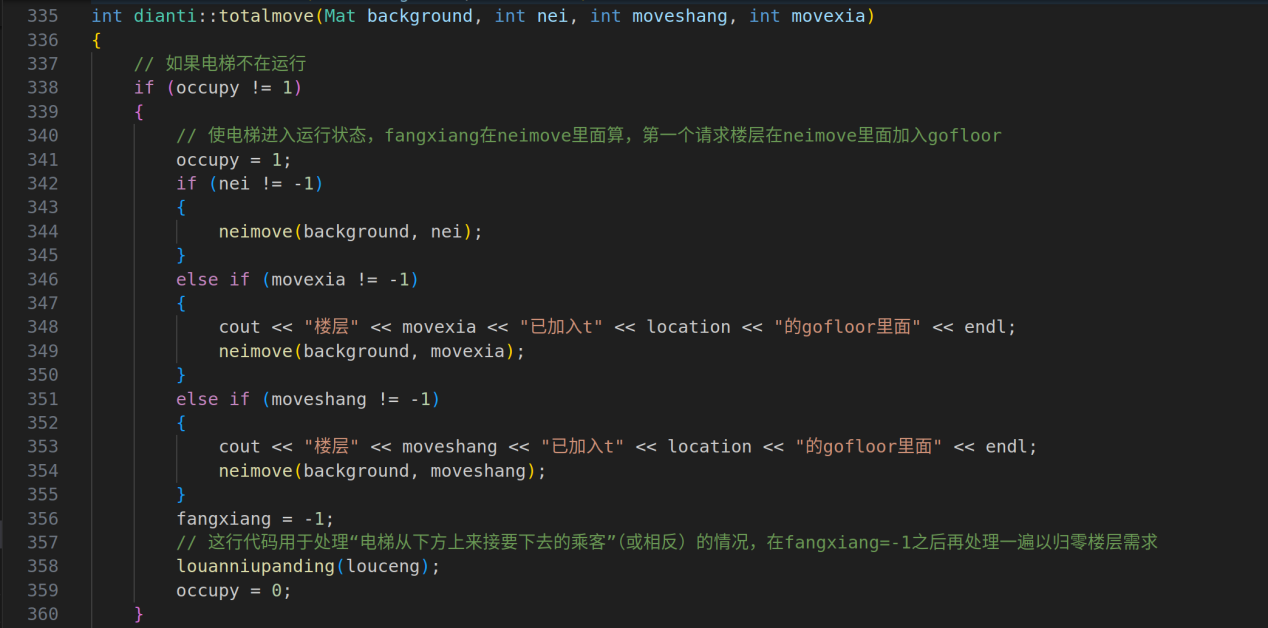
4，创建线程后，每个线程内会根据传入的参数进行操作。该函数最重要的地方在于判断这个请求是否被读取过，如果被读取，becheck变量会变成1。这个函数主要是为了避免同一个需求占用多个电梯资源。



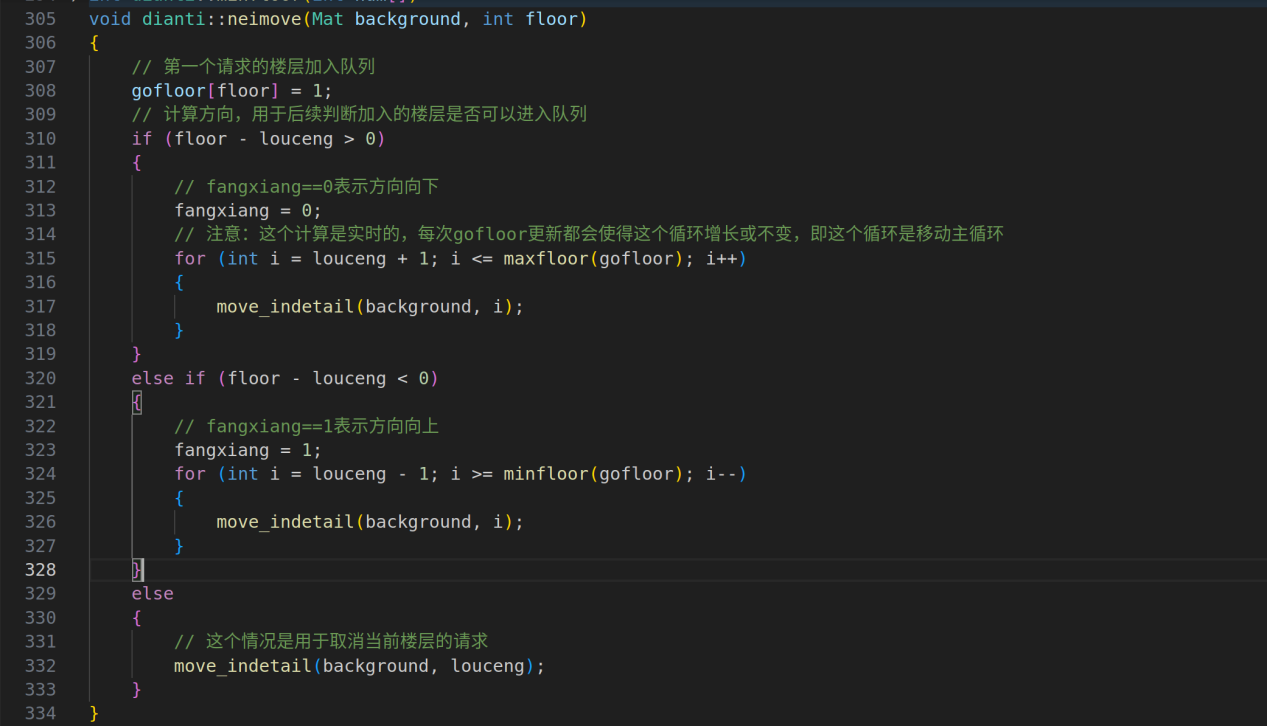
5，然后进入total\_move函数中，根据传入的参数和电梯的状态进行操作。

如果电梯不在运行状态，则创建运行状态并计算方向，开始循环

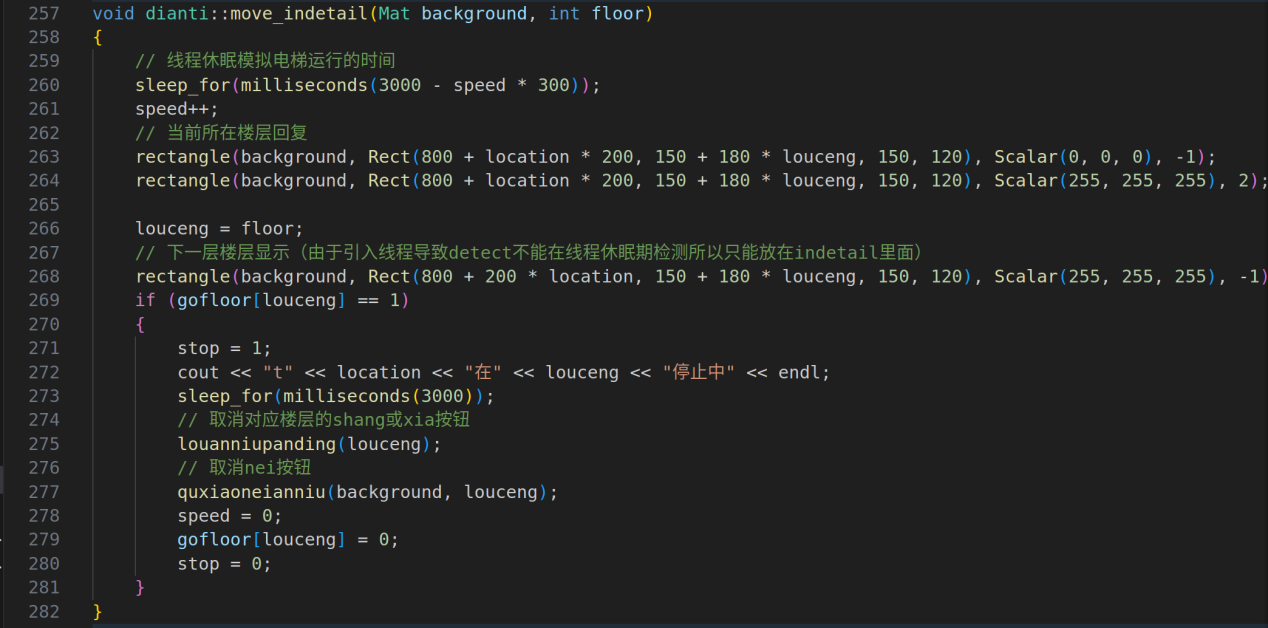
如果电梯已在运行状态，则后续的请求将不能改变电梯已有的运行方向和目标楼层列表，只能进行判断，尝试加入当前运行状态的列表中。



6，进入nei\_move函数，创建运行循环。每个运行循环会先计算运行方向，并时刻接收新指令，然后逐楼层移动，直到运行到请求队列的最高层



1. 以下是每层楼之间具体移动的代码，只是途径的楼层不会进入if循环。对于要停的目标楼层，会进入stop状态



1. 当nei\_move循环完成后，fangxiang会归-1，total\_move函数归零occupy变量，使得电梯重新变为未运行状态。

3）代码详解

在已经了解该代码的基本运行流程之后，开始讲述每个函数的作用和实现。

——dianti类：

dianti类是代表电梯对象的类，后续的操作将基于这个类的对象进行。对成员函数的详解在后续的流程讲解中具体解释。

louceng——表示当前dianti对象所处的位置，0代表16楼，1代表13楼，依次类推，5代表1楼。

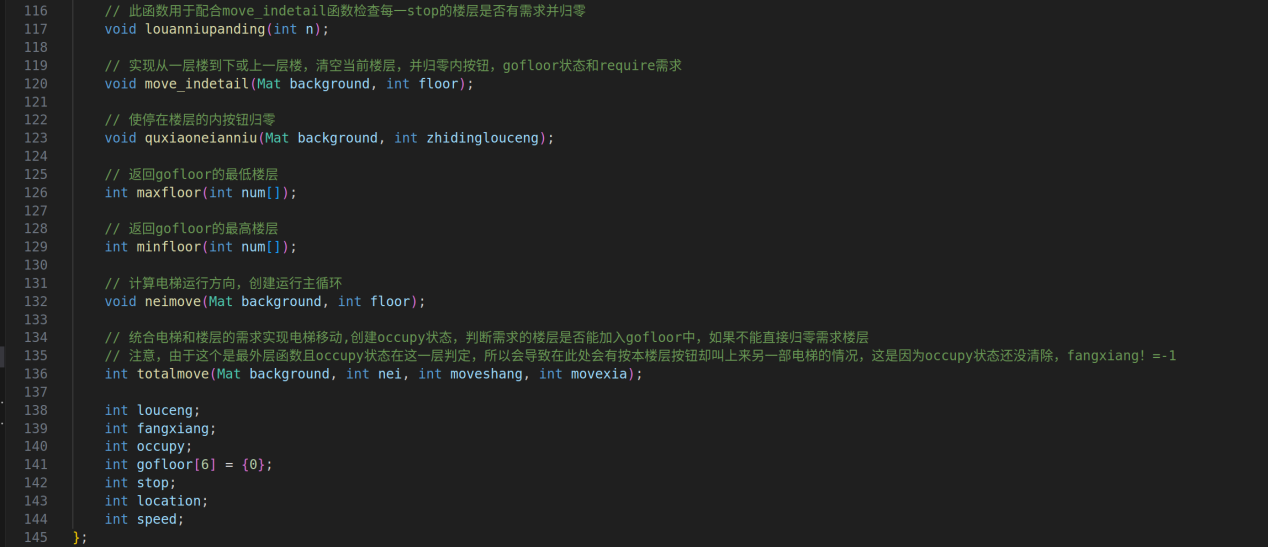
fangxiang——电梯运行过程中的方向，决定dianti最终可以到1楼还是16楼，fangxiang==1代表向上，fangxiang==0代表向下，fangxiang==-1代表静止。

occupy和stop变量，==1时分别代表正在运行和在某层楼停止，主要是便于后续lou对象的部分函数实现相应功能。

gofloor数组——代表运行过程中停的楼层，在点击输入数据并成功加入队列时gofloor的对应下标的数据会变成1，stop在某一楼层后变回0。

location——代表电梯的编号，最左边的电梯是1号，最右边的电梯是3号。这个编号主要是为了方便画图和调试时区分哪个电梯做了什么操作。

speed——代表电梯的速度，这个变量在运行过程中模拟电梯逐渐加速的过程，原本是要作为算法的一部分加入计算考虑，但是经过实测发现电梯停止导致的加速过程仅使电梯每运行3层减慢2秒左右，远远不及其他因素的影响大，所以仅作为对现实的模拟，在本次算法中不考虑。



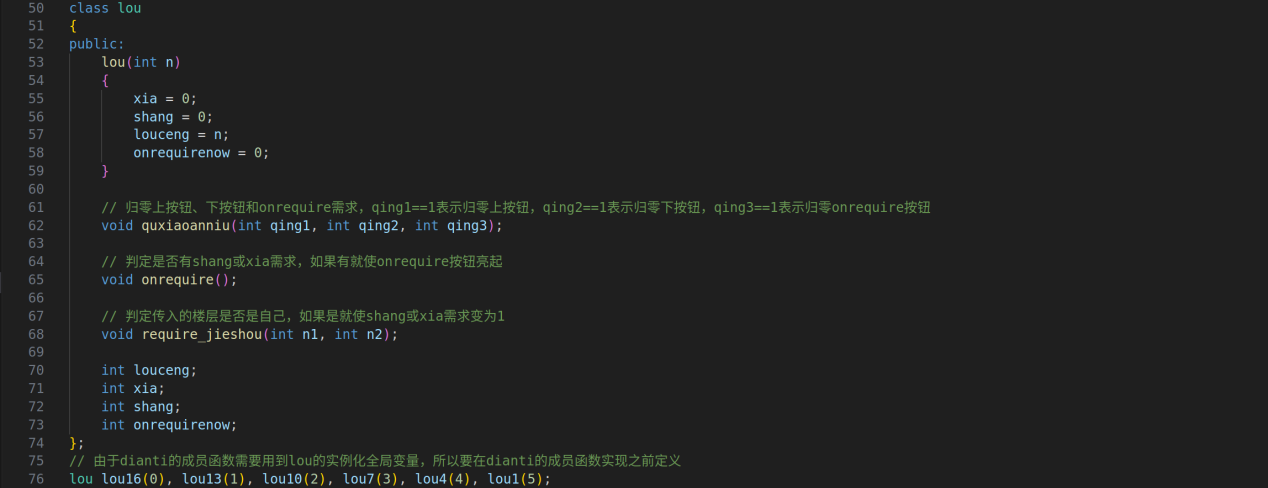
——lou类：

lou类是代表每一个楼层的类，在构造函数中传入的编号代表对应的楼。同上，0代表16楼，5代表1楼。由于dianti这个类里面有个函数，要用到实例化的lou对象，但是lou里面又有函数要用到dianti类的参数，所以做了dianti的前向声明然后把lou的定义放在了电梯的前面。

louceng——代表楼层，0代表16楼，5代表1楼。

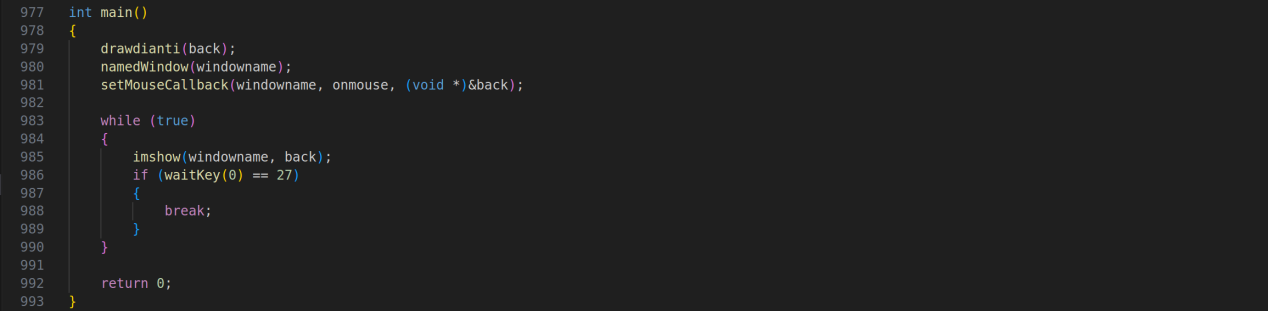
xia和shang——代表请求状态，==1代表有乘客发出要向下或向上的请求。此处为了类的整体性和ui的美观性没有对16楼和1楼的按钮和状态做特殊区分。

onrequirenow——代表该楼处于请求状态，在detect函数中用于判断onrequest提示灯是否亮起。xia或shang任意一个变量==1时onrequirenow赋值为1.



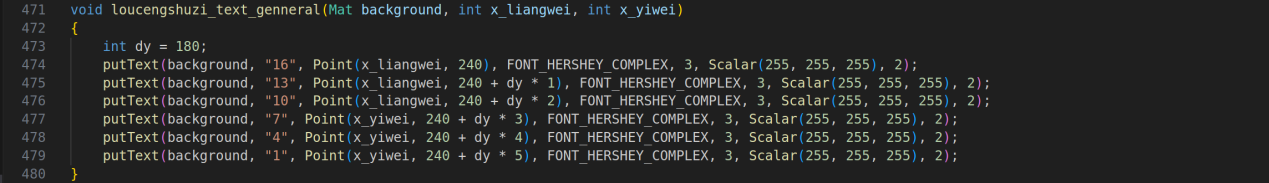
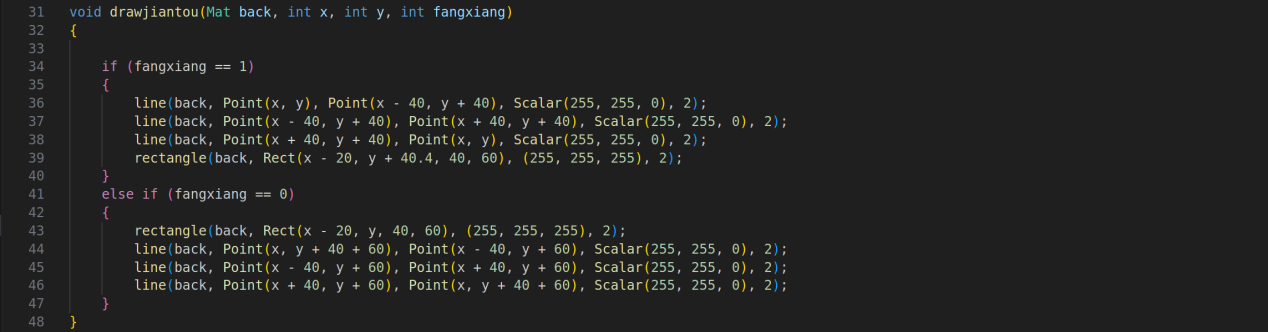
——main函数：

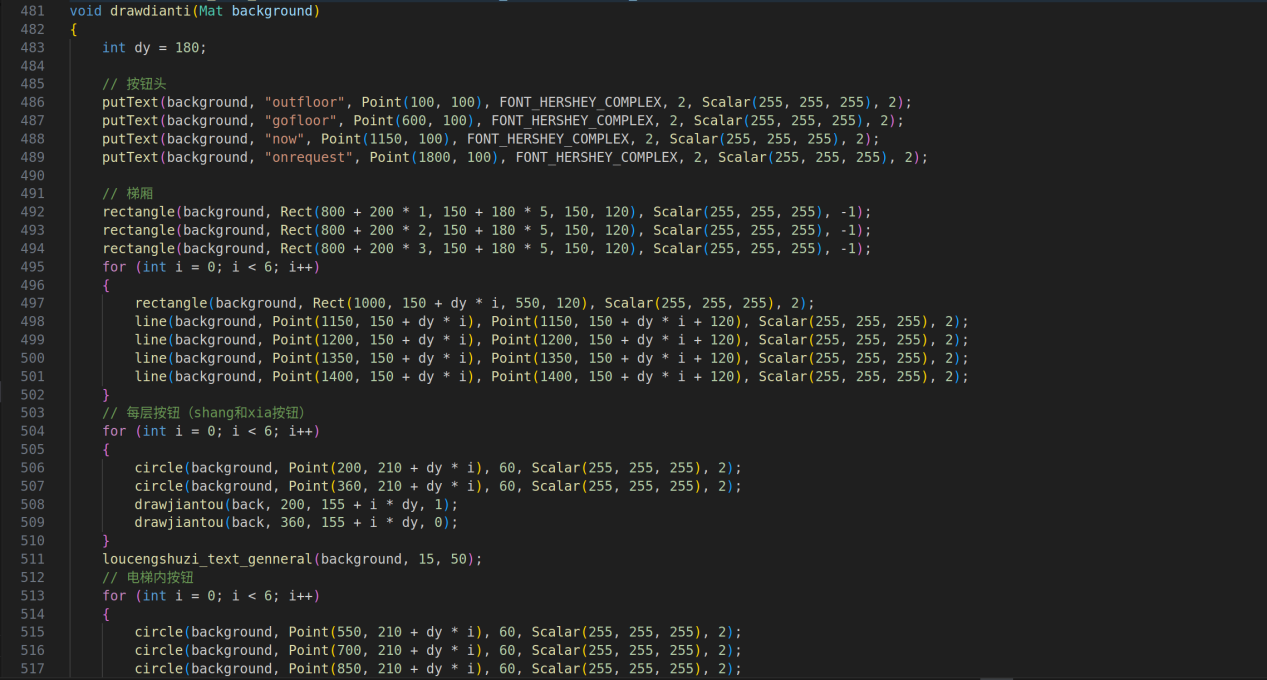
drawdinati函数用于绘制ui界面，绘制后进入setmousecallback函数，并不断显示修改后的图片，waitkey（0）==27表示按Esc退出循环。

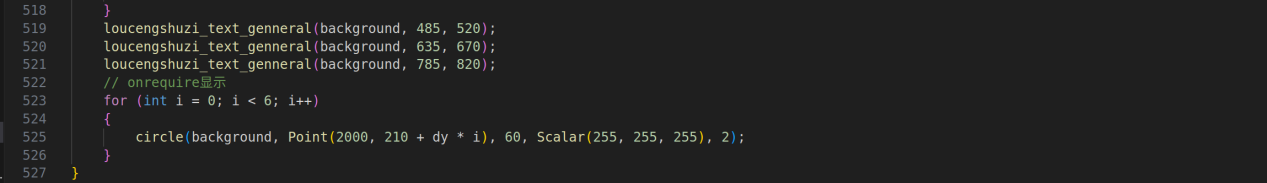


——drawdianti函数及其相关函数：

drawjiantou函数用于绘制箭头形如 和 。loucengshuzi\_text\_genneral函数用于在指定位置写16、13、10等表示楼层的数字。然后逐步绘制标头，表示电梯的梯厢，代表每个楼的上下箭头的按钮，（包含一个圆和箭头），代表每个电梯内部的按钮（3列）并使用loucengshuzi\_text\_genneral函数标上数字，最后画出代表楼层需求的onrequest按钮。







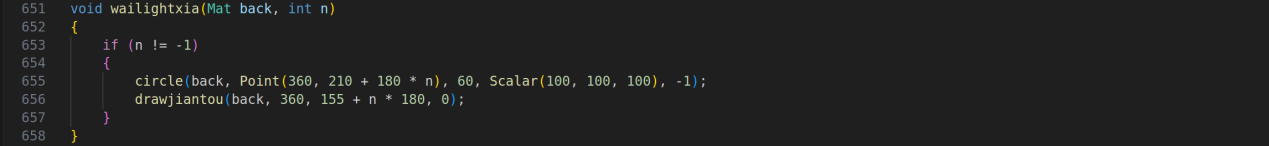
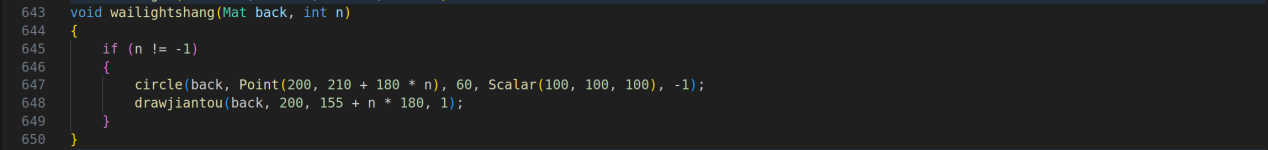
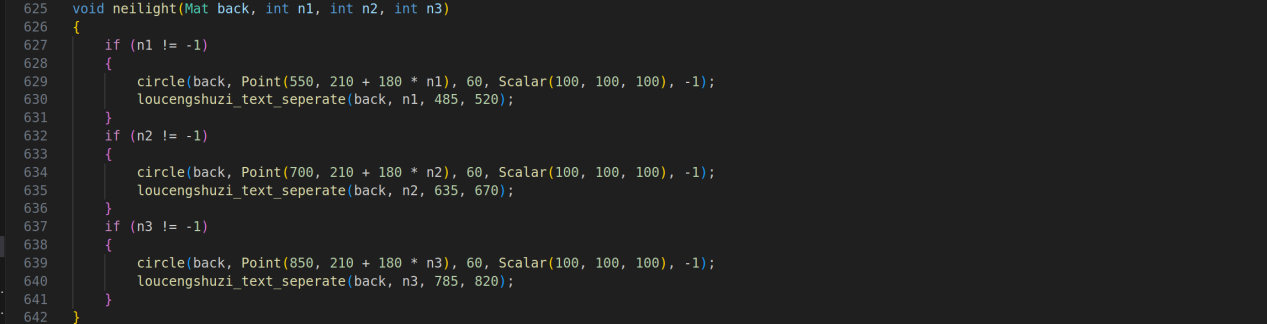
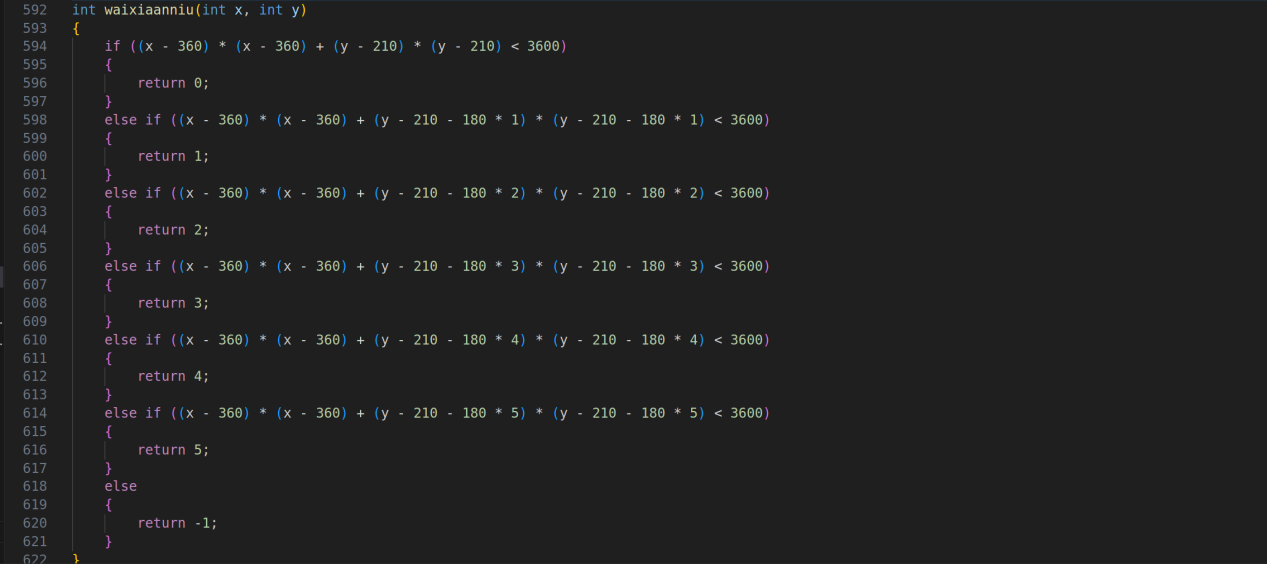
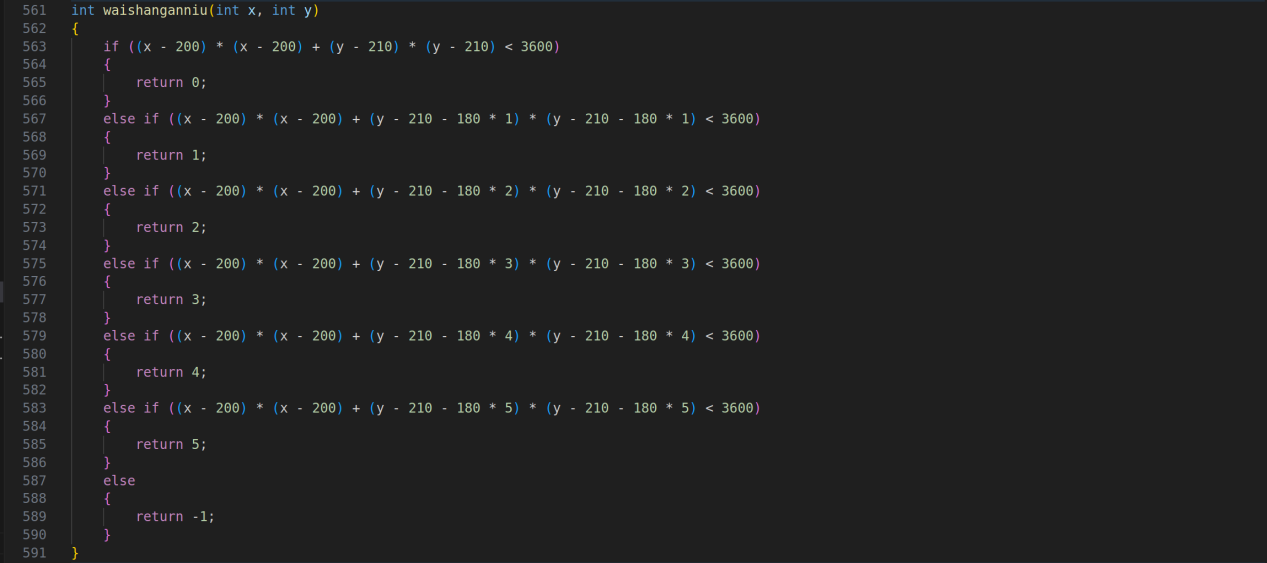
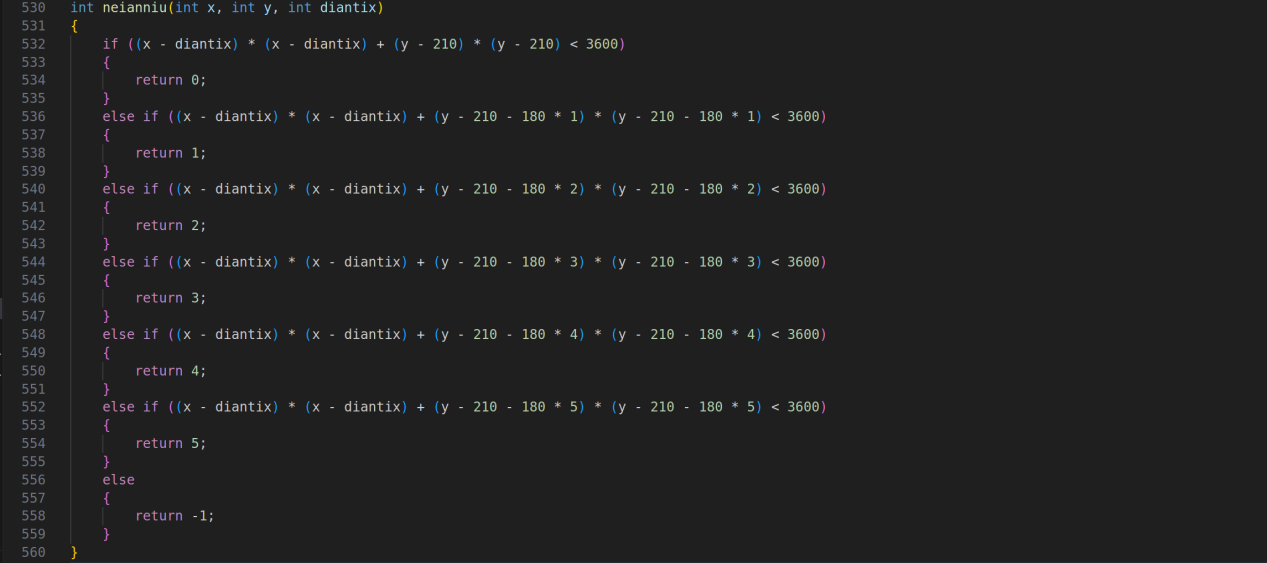
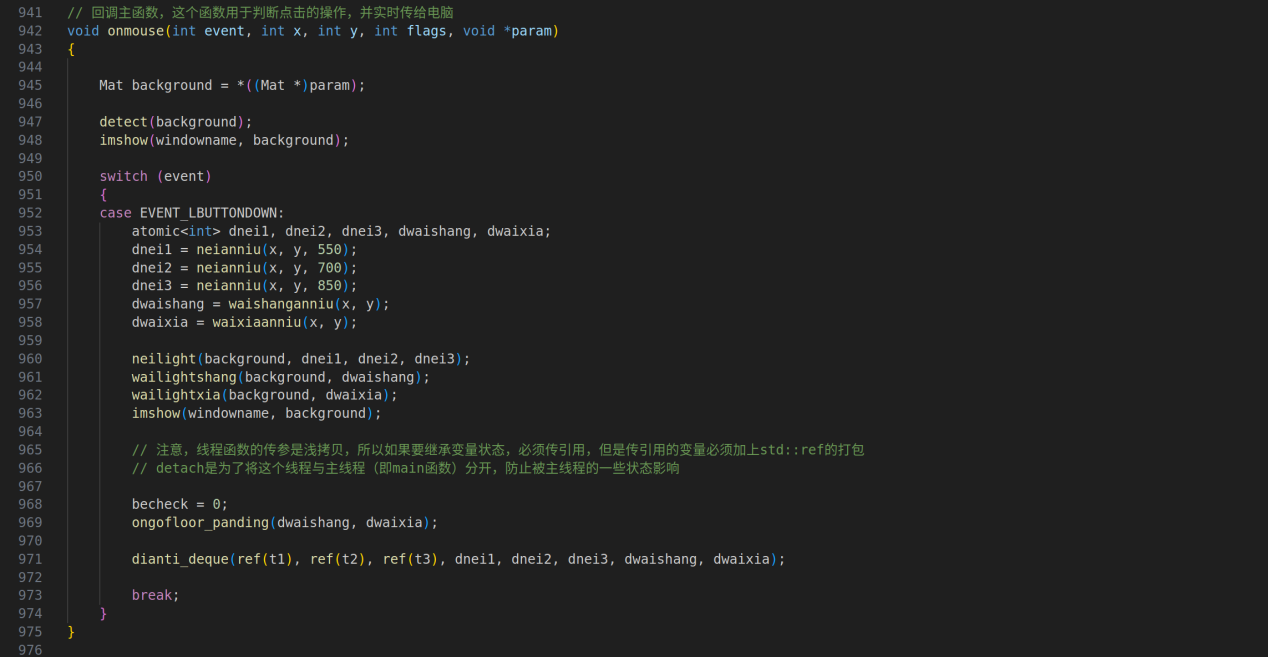
——setmousecallback函数及针对点击的相关函数：

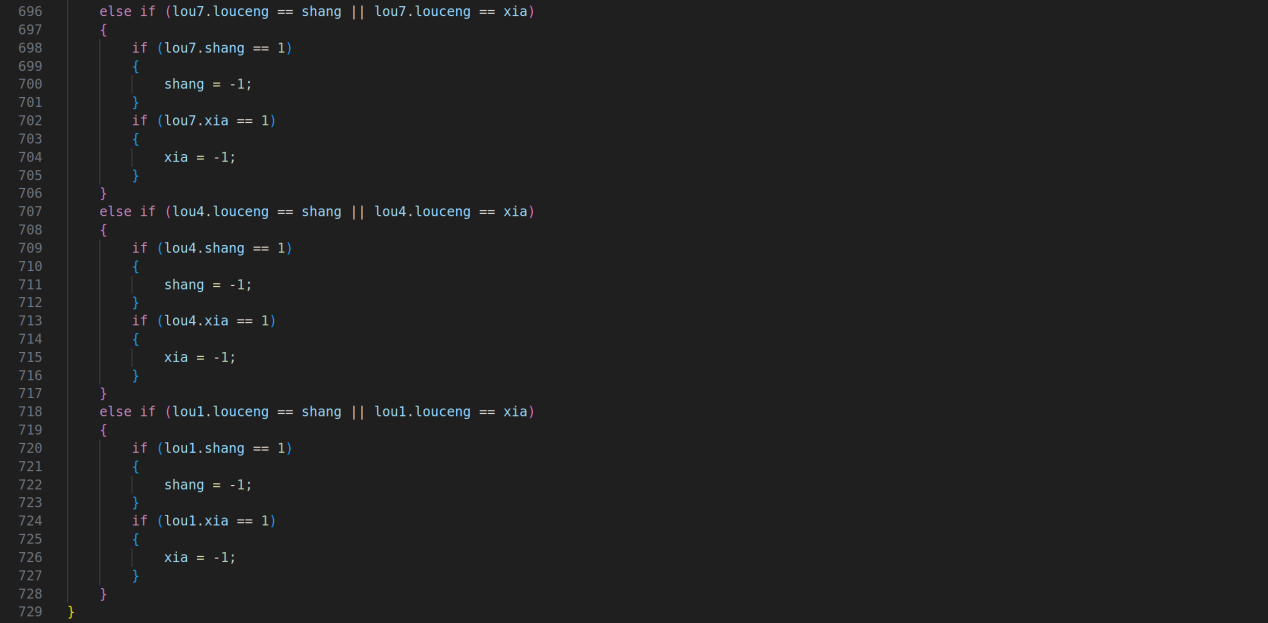
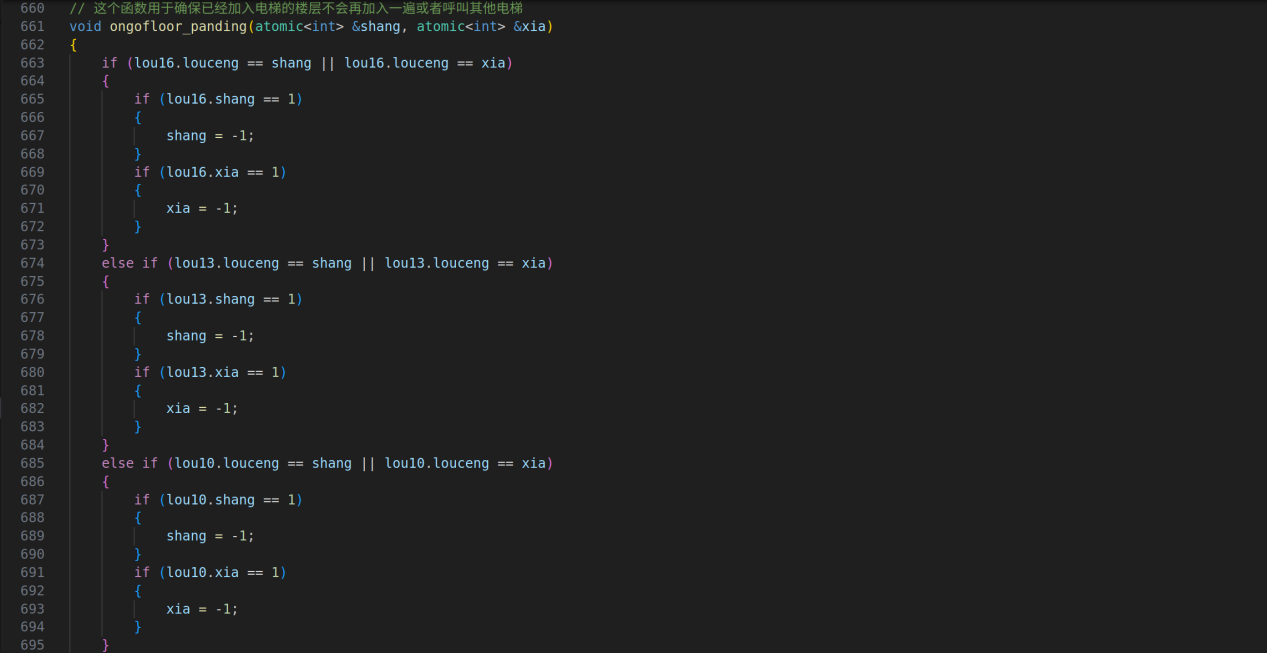
setmousecallback函数是一个回调函数（即一直运行，但也等待参数传入进行下一步操作的函数）。其中的主函数是onmouse函数，未点击时会一直执行detect函数，detect函数内是一些必要的实时检测语句。（见下文详解）

点击后，创建5个原子变量（可以自动执行mute互斥量的上锁和解锁操作，防止在多线程中发生读取错误）用于接收计算的位置信息，通过neianniu，waishanganniu和waixiaanniu来计算鼠标点击的位置。然后将计算的结果传给neilight，wailightshang和wailightxia函数，并使对应的图标变化（亮起），显示改变后的图像。在计算的过程中，如果在对应的按钮中，则返回该按钮代表的楼层值（0代表16楼，5代表1楼），否则返回-1。这可以保证每次计算得到的5个值至多只有一个是有效的，其他的值均为-1。

然后进行数据重复判定，becheck也是一个全局原子变量，定义位置如图，用于在thread\_using函数中确认这组数据是不是第一次被检测，如果不是，则不使用，改为检测是否在请求数组中。（在后续流程中详解）并通过ongofloor\_panding函数检测这个楼层是否已经请求过，如果已经请求过则直接归零数据，防止同一个楼反复申请导致占用多部电梯。

若通过ongofloor\_panding，则进入dianti\_deque函数进行对数据的电梯资源分配。



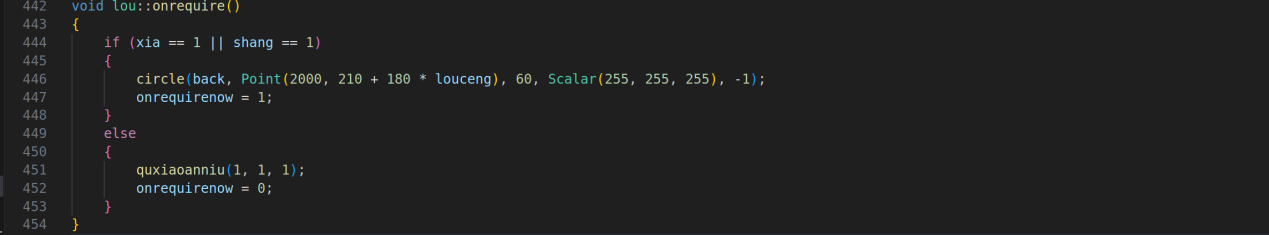
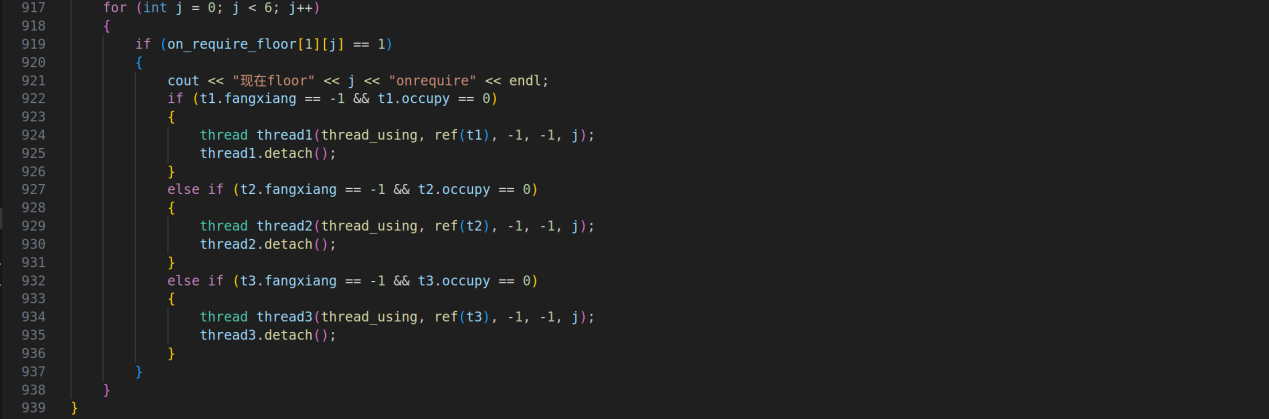
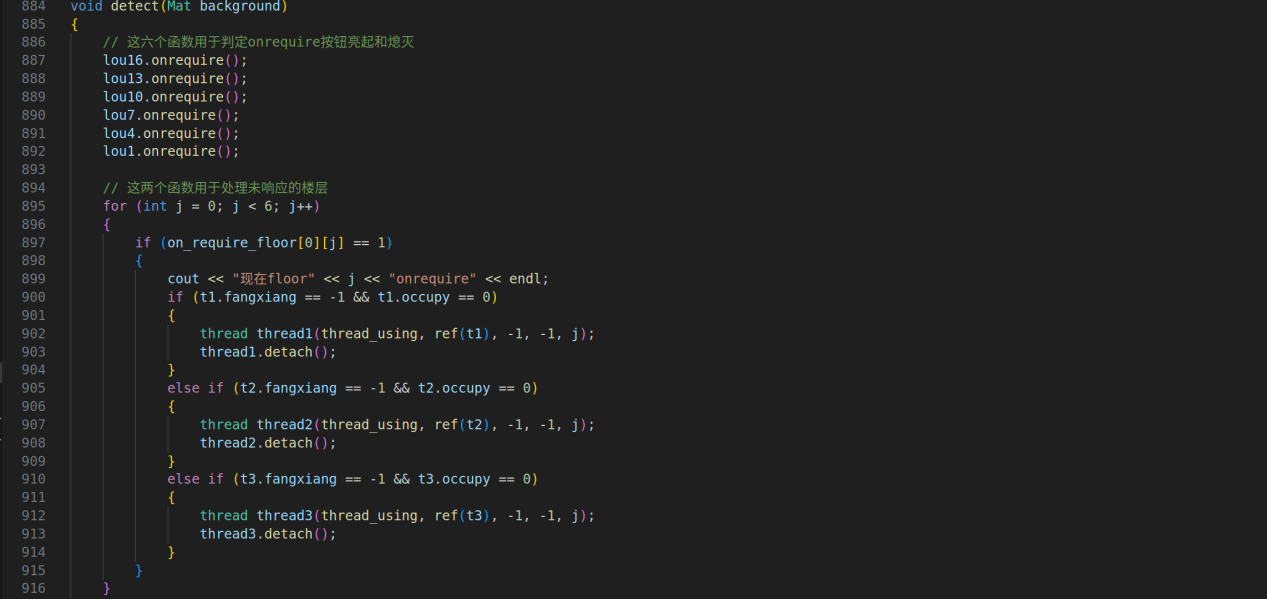


——detect函数：

第一组语句用于判断每层楼的onrequirenow的取值，shang或xia==1就赋值为1，否则赋值为0。

第二组语句用于判断当前是否有待处理的楼层，有的话进一步检测是否有电梯处于空闲状态，如果有则直接将待处理的请求交给空闲的电梯。

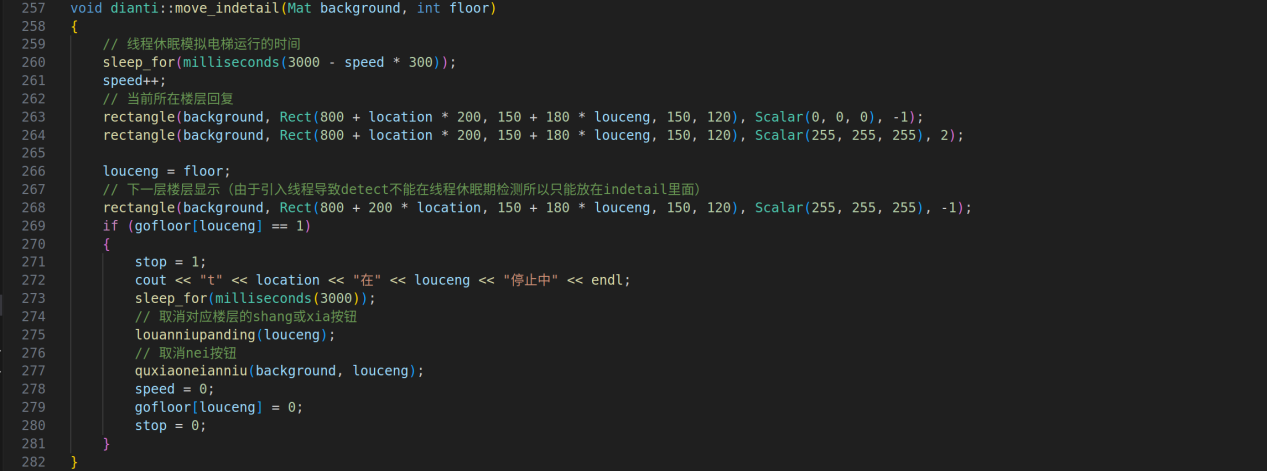
（注意：有待处理的请求是三个电梯都处于运行状态时无法处理的请求，暗示此时处于高峰期——各楼层发出向1楼的请求或1楼发出向各楼层的请求，而根据运行逻辑——优先去往正在请求的楼层中最接近终点的楼层，即向上时越靠近16楼越优先，向下时越靠近1楼越优先，会在电梯接到乘客后立刻将未处理的乘客加入队列，而这个队列一定是符合当前高峰期运行逻辑的。例：16,13,10,7楼同时发出向下请求，则7楼被闲置，当10楼的电梯接到后会立刻将7楼加入队列，并计算出fangxiang==0，与乘客去往1楼的请求是同方向的此时只要在电梯内按下1楼的按钮即可让10楼和7楼的乘客都去往1楼）。



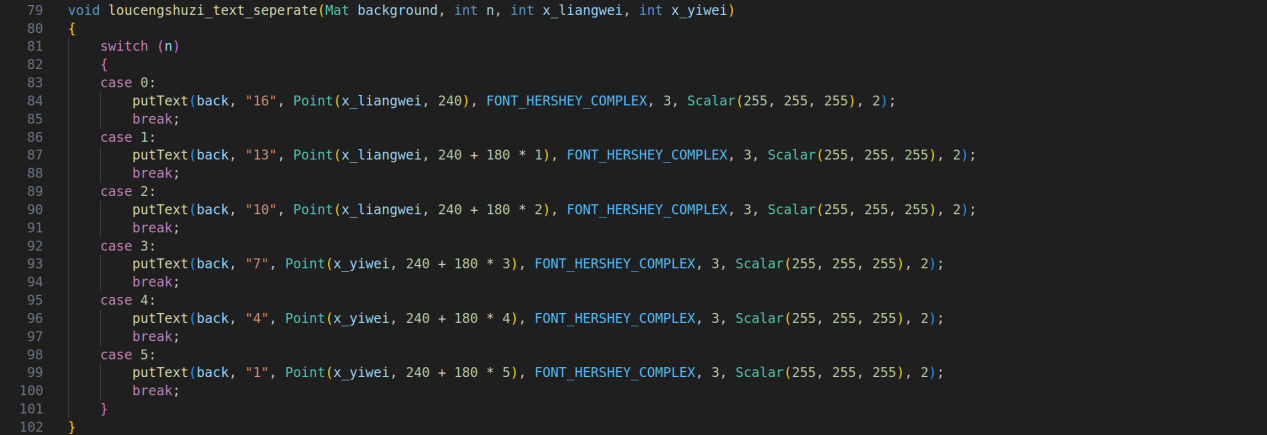
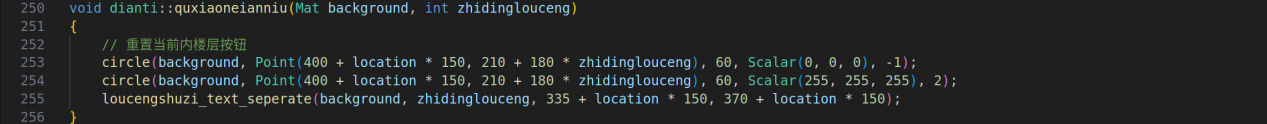
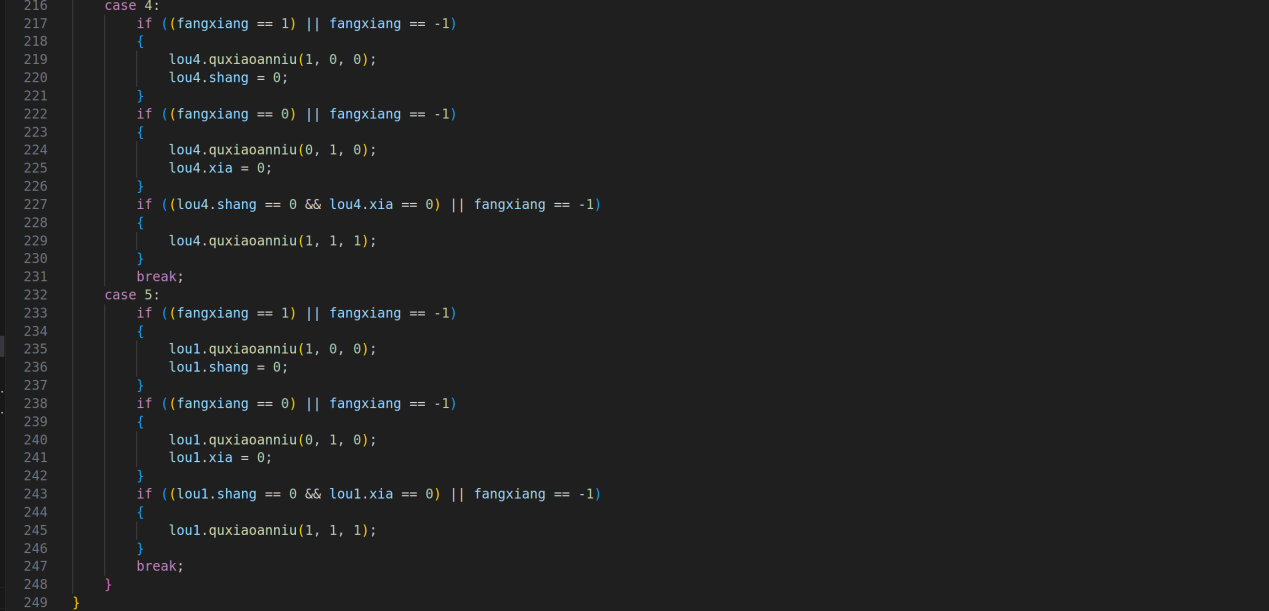
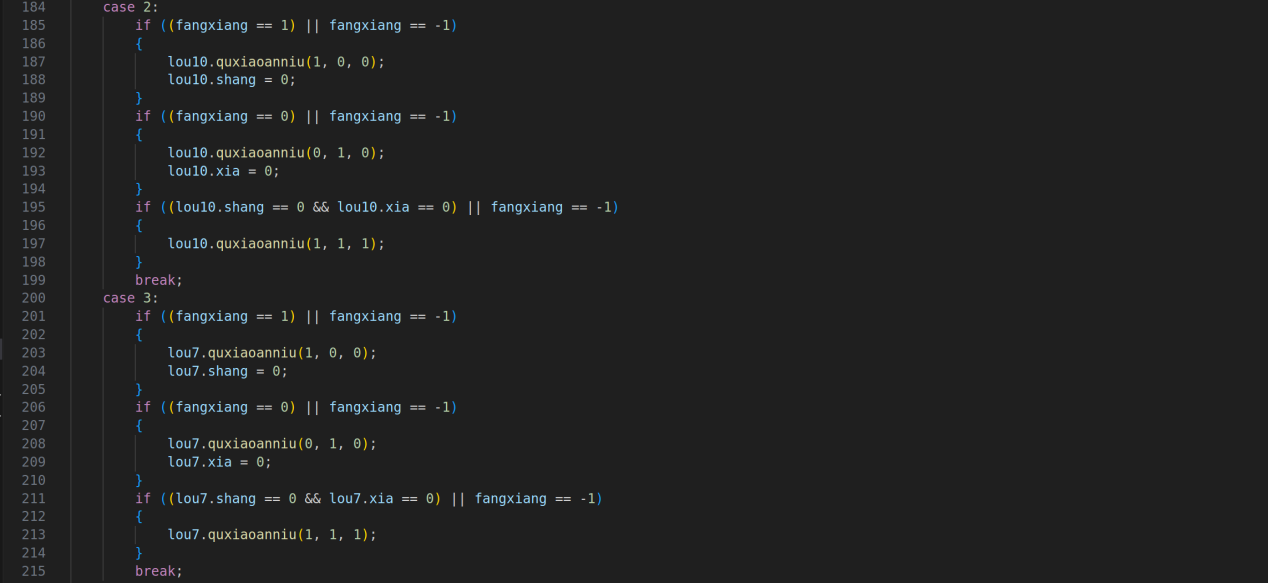
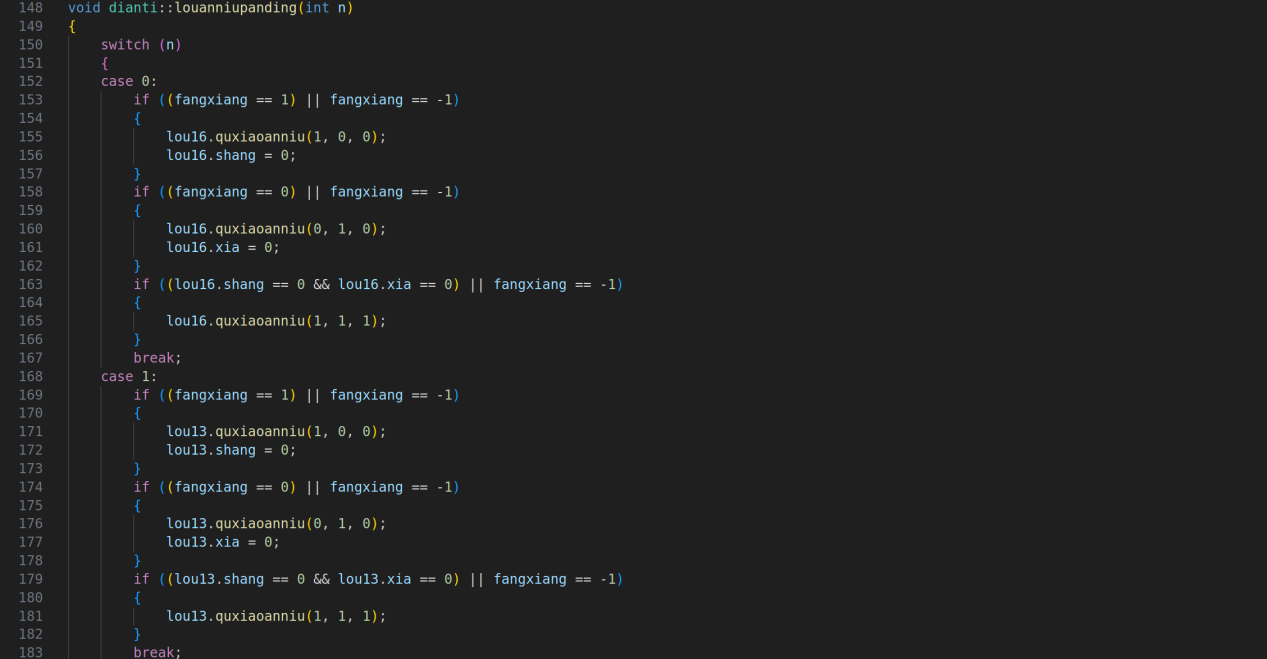
——电梯线程创建及运行相关函数

—move\_indetail函数：

先使线程休眠3s左右，然后加速，将当前所在楼层的图标还原。然后使自身的louceng变为neimove计算出的下一层或上一层，将这一层的图标变为白色，判断这一层是不是要停的楼层，如果是就额外再休眠3s左右，发出提示信息，通过quxiaoneianniu函数取消该楼层的电梯按钮，通过louanniupanding函数取消对应的lou对象的上按钮或下按钮，将速度归零，使当前的队列该楼层数据归零。

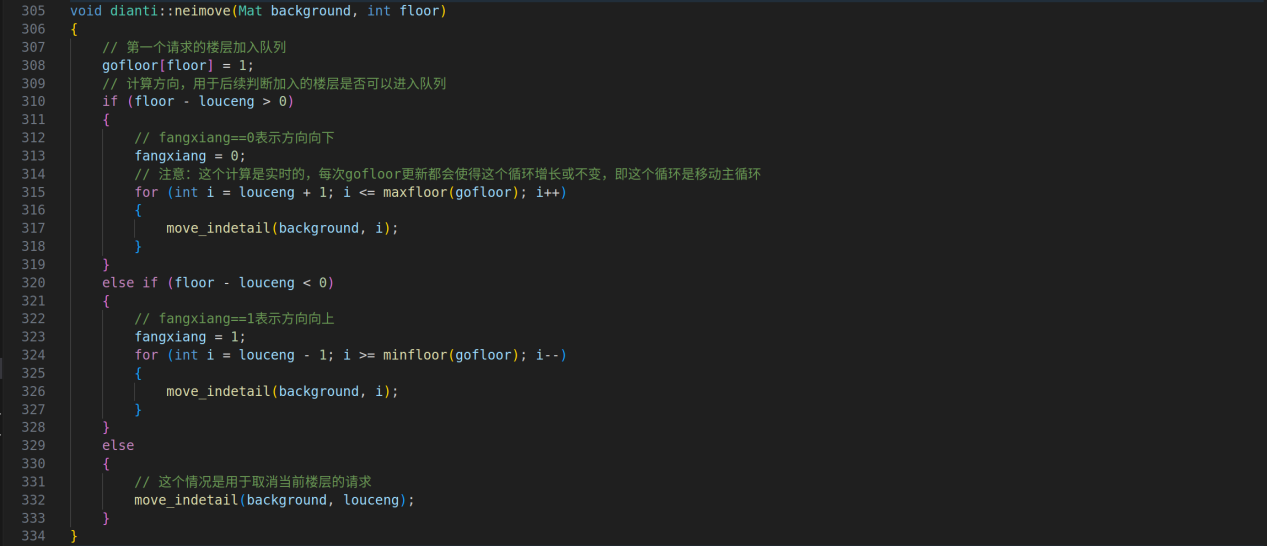


（注意：只能清零当前方向的按钮，如向上的电梯不能清除向下需求的按钮）



—neimove函数

这是创建循环的函数，并且只在电梯未运行时调用。计算total\_move传入的楼层与自己当前的楼层的差，计算出方向，并根据方向决定是向下移动还是向上移动。如果正好等于当前楼层，也必须进行一次操作，否则这一层的按钮不会消。创建循环后，会逐层途径经过的楼层，由move\_indetail来判断是否在这一层停止。



—total\_move函数

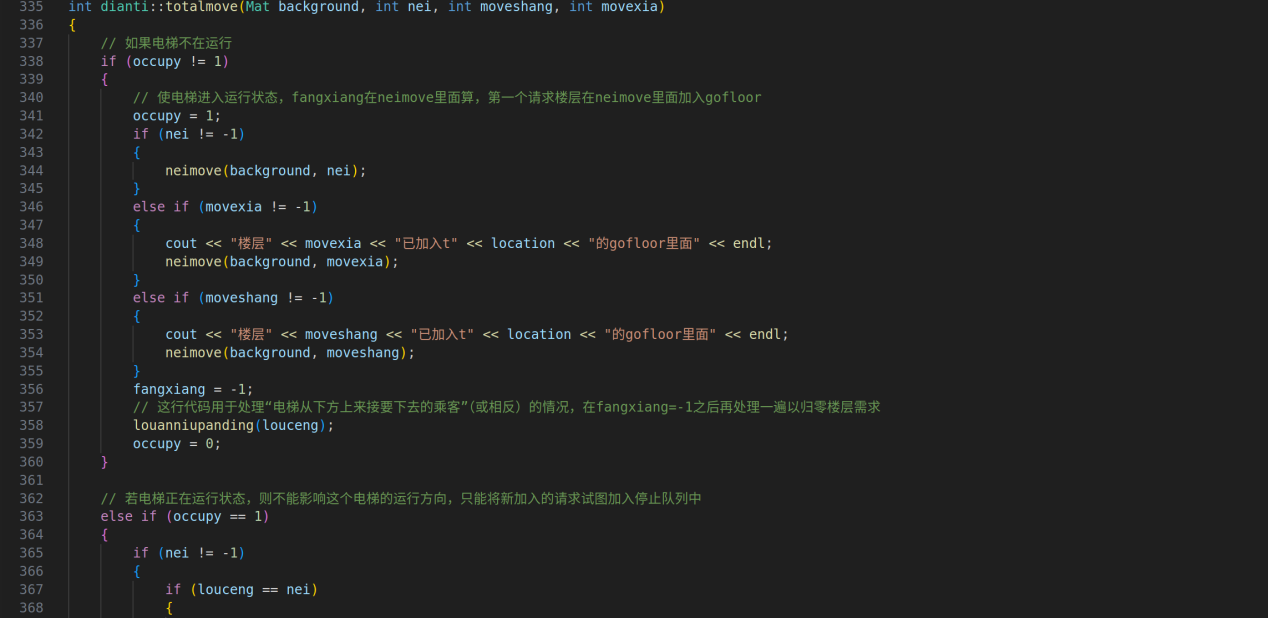
这是统合数据并根据传入数据和电梯状态判断是否可以让请求加入队列的函数。

如果电梯不处于运行状态，则将不等于-1的数传给nei\_move创建循环，并使occupy=1，代表电梯正在运行。

如果电梯正处于运行状态，则将请求的数据与当前电梯的方向和位置对比，如果是来自电梯内按钮的请求，并且在未到达的楼层中，则可以加入gofloor数组，否则提示“不可去往”并通过quxiaoneianniu函数立刻清除请求。（如向下的电梯正处于16楼，有一个前往4楼的请求，则可以前往。）

同理处理来自其他楼层的向上或向下的请求。（如向下的电梯正处于16楼，有一个来自4楼的向下请求，则可以接上这个客人，但来自4楼的向上请求则不可以）如果有不能处理的请求，则将这个请求纳入on\_require\_floor全局数组当中，通过thread\_deque或者detect交由其他电梯运行。

（注：因为这是在一个线程内，上一个线程的函数会持续运行，所以新线程可以提前于旧线程运行，所以将新数据加入队列后，旧线程的循环函数仍然可以得到新的gofloor数组状态。）

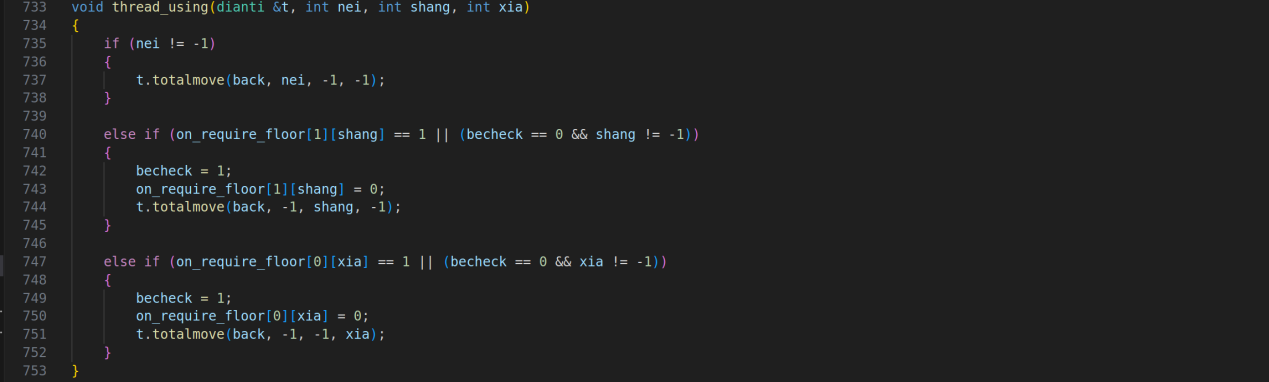


—thread\_using函数

这个是线程启动后运行的函数，首先判断需求是不是内按钮需求，这个判断无需经过becheck检查，因为thread\_deque函数中内按钮需求不会传到其他电梯的线程中。

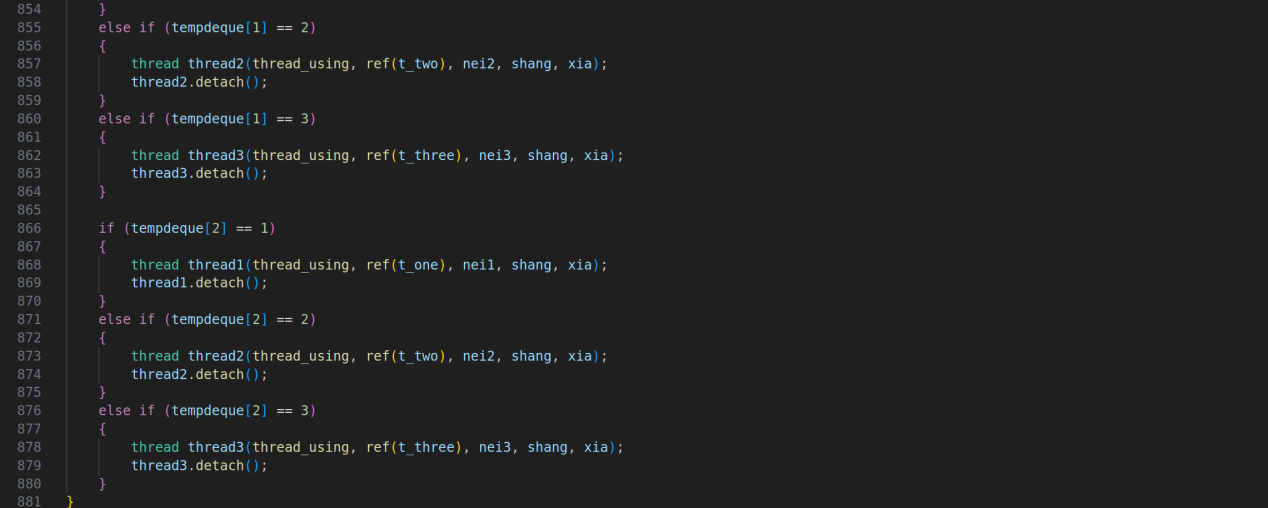
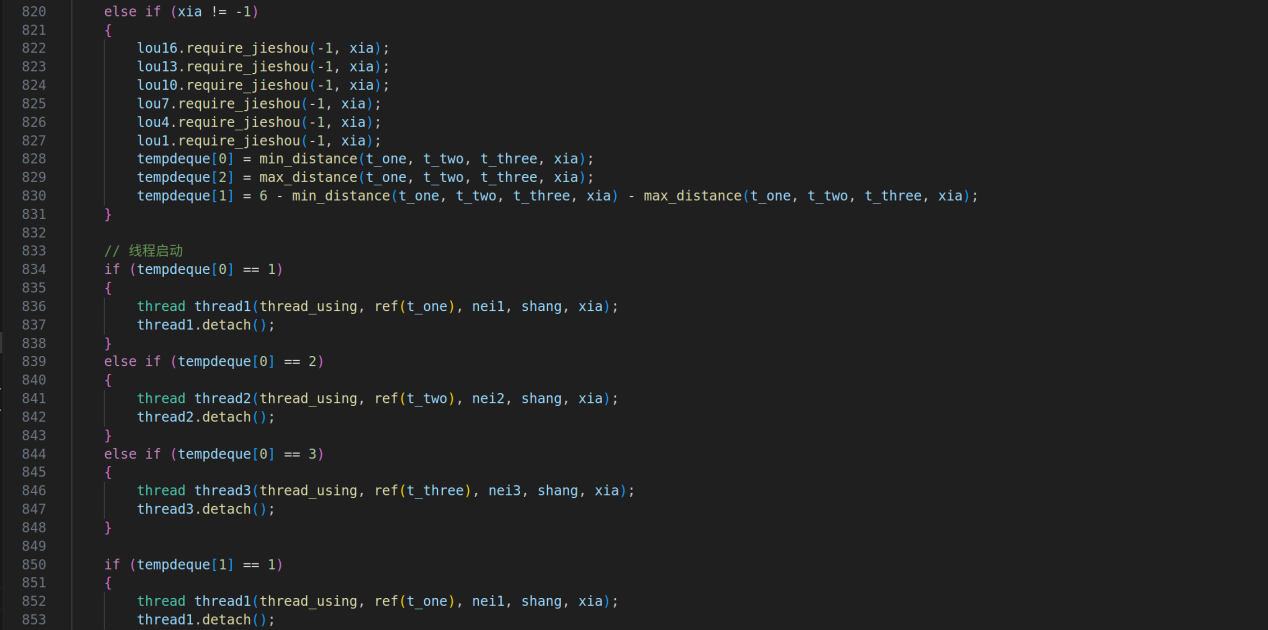
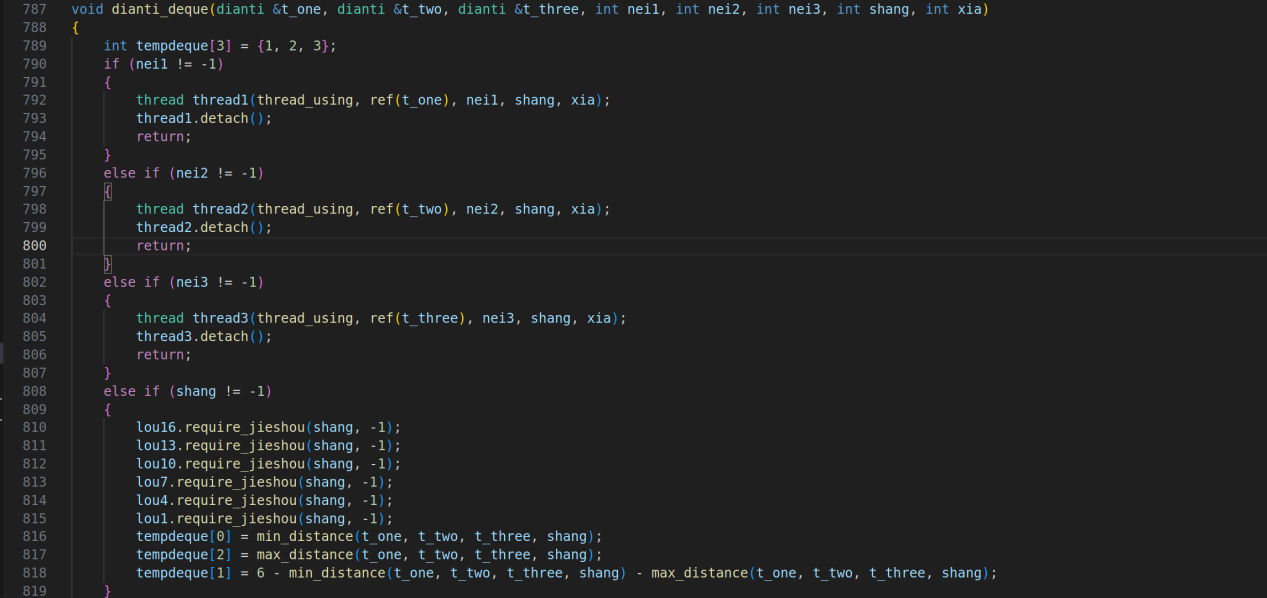
如果是来自楼层的请求，则需要检查becheck，这个全局原子变量仅在单次点击中的三个线程创建时生效，点击之后就赋值为0，并在经过第一个线程之后被赋值为1，赋值为1代表后续的线程的电梯需要检查on\_require\_floor数组，在数组中则需要将数组的数据归零之后处理这个请求，不在数组中代表这个数据已经被上一个线程解决不需要再处理。

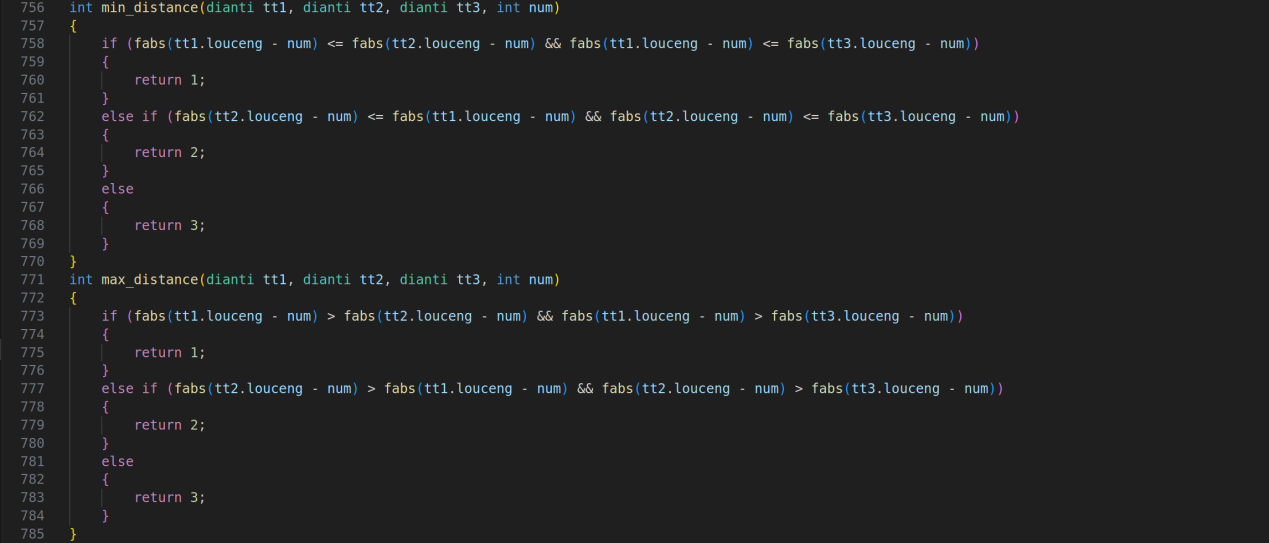
（注意这里必须是原子变量否则快速进行的第二个线程很有可能会在第一个线程将becheck赋值为1之前调用becheck）



—dianti\_deque函数：

接收来自回调函数的所有数据，提取出不是-1的有效数据，然后通过min\_distance和max\_distance两个函数计算距离请求楼层最近的电梯和最远的电梯，若三部电梯中有电梯位置相同，取编号小的为min\_distance返回值，编号大的为max\_distance返回值。然后根据距离目标楼层顺序排序，按顺序启动线程。





### **实验结果及分析 (Experimental Results and Analysis)**

成果见视屏1，模拟电梯运行的程序很好解决了高峰期16楼上来的电梯爆满的问题，上16楼的电梯一定是空载的，并且有能力向下兼容。在正常情况下有良好的处理请求能力，见视屏2。

### **总结与展望 (Conclusion and Future Work)**

本次研究落脚点在于如何让电梯在高峰期提高效率，避免无效运行。并通过这些算法给每个电梯分配任务，让其能在最短的时间内尽可能多的处理所有请求。该代码仅为模拟电梯运行，实际情况下还可以引入载重感应器，满载情况下不再接收来自其他楼层的请求，进一步提升电梯的运行效率。并且还可以将没有纳入考虑的speed变量纳入运行时长进行计算。或者引入时间因素，在指定的高峰期提前准备。

研究当前的电梯运行算法发现，当前电梯有引入大数据的倾向，即根据大数据模型推算高峰楼层，更加贴合实际情况地进行电梯群控调度，使其能贴合各大楼的实际情况，处理高峰期、汇聚、分散等情况。

****参考文献****  
[【算法】电梯调度算法/磁盘扫描算法\_电梯算法和扫描算法的区别-CSDN博客](https://blog.csdn.net/SunshineTan/article/details/80305096)