南昌大学

《计算机综合课程设计》

课程设计

题　　目：电梯调度算法模拟

学生姓名：刘盼

学　　号：6103114110

专业班级：计科 143

指导教师：白似雪

二零一七 年 六　月　 三十 日

目　　录

1.课程设计的目的

(1)使用面向对象思想，设计单个电梯调度算法

(2)实现多个电梯调度管理，为乘客分配最优电梯

2. 课程设计题目描述

说明：电梯调度算法的基本原理是如果在电梯运行的方向上有人要使用电梯，则电梯继续往那个方向上运动，如果电梯中还有人没有到达目的地则继续向原方向运动。具体而言，如果电梯现在朝上运动，如果当前楼层的上方和下方都有请求，则先响应所有上方的请求，然后响应下方的请求，如果电梯向下运动，则刚好相反。

要求：模拟多人在不同楼层同时要求到各自目的地时电梯的响应顺序，写出程序设计思路，给出源代码。

例如：当前楼层是 4，结构体数组中共有 3 个人，A：7→3，B：6→10，C：7→8。输出应该是：

当前楼层为 6，B 进入；

当前楼层为 7，C 进入；

当前楼层为 8，C 出去；

当前楼层为 10，B 出去；

当前楼层为 7，A 进入；

当前楼层为 3，A 出去。

3. 课程设计报告内容

3.1 单电梯解决思路

电梯类具有属性如下:

(1)楼层高度

(2)当前所在楼层

(3)当前运动方向(0 表示停留 1 表示上升 -1 表示下降)

(4)按钮状态 List(两个集合 每层楼上/下按钮是否激活)

(5)上升等候区 Map(Value 为 List，包含乘客目的楼层)

(6)下降等候区 Map(Value 为 List，包含乘客目的楼层)

(7)电梯内乘客数量

电梯类方法如下:

(1)run:状态机，执行一次，消耗一个单位时间，电梯可上升/下降一层，或停留原地，内部状态发生改变
(2)up_chooice:电梯上升状态时，消耗一个单位时间后，根据内部状态，决定下一次的运动方向
(3)down_chooice:电梯上升状态时，消耗一个单位时间后，根据内部状态，决定下一次的运动方向
(4)getin:电梯运动到某一层楼后，乘客进入电梯，内部状态发生改变
(5)getout:电梯运动到某一层后，到达目的地的乘客出去
(6)addpassenger:参数为起始楼层、目的楼层，判断参数正确后，修改电梯属性 -- 按钮列表、等候区 Map

运行流程图:

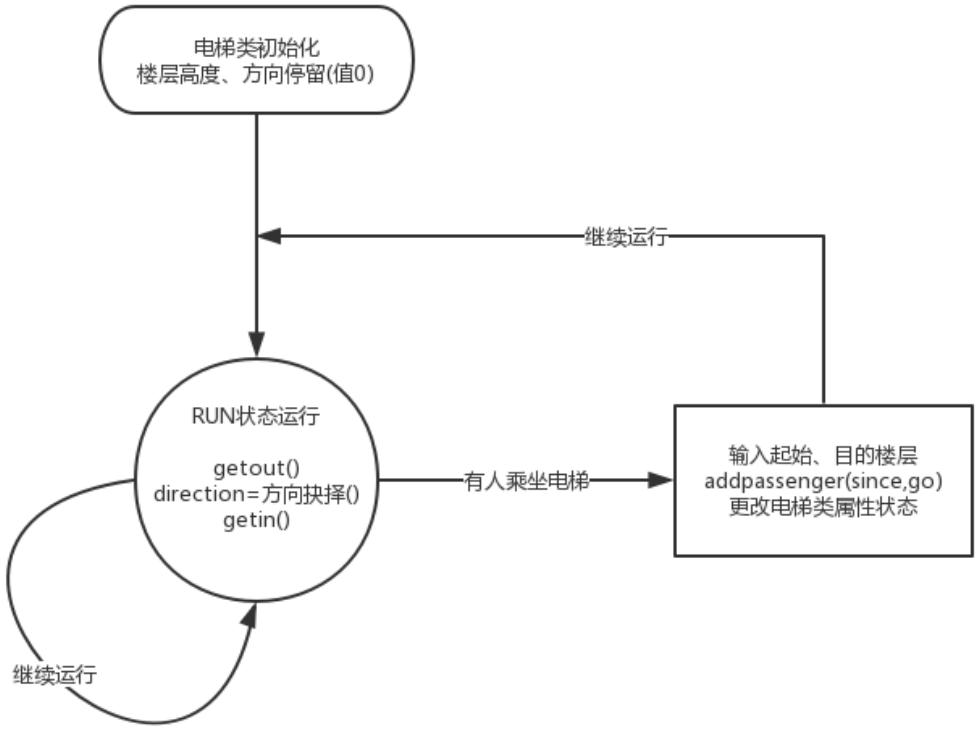


图 1.单电梯类运行流程图

流程解释:
(1)类初始化

```
class lift:
    def __init__(self,height):
        self.height = height
        self.now = 0     #当前楼层
```

```
        self.direction = 0    #运动方向 上为1 下为-1 停留为0
        self.button = [set(),set()]    #按键状态字典 记录每一层楼上下按钮是否被按下
        self.target ={k:[] for k in range(height)}      #记录电梯中乘客的目的楼层
        self.waiting_up = {k:[] for k in range(height)}      #每层楼等候区 []中为等候人员目标楼层
        self.waiting_down = {k:[] for k in range(height)}      #每层楼等候区 []中为等候人员目标楼层
        self.number = 0
        self.info = ''
        self.pad = lambda s: s + ( 11- len(s)) * ' '
```

(2)RUN 状态运行

```
def run(self):
    self.now += self.direction
    if self.direction == 1:
        self.info += '电梯上升中\n'
        self.out()
        self.direction = self.up_choice()        #方向抉择
        self.getin()
        pass
    if self.direction == -1:
        self.info += '电梯下降中\n'
        self.out()
        self.direction = self.down_choice()    #方向抉择
        self.getin()
        pass
    if self.direction == 0:
        self.info += '电梯停留在 %d 层\n' % self.now
        self.direction = self.up_choice()    #停留状态调用上升抉择
        self.getin()
```

(3)乘客乘坐

```
def addpassenger(self, since, go):
```

```
    if since == go or since not in range(self.height) or go not in
range(self.height):
        self.info += 'orz 你可能想跳楼\n'
    else:
        if since < go:
            self.button[1].add(since)              #上升按钮被按下
            self.waiting_up[since].append(go)      #加入此层楼上升等候
区
        else:
            self.button[0].add(since)              #下降按钮被按下
            self.waiting_down[since].append(go) #加入此层楼下降等候区
```

(4)方向抉择

以上的方法都十分简单，getin()、getout()也被我省略，因为两个方法并不复杂，只需要修改类的属性，将乘客加入电梯内，从等候区移除，消除楼层按钮。此算法的核心在于方向抉择，方向抉择函数决定在当前状态下电梯下一步应该运行的方向，先看状态转变图

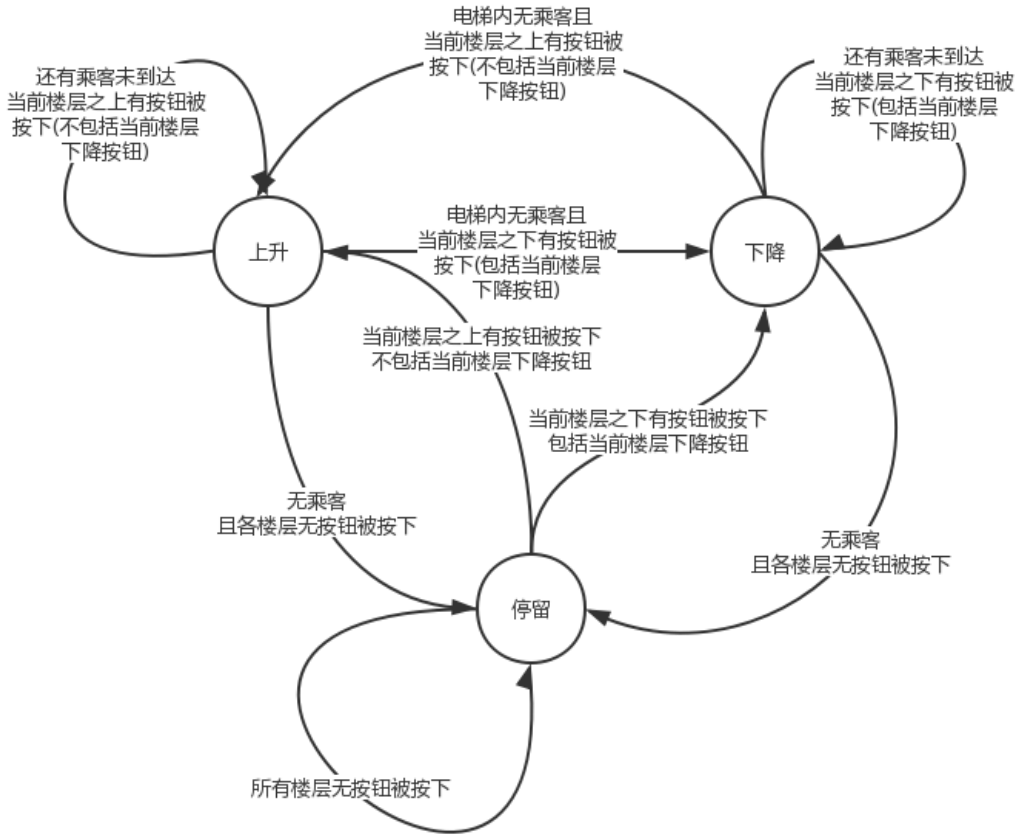


图 2.电梯状态转换图

可以看出上升状态的抉择函数与停留状态的抉择函数相同，所以在 RUN 运行中两者调用同以抉择函数

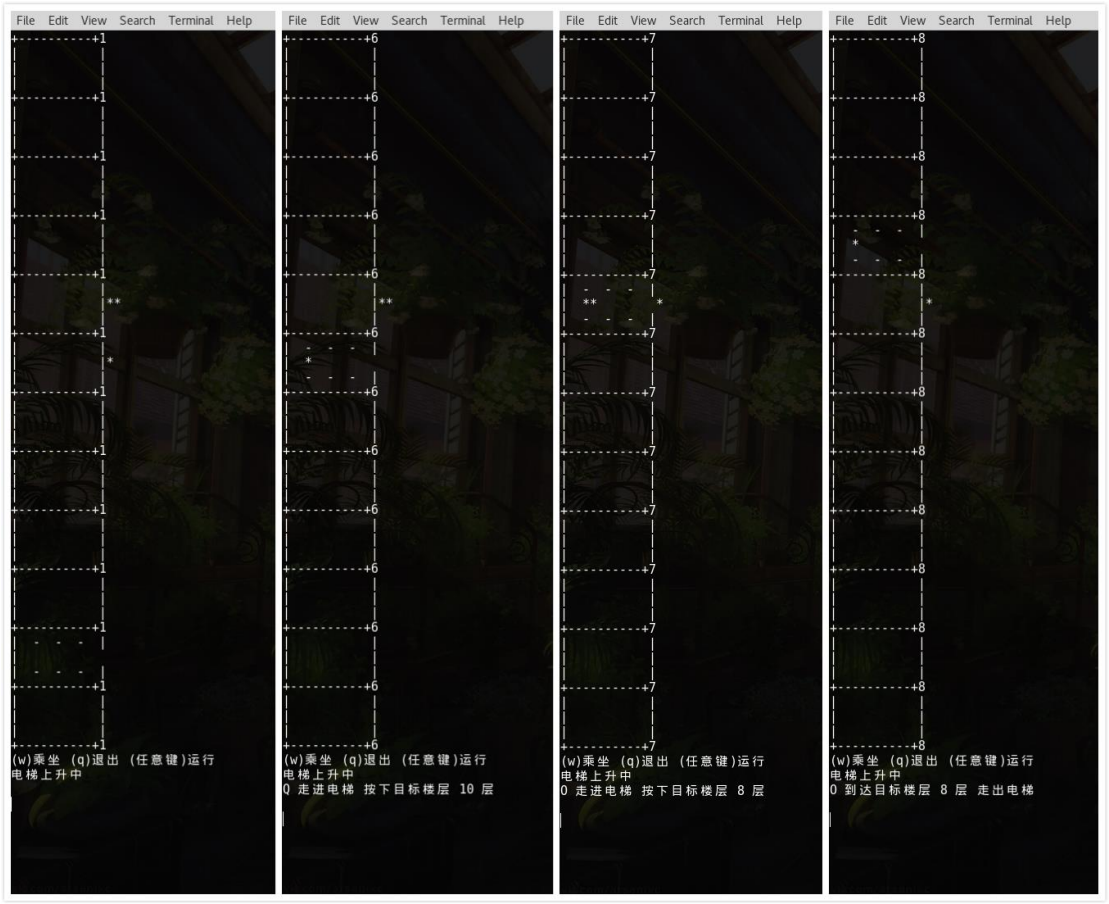
抉择函数如下，需要特别主要当前楼层的包含情况：

```
def up_choice(self):
    for floor in range(self.now,self.height):
        if self.target[floor] or floor+1 in self.button[0] or floor in self.button[1]:        #如果有乘客的要上去 或者上面有人按下上/下按钮
            return 1    #返回继续上升

    for floor in range(self.now + 1):
        if self.target[floor] or floor in self.button[0] or floor in self.button[1]:        #如果有乘客的要上去 或者上面有人按下上/下按钮
            return -1    #返回下降
    return 0    #电梯停留不动

def down_choice(self):
    for floor in range(self.now + 1):
        if self.target[floor] or floor in self.button[0] or floor-1 in self.button[1]:        #如果有乘客的要下去 或者下面有人按下上/下按钮
            return -1    #返回继续下降

    for floor in range(self.now , self.height):
        if self.target[floor] or floor in self.button[0] or floor in self.button[1]:        #如果有乘客的要上去 或者上面有人按下上/下按钮
            return 1    #返回上升
   return 0    #电梯停留不动
```

3.2 单电梯运行结果

(1) 使用第三方库 curses 开发交互界面

(2) 运行结果如图


File Edit View Search Terminal Help
(w)乘坐 (q)退出 (任意键)运行
电梯上升中
(w)乘坐 (q)退出 (任意键)运行
电梯上升中
Q 走进电梯 按下目标楼层 10 层
(w)乘坐 (q)退出 (任意键)运行
电梯上升中
O 走进电梯 按下目标楼层 8 层
(w)乘坐 (q)退出 (任意键)运行
电梯上升中
O 到达目标楼层 8 层 走出电梯

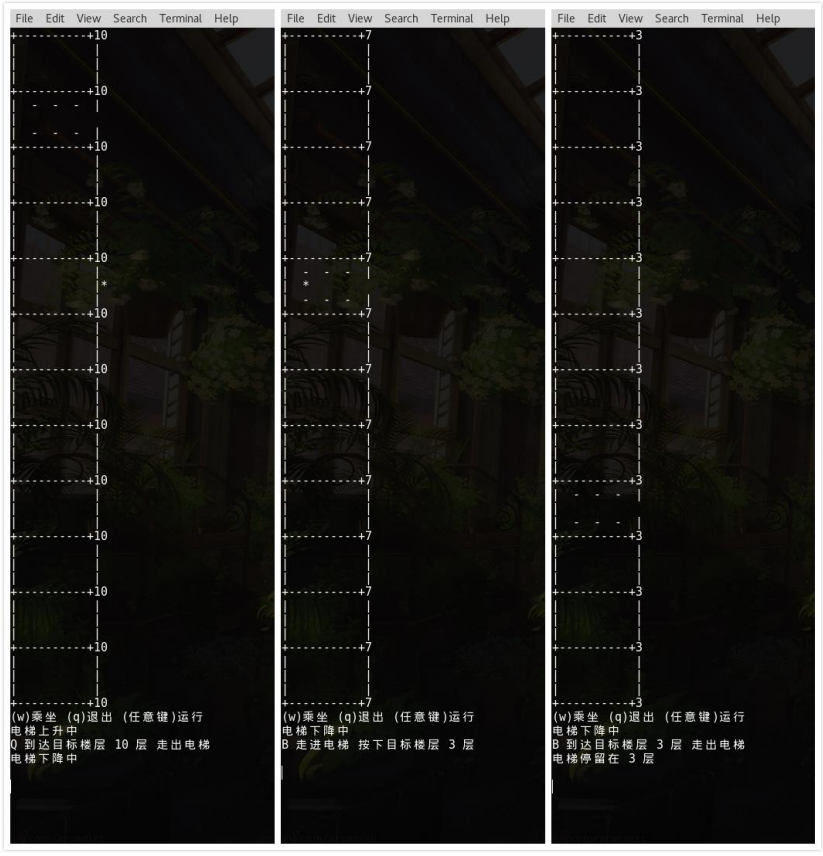

File Edit View Search Terminal Help
(w)乘坐 (q)退出 (任意键)运行
电梯上升中
Q 到达目标楼层 10 层 走出电梯
电梯下降中
(w)乘坐 (q)退出 (任意键)运行
电梯下降中
B 走进电梯 按下目标楼层 3 层
(w)乘坐 (q)退出 (任意键)运行
电梯下降中
B 到达目标楼层 3 层 走出电梯
电梯停留在 3 层

图 3. 单电梯类运行结果

3.3 多电梯调度算法

多电梯调度由上面设计的单电梯类组成，只是在外面封装了一下，编写一个 Manage 类，当由乘客需要乘坐电梯时，由 Manage 响应，然后根据优先级判决，选择一个最快的电梯，调用电梯类的 addpassenger(since,go)，将乘客加入该电梯的等候区，不影响其他电梯，各个电梯的运行互不干扰

运行流程如图:

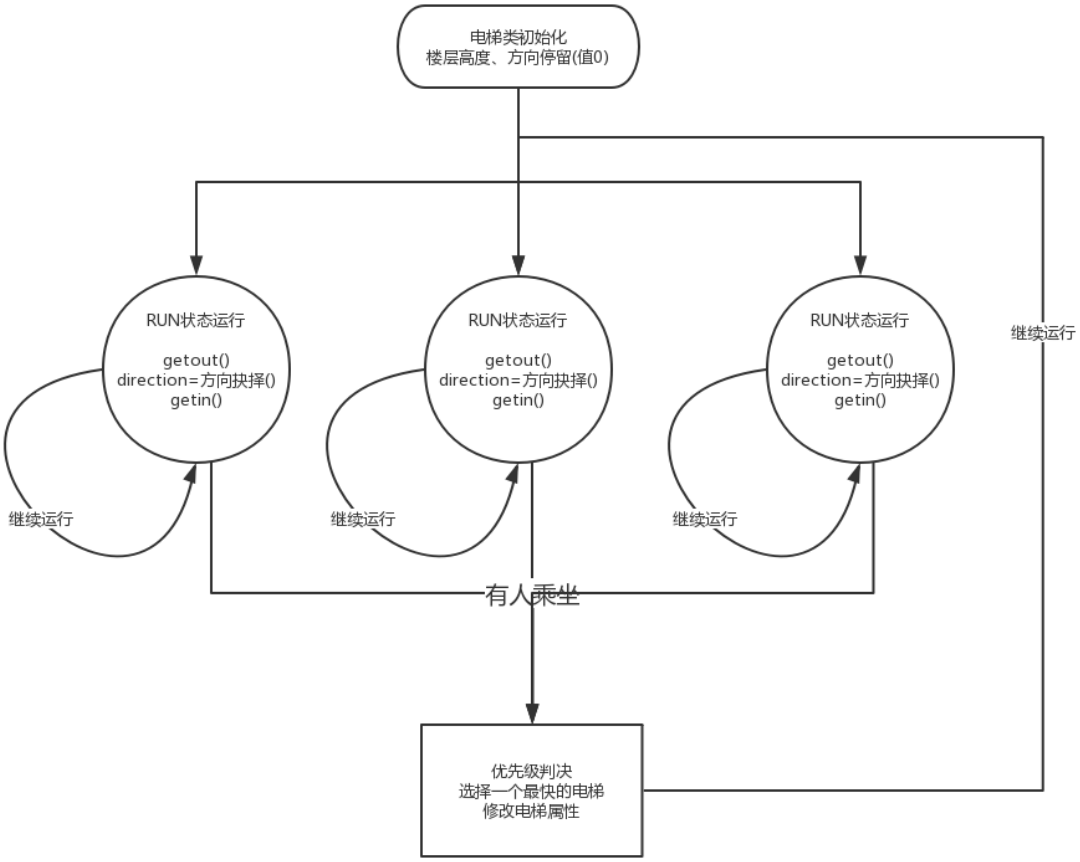


图 4.多电梯运行流程图

优先级抉择:

为每个电梯类增加优先级判定函数，传递给此方法乘客的起始楼层、乘坐方向(-1 or 1)；此方法会根据电梯当前所在位置、运行方法，计算返回接到乘客的预计路程。

所以返回值越小，优先级越大，乘客可以越快乘坐，当一个乘客需要乘坐时，将参数传递给每一个电梯实例的优先级判定函数，然后选择返回值最小的电梯，将乘客添加进去，其余的电梯将不会响应此乘客

优先级判定函数分析:

优先级判定函数是外层 Manage 类的核心

分析之后，优先级的影响参数有 4 个 - 电梯/乘客所在位置、电梯/乘客运行方向

不考虑上下方向，分为四类情况

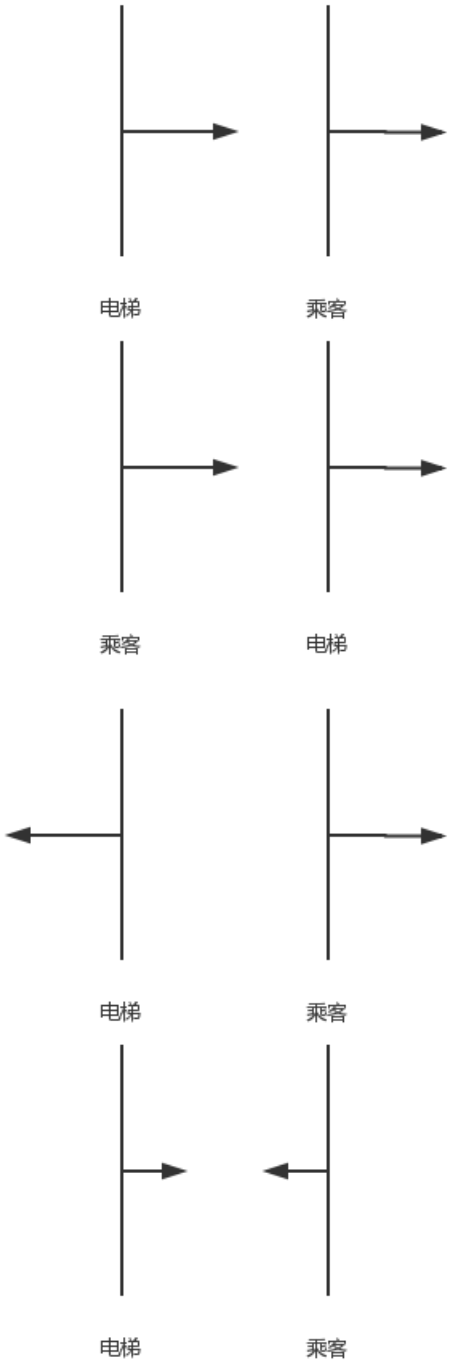


图 5.电梯与乘客运动关系图

如上，上面四种情况忽略了绝对方向，只考虑电梯和乘客的相对位置、相对方向

大概分析一下可以知道，第一种情况，优先级最大(最快可乘坐)

接下来分析其他情况电梯运行的路程:

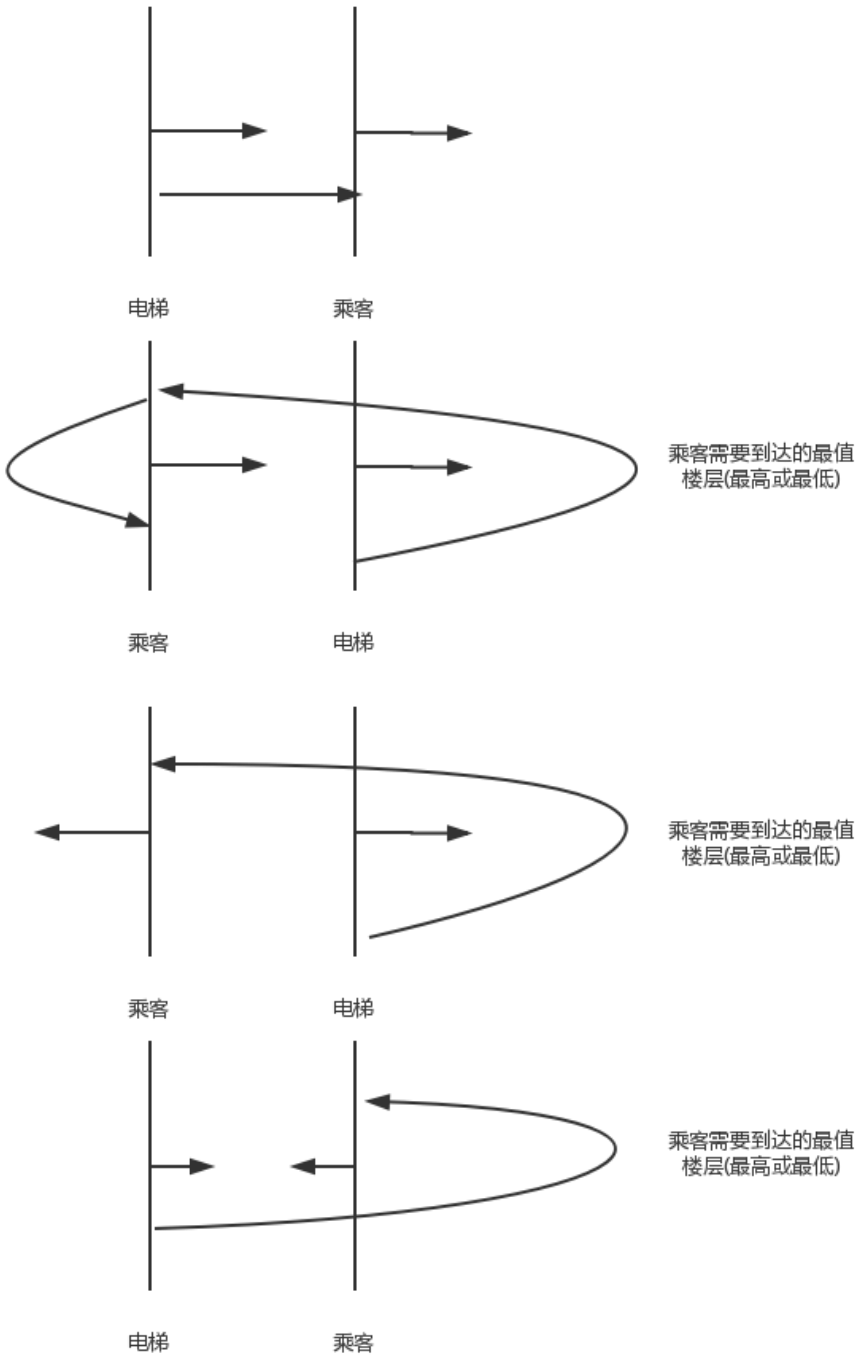


图 6. 电梯运行路程图

图中有个最值，它指什么？

最值表示电梯应该到达的边界楼层，比如电梯上升中，6 楼有一位乘客需要到 2 楼，而电梯内部有乘客要达到 8 楼，9 楼有个乘客需要到达 4 楼。

那么此种情况，电梯会一直上升到 9 楼，9 楼就是边界楼层

那么如何求边界楼层，先考虑方向

电梯在上升中，边界楼层应该是

max(电梯内乘客要到达的楼层)

max(上/下按钮被激活的楼层)

max(即将上电梯的乘客的目的楼层)

上面三者的最大值

下降方向类似，但是在编写函数时应该忽略掉方向因素，采用相对方向

接着来分析上面四种情况电梯接到乘客的路程

第一种情况，返回路程 = abs(电梯楼层 - 乘客楼层)

第二种情况，有一个反向最值，可以根据相反方向的等待乘客目标楼层求反方向最值，但是变化比较大，暂时忽略，加个随机值(不超过电梯此时高度)

返回路程 = 2 * abs(电梯到达最值 - 电梯楼层) + abs(电梯楼层 - 乘客楼层) + 随机值

第三种情况

返回路程 = 2 * abs(电梯到达最值 - 电梯楼层) + abs(电梯楼层 - 乘客楼层)

第四种情况，需要注意，最值可能介于乘客与电梯之间，但此时电梯不能回头，需要继续像乘客运动，然后接到乘客转换方向

如果发生此情况

返回路程 = abs(电梯楼层 - 乘客楼层)

否则

返回路程 = 2 * abs(电梯到达最值 - 乘客楼层) + abs(电梯楼层 - 乘客楼层)

来看一下求电梯最值的函数:

```
def get_max_floor(self,since):     #获取当前方向需要到达的最大楼层，取决于乘客目的地的最值
    for floor in range(self.height)[::-self.direction]:    #楼层倒顺序
        if self.target[floor] or self.waiting_up[floor] or self.waiting_down[floor]: #有乘客上去 上方有人在等候
            return self.real_max(floor)     #返回最值
```

```
    return -1     #表明电梯上没有乘客 返回-1 因为此时电梯在运动 如果上面的情况都没发生 就是发生了错误
```

最值取决于电梯的运行方向，这里用了一个小技巧，python 中的 List 切片 - 可以看到第二行做了一个楼层的迭代，当 direction 为 1 时，会倒序迭代，从最高楼层开始做迭代，直到 target[floor]不为空(即有乘客在这层下电梯)或 楼层的上/下按钮被触发，当这两种情况发生时，电梯需要一直运动到此楼层才能改变方向，此楼层即为最值。

但是，可以看到函数返回了 real_max(floor)，说明此时的 floor 可能也不是最值。

想象一下此时求出的最值为 9，但是有个乘客在 8 楼等着上电梯，他的目标楼层是 11 楼，这样的话，电梯必须到达 11 楼。

这就是上面所说的 max(即将上电梯的乘客的目的楼层)

Real_max 这个函数就在做这样的判断，如下:

```
def real_max(self,floor):
    try:
        if self.direction >0:
            return max(max(max([self.waiting_up[i] for i in range(self.now,self.height)], key=lambda x: max(x) if x else -1))  ,floor)
        else:
            return min(min(min([self.waiting_down[i] for i in range(self.now)], key=lambda x: min(x) if x else self.height)) ,floor)
    except:
        return floor
```

如下，为优先级判定函数:

```
def priority(self,since,go,direction):
    if self.direction == 0:
        return abs(since -self.now)  #电梯停留 也是最有决策 路程为两者之差
    if self.direction == direction:
        if self.direction * (since - self.now) > 0:      #最优决策，乘客可以很快乘坐
            return abs(since -self.now)     #返回相差的楼层
```

```
        else:
            return 2 * abs(self.get_max_floor(since) - self.now) +
abs(self.now - since) + random.randint(0,since)
    else:
        if (self.now - since) * self.direction > 0:    #乘客与电梯背对
而行
            return 2 * abs(self.get_max_floor(since) - self.now) +
abs(self.now - since)
        else:   #电梯与乘客相对而行
            max_floor = self.get_max_floor(since)
            if (max_floor - self.now) * (max_floor - since) < 0:  #最
值位于电梯与乘客之间
                return abs(self.now - since)
            else:
                return 2 * abs(self.get_max_floor(since) - since) +
abs(self.now - since)
```

可以看到四种情况都在里面，第四种情况再次做了一个判断

判断是否在电梯与乘客之间，使用一个数学技巧

判断(最值 - 电梯楼层) * (最值 - 乘客楼层)的符号

为正数时，即不在乘客与电梯之间，返回 2 * abs(电梯到达最值 - 乘客楼层) + abs(电梯楼层 - 乘客楼层)

为负数时，返回 abs(电梯楼层 - 乘客楼层)

电梯的选择:

核心的优先级判定完成了，再看看 manage 中如何为乘客选择电最优梯:

```
def addchoose(self,since,go):
    self.info += '有人乘坐电梯 %d -> %d\n'%(since , go)
    direction = 1 if go > since else -1    #方向
    prioritys = []
    for lift in self.lifts:
        prioritys.append(lift.priority(since,go,direction))

    self.lifts[prioritys.index(min(prioritys))].addpassenger(int(sinc
    e),int(go)) #分配优先值最小的电梯给乘客
```

```
self.info += '分配最优电梯 第 %d 号电梯 进入等候区\n' %
prioritys.index(min(prioritys))
```

可以看到，Mange 接到乘客请求后，会迭代电梯，分别求优先级，然后将乘客加入优先级最大的电梯等候区

3.4 多电梯调度运行结果

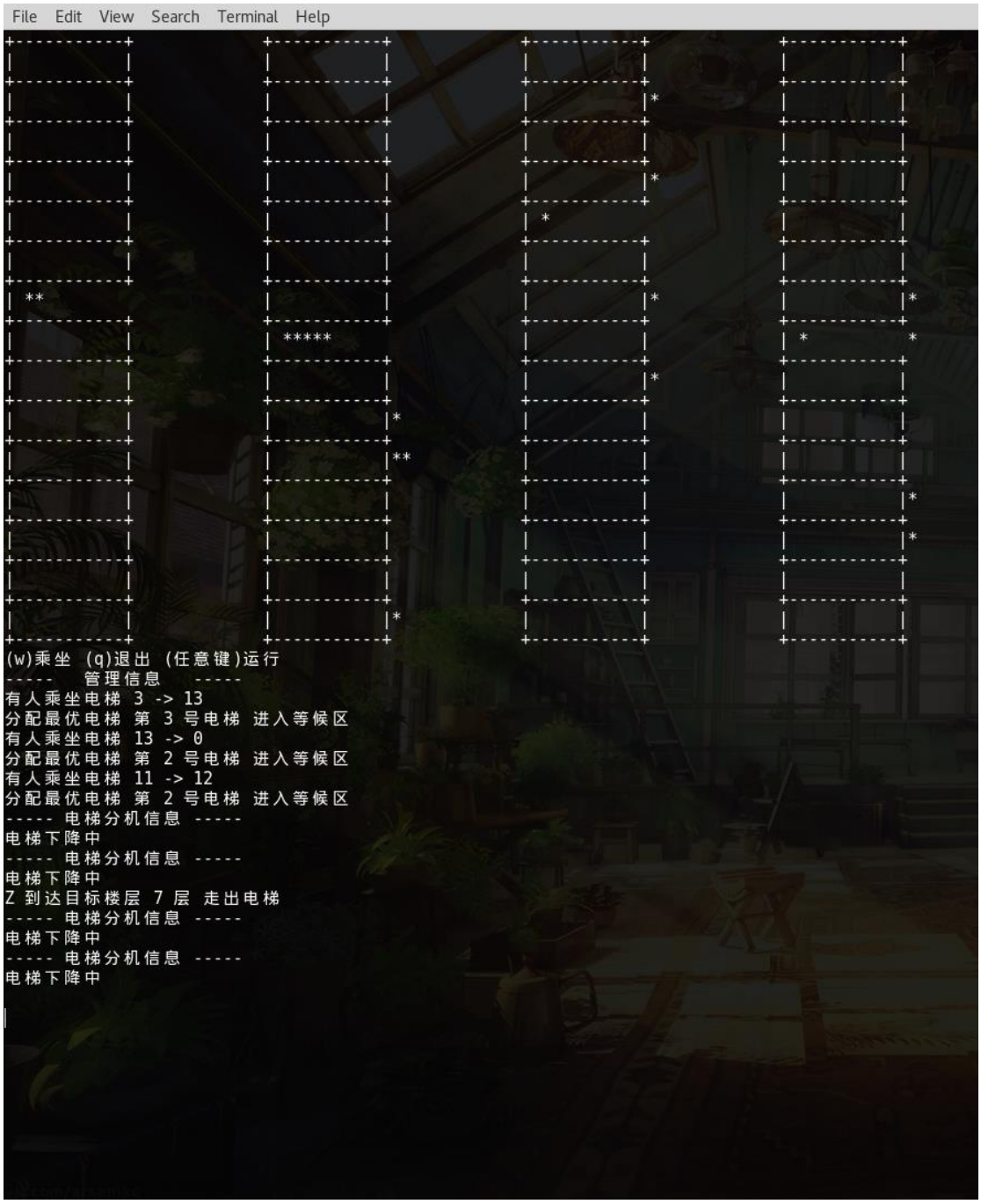


图 7.多电梯运行结果

运行视频

多电梯调度.webm

4. 结论

单电梯类设计基于状态机，状态转变根据电梯实际运行的状态转变图设计，与现实中大多数电梯工作方式相同

多电梯调度算法使用贪心法的思想，为每一位乘客安排当前状态下最优的，每一步最优，总体自然不会太差

经过长时间测试，调度算法大部分时间为较优，四台电梯负载均衡

需要注意，我们这里是理想状态，而且忽略了一些不定因素，比如:

(1)电梯开门关门需要时间，在模拟中忽略了

(2)在优先级判定之后，可能会有新的乘客到来，而导致最值改变，那么如果要达到完美，需要在最值改变之后，再次计算每个等待乘客最优的电梯，重新分配，考虑工作量比较大，暂时忽略

参考书目:

[1] Magnus Lie Hetland，Python 算法教程，人民邮电出版社，2016-1-1