**实验报告**

题目：电梯模拟

姓名：伊昕宇 PB16001749

1. **需求分析**
2. 设计一个电梯模拟系统，模拟电梯和乘客的动作。通过flash以动画的方式呈现。采用编程语言Action Script 3。
3. 画好电梯（开门、关门），人（行走，站立，被选定），钟表等动画作为影片剪辑元件，画好背景。做好其他信息提示和信息输入的文本框，用按钮影片剪辑进行交互控制。
4. 用户可以选择开始模拟，可以选择系统自动生成人，可以点击随机生成一个人，可以输入一个人的基本信息而创建一个指定的人。对于正在排队的人，用户可以点击一个人而显示该人的基本信息。对于模拟的时钟用户可以选择从开始模拟时计时，或选用系统时间作为模拟。用户可以通过修改代码中的常量值改变特定环节的进行时间或其他方面内容。模拟动画还将提供电梯的当前层，电梯载重，已成功到达目的楼层的人数，等不及而离开的人数，和所有人总等待时间。
5. 除交互部分外，模拟动画会展示每个环节，人们走到电梯会按下相应按钮，每个人都有一个最大等待时间，超过这个时间他会离开。电梯在一个楼层超过一定时间后会自动回到1楼等待人们按下按钮。人和电梯的动作均需要耗费一定的时间。
6. 点击开始模拟即可一直运行下去。通过交换元件图片可以用其他图片作为人、电梯、背景。一些主要的常量也在代码的开头可以修改。模拟动画通过人的出现触发其他一系列事件。
7. **概要设计**
8. 设定链表的抽象数据类型定义：

**ADT** LinkList {

**数据对象**：D = { ai | ai ∈ NumberSet , i = 1 , 2 , ... , n >= 0 }

**数据关系**：R1 = { < ai-1 , ai > | ai-1 , ai ∈ D , i = 2 , ... , n }

**基本操作**：

InitList ( L : Object )

操作结果：构造一个空链表L。

RemoveQueue( myQueue: Object , myMan: Man , mood: Number = 0 )

初始条件：myQueue和myMan存在，myMan在myQueue中。

操作结果：只有前两个参数或第三个参数mood = 0时，在queue中删除 man并前移后面的人（使他们走到上一个人的位置）

mood = 1时只删除链表中myMan这个人。

QueueTraverse ( myQueue: Object , visit: Function )

初始条件：myQueue已存在。

操作结果：从头到尾依次对myQueue中的元素调用visit ( )

EnQueue ( o: Object , addo: Object )

初始条件：o和addo已存在。

操作结果：addo插入队列o。

DeQueue ( o: Object ) : Object

初始条件：o存在且不为空。

操作结果：删除o队头并返回。

makeOrder ( myQueue : Object , num: Number )

初始条件：myQueue存在且为num楼层的链表。

操作结果：将myQueue中的人的位置重新排列。

allQueueInOrder ( )

初始条件：各楼层队列都存在。

操作结果：将各楼层队列中的人的位置重新排列。

} **ADT** LinkList ;

1. 设定人的抽象数据类型为：

**ADT** Man {

**数据对象**：D = { ai | ai ∈ NumberSet , i = 1 , 2 , ... , n >= 0 }

**基本操作**：

removeMan ( )

初始条件：人的移除列表已存在。

操作结果：将位于人的移除列表中、舞台上以实例化的人在显示列表中 移除。

transMan ( )

初始条件：电梯的链表已存在。

操作结果：将位于电梯链表中、不在电梯的显示列表中、舞台上以实例 化的人在显示列表中移除，并在电梯的显示列表中添加一个 人的实例。

timedown ( )

初始条件：各个楼层链表已存在。

操作结果：将所有排队的人剩余等待时间减少一秒并刷新显示。

move(mc: MovieClip, fromx: Number, fromy: Number, tox: Number, toy: Number, usetime: Number)

初始条件：将Man转换为MovieClip类型。

操作结果：将该人从( fromx , fromy )移动到( tox , toy )，用时usetime。

showInformation ( tar: Man )

初始条件：tar已存在。

操作结果：显示tar这个人的信息。

newManArrive ( newman: Man )

初始条件：newman已存在。

操作结果：添加显示对象newman，楼层人数加一，插入该楼层队列， 播放行走和等待的动画，按下相应叫梯按钮。触发电梯移动。

makeMan(manweight: Number, from: Number, to: Number, maxWaitTime: Number, restWaitTime: Number): Man

操作结果：创建一个Man并返回，该人的属性在参数中。

onWalking ( )

初始条件：人已存在。

操作结果：播放人正在行走的动画。

onStanding ( )

初始条件：人已存在。

操作结果：播放人正在站立的动画。

onChoosing ( )

初始条件：人已存在。

操作结果：播放人被选中的动画。

waitLineMove( )

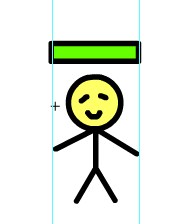
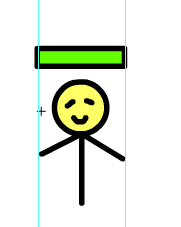
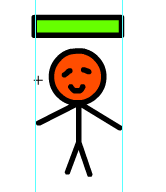
初始条件：人已存在。

操作结果：播放人时间条减少的动画。

} **ADT** Man ;

该人与人的影片剪辑元件链接。元件为Action Script导出。类为Man，基类 flash.display.MovieClip。

1. **详细设计**
2. **人的影片剪辑元件**

1人静止的样子（未被选中） 2人被选中的样子 3人行走的样子（动画）

首先新建了一个影片剪辑元件，为Action Script导出，定义类名为Man。

第一帧为一个空白帧，用于存放人的各种属性和方法，AS代码如下：

import flash.display.MovieClip;

//此帧用来存放人的属性

var manweight: Number;

var from: Number;

var to: Number;

var maxWaitTime: Number;

var isInLift: Boolean;

var restWaitTime: Number;

function waitLineMove(): void {

this.timeLine.x = (1 - Number(this.restWaitTime / this.maxWaitTime)) \* (parent as MovieClip).WAITLINELENGTH; //时间条右移

//trace("debug:时间条x"timeLine.x);

}

第二帧为人静止站立的帧，见图1，存放了一些人的方法，代码如下：

stop();

function onWalking(): void { //播放行走动画

gotoAndPlay(3);

}

function onStanding(): void { //播放静止动画

gotoAndStop(2);

}

function onChoosing(): void { //播放选中动画

gotoAndStop(7);

}

第三到六帧为人行走的动画，见图2为其中一帧，在第六帧有AS代码如下，实现 动画循环

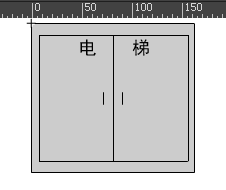
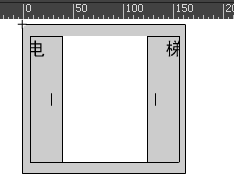
gotoAndPlay(3); //播放第3帧以实现循环的行走动画

第七帧为人被选中的帧，见图3，代码如下：

stop(); //此帧的人被选中

对于图层，第一层存放了人的图片，第二层为等待时间条的影片剪辑的一个实例， 实例名为timeLine，第三层为遮罩层，包含与等待时间条一样大小的一个图片， 对第二层进行遮罩，这样移动timeLine时形成时间条减少的动画效果。第四层设 计了等待条的黑色边框。

1. **电梯的影片剪辑元件**

第一帧用来存放电梯的属性和方法，代码如下

var isOpen: Boolean = false;

function onOpening(): void {

gotoAndPlay(2);

}

function onClosing(): void {

gotoAndPlay(26);

}

stop();

第二到二十五帧为电梯开门的动画，如图2，第二十五帧代码如下

this.isOpen = true;

stop();

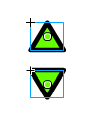
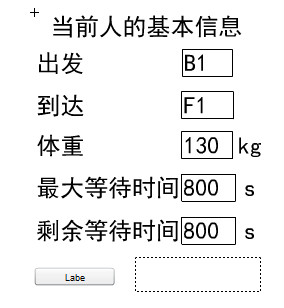
第二十五到五十帧为电梯关门的动画，第五十帧代码如下

this.isOpen = false;

stop();

一共五个图层，通过遮罩和补间形成开关门动画。

1. **基本信息的影片剪辑元件**



只有一帧一个图层，用于显示一个人的基本信息，代码如下

this.confirm.label = "生成指定的人";

该帧中还定义了六个文本框及一个按钮的实例名。

**4. 电梯按钮的影片剪辑元件**

该影片剪辑（图一）嵌套了两个电梯按钮影片剪辑的实例（图二），实例名分别为B1UP B1DOWN。在每个小按 钮中存在两帧，第一帧为绿色（点亮）状态，第二帧为白色（熄灭）状态。在整 个电梯按钮影片剪辑元件中，共有两帧，第一帧代码如下：

import flash.display.MovieClip;

switch ((parent as MovieClip).up[1]) {

case 1:

B1UP.gotoAndStop(1);

break; //上升灯亮

case 0:

B1UP.gotoAndStop(2);

break; //上升灯灭

}

switch ((parent as MovieClip).down[1]) {

case 1:

B1DOWN.gotoAndStop(1);

break; //上升灯亮

case 0:

B1DOWN.gotoAndStop(2);

break; //上升灯灭

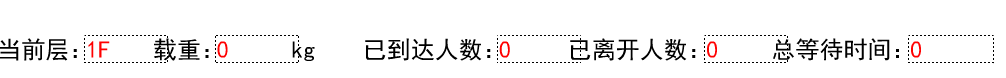
}

第二帧为了不断刷新第一帧代码，刷新按钮灯显示状态，代码如下：

gotoAndPlay(1); //形成循环刷新状态

按钮一共复制了五个，不在重复代码。

**5. 模拟信息的影片剪辑元件**



共两帧，已为一些文本框并命名。第一帧主要为了显示五个信息，代码如下

var cf: Number = (parent as MovieClip).liftCurrentFloor;

var wei: Number = 0;

switch (cf) {

case 0:

liftFloor.text = "1B";

break;

case 1:

liftFloor.text = "1F";

break;

case 2:

liftFloor.text = "2F";

break;

case 3:

liftFloor.text = "3F";

break;

case 4:

liftFloor.text = "4F";

break;

}

var p: Object = (parent as MovieClip).liftQueue;

while ((p = p.next) != undefined) {

wei += p.manweight;

}

liftWeight.text = wei.toString();

succ.text = (parent as MovieClip).manSucceed.toString();

fail.text = (parent as MovieClip).manLeave.toString();

var temp: Number = 0;

for (var ii: int = 0; ii < 5; ii++) {

temp += (parent as MovieClip).floorManNum[ii]; //总人数

}

(parent as MovieClip).totalWaitTime += (1 / 24) \* temp; //总时间

sec.text = (parent as MovieClip).totalWaitTime.toString();

第二帧为了刷新第一帧的显示，形成循环，代码如下

gotoAndPlay(1); //形成循环 刷新数字

**6. “开始模拟”文字的影片剪辑元件**

****

仅仅为了播放开始模拟文字。第一帧为空白帧代码如下

stop();

function start(): void {

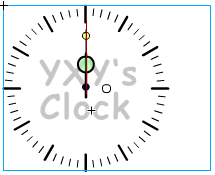
gotoAndPlay(2); //播放动画

}

第47帧为空白帧，代码如下

stop();

**7. 表盘的影片剪辑元件**

含有两帧，用于旋转表针，以定义了三种表针的实例名，第一帧代码如下

var ns: Date = new Date();

if ((root as MovieClip).gameMood == true) { //显示游戏时间 否则显示系统时间

ns.setHours(0);

ns.setMinutes(0);

ns.setSeconds((root as MovieClip).time);

}

var ds: Number = ns.getSeconds();

var dm: Number = ns.getMinutes();

var dh: Number = ns.getHours();

second\_hand\_mc.rotation = Number(ds) \* 6;

minute\_hand\_mc.rotation = (Number(dm) + Number(ds / 60)) \* 6;

hour\_hand\_mc.rotation = Number(dh + Number(dm) / 60) \* (360 / 12);

第二帧为了刷新表盘的显示，形成循环

**8. 数字时间的影片剪辑元件**

在表盘下面显示数字时间，可显示系统时间或模拟进行的时间，与表盘同步，第 一帧代码如下：

import flash.display.MovieClip;

var nowday: Date = new Date();

if ((root as MovieClip).gameMood == true) { //显示游戏时间 否则显示系统时间

nowday.setHours(0);

nowday.setMinutes(0);

nowday.setSeconds((root as MovieClip).time);

}

function add(num: Number): String {

if (num == 0) return ("00");

else if (num < 10) return ("0" + num.toString());

else return (num.toString());

}

date\_hour\_txt.text = add(nowday.getHours());

date\_minute\_txt.text = add(nowday.getMinutes());

date\_second\_txt.text = add(nowday.getSeconds());

第二帧为了刷新第一帧形成循环，代码如下

gotoAndPlay(1);

//形成循环 不断刷新时间显示

**9. 主时间轴**

主时间轴只有一帧，完成主要功能。一些常量和变量定义如下：

import flash.events.MouseEvent;

import flash.display.MovieClip;

import flash.text.engine.TabAlignment;

import flash.display.DisplayObjectContainer;

import flash.events.Event;

const WAITTIMEMIN: Number = 10; //随机的最小等待时间（多少s）

const WAITTIMEMAX: Number = 399; //随机的最大等待时间（多少s）

const WEIGHTMIN: Number = 30; //随机的人的最低重量（kg）

const WEIGHTMAX: Number = 200; //随机的人的最高重量（kg）

const MANBORNX: Number = 1250; //人出生时的x位置

const MANLEAVEX: Number = 1250; //人离开时的x位置

const QUEUELINE: Number = 250; //排队的第一个人站的x位置-100的值

const WALKTOLIFTTIME: Number = 1000; //走到电梯所需要的时间数

const LEAVETIME: Number = 1000; //离开时行走的时间

const LIFTCLO: Number = 40; //电梯灵敏度，毫秒

const SLEEPTIME: Number = 60000; //多长时间没人坐电梯，电梯自动回到1楼

const WAITLINELENGTH: Number = 85; //等待显示条 的长度

const MOVEFRONTTIME: Number = 500; //前面的人离开了向前走的时间

const LIFTFLOORTIME: Number = 2000; //电梯层之间的移动时间

const GETINLIFTTIME: Number = 1000; //走进电梯的时间

const LIFTWAITTIME: Number = 1000; //人进电梯之后电梯的等待时间

const MANLEAVELIFTTIME: Number = 1000; //人下电梯之后走动时间

const ORDERTIME: Number = 1000; //每隔多长时间拍一次队

const RANDOMMAX: Number = 20; //自动生成的最大间隔时间

const RANDOMMIN: Number = 2; //自动生成的最小间隔时间

const LIFTX: Number = 100; //进电梯的x

var beginnum: Number;

var time: Number = 0; //以s为单位

var manOnChoosing: Man; //当前被选中的人

var maninit: Boolean = false; //是否有选中的人了

var liftCurrentFloor = 1; //电梯当前层

var liftCurrentDir = 2; //2表示静止 0向上 1向下

var liftTime: Number;

var startCounting: Boolean = false; //电梯开始因为静止而计时

var liftcount: Number; //保存settimeout

var cannotwaitTime: Number; //保存settimeout

var manLeave: Number = 0; //等不及而离开的人

var isStill: Boolean = true; //电梯是否静止

var manSucceed: Number = 0; //成功到达的人

var isCheck: Boolean = false; //电梯是否被检查过 在该层状态

var back1: Boolean = false; //回到1层但不开门的指示

var gameMood: Boolean = true;

var totalWaitTime: Number = 0; //总等待时间

var automan: Number;

var flagclick: Number;

完成了一些注册元件侦听鼠标事件，初始化一些数据结构，初始化一些数值和显 示，代码如下：

clockMoodSelect.addEventListener("click", changeMood);

begin\_btn.label = "开始模拟";

begin\_btn.addEventListener("click", begin);

random\_btn.label = "随机生成一个人";

random\_btn.addEventListener("click", randomMakeMan);

//创建事件链表

var upEventList: Object = new Object();

var downEventList: Object = new Object();

//创建五个楼层链队列

var B1Queue: Object = new Object();

var F1Queue: Object = new Object();

var F2Queue: Object = new Object();

var F3Queue: Object = new Object();

var F4Queue: Object = new Object();

var liftQueue: Object = new Object();

B1Queue.tail = B1Queue; //队尾指针

F1Queue.tail = F1Queue;

F2Queue.tail = F2Queue;

F3Queue.tail = F3Queue;

F4Queue.tail = F4Queue;

var floorManNum: Array = new Array(); //创建数组记录每层排队的人数

var up: Array = new Array(0, 0, 0, 0, 0); //没有上升需求

var down: Array = new Array(0, 0, 0, 0, 0); //没有下降需求

var removeList: Array = new Array(); //移除的人

floorManNum[0] = floorManNum[1] = floorManNum[2] = floorManNum[3] = floorManNum[4] = 0; //当前都为0

Information.confirm.addEventListener("click", makeMyMan); //创建指定特征的人 监听点击事件

makeManAuto\_btn.addEventListener("click", autoMan);

makeManAuto\_btn.label = "自动生成人";

4个最关键的函数的的代码如下：

function liftMoveUp(): void { //实现电梯上升

isStill = false; //电梯正在上升

move(lift, lift.x, lift.y, lift.x, lift.y - 150, LIFTFLOORTIME);

setTimeout(function (): void {

liftCurrentFloor++; //电梯当前层+1

isStill = true; //电梯停

isCheck = false;

}, LIFTFLOORTIME);

}

//实现电梯上升，电梯下降的代码同理

//以下是电梯到达一楼层驱动的 判断电梯动作发生先后顺序 并播放相应动画 的函数

function liftArrive(mood: Number = 1): Boolean { //电梯到达的检测函数 执行电梯在这层的活动，先下后上，结束后 0表示强制检查

if (mood) {

if (isCheck == true) return false; //已经检查过了

isStill = false;

}

var ppp: Object = liftQueue;

var goout: Boolean = false,

goup: Boolean = false,

godown: Boolean = false;

while ((ppp = ppp.next) != undefined) {

if (ppp.to == liftCurrentFloor) goout = true; //有人要下电梯

else if (ppp.to > liftCurrentFloor) goup = true;

else godown = true;

}

switch (liftCurrentDir) { //状态的转变，尽量转换为2 静止状态

case 2:

break;

case 0:

if (goup == true) break; //乘客需要上楼

if (liftCurrentFloor == 4) {

liftCurrentDir = 2;

break; //变成静止状态

}

liftCurrentDir = 2; //变成静止状态

for (var i: int = liftCurrentFloor + 1; i < 5; i++) {

if (up[i] == 1 || down[i] == 1) liftCurrentDir = 0; //回到上升状态

}

break;

case 1:

if (godown == true) break; //乘客需要

if (liftCurrentFloor == 0) {

liftCurrentDir = 2;

break; //变成静止状态

}

liftCurrentDir = 2; //变成静止状态

for (var i: int = liftCurrentFloor - 1; i >= 0; i--) {

if (up[i] == 1 || down[i] == 1) liftCurrentDir = 1; //回到下降状态

}

break;

}

if (up[liftCurrentFloor] == 1 && liftCurrentDir != 1 || down[liftCurrentFloor] == 1 && liftCurrentDir != 0 || goout == true) { //有人按了这一楼的上升按钮或有人要下电梯

isStill = false; //电梯不需要在检测

lift.onOpening(); //开门

if (liftCurrentDir != 1) up[liftCurrentFloor] = 0; //熄灭按钮

if (liftCurrentDir != 0) down[liftCurrentFloor] = 0

setTimeout(function () {

var p: Object = getQueue(liftCurrentFloor);

var q: Object = p;

var pp: Object = liftQueue;

var qq: Object = pp;

while ((pp = pp.next) != undefined) { //查看谁要下电梯

if (pp.to == liftCurrentFloor) {

RemoveQueue(liftQueue, pp as Man, 1); //链表中删除这个人

(lift as DisplayObjectContainer).removeChild(pp as Man); //移除这个人的显示

addChild(pp as Man); //舞台上添加这个人的实例

pp.onWalking();

removeList.push(pp);

move(pp as MovieClip, LIFTX, 700 - liftCurrentFloor \* 150, MANLEAVEX, 700 - liftCurrentFloor \* 150, MANLEAVELIFTTIME);

}

}

setTimeout(removeMan, MANLEAVELIFTTIME);

if (goout == true) { //有人下电梯

var goin: Boolean = false; //没人上电梯

setTimeout(function () {

var r: Object = q;

while ((p = p.next) != undefined) { //查看要上楼的人走没走

if (p.to > p.from && liftCurrentDir != 1 || p.to < p.from && liftCurrentDir != 0) { //要上楼

goin = true;

RemoveQueue(q, p as Man); //从队列中删除

floorManNum[liftCurrentFloor]--;

if (p.next != undefined) delete p.next; //删除后续链接

EnQueue(liftQueue, p); //插入到电梯链表

p.onWalking();

move(p as MovieClip, p.x, p.y, LIFTX, p.y, GETINLIFTTIME);

p = r;

} else r = r.next;

}

if (goin == true) setTimeout(transMan, GETINLIFTTIME);

if (goin == true) setTimeout(function () { //关门

lift.onClosing();

setTimeout(function () { //关门之后进入上升状态

// liftMoveUp();

// isStill=false; //不再被检测

isStill = true; //继续检测

isCheck = true;

}, 1000);

}, GETINLIFTTIME + LIFTWAITTIME);

else setTimeout(function () { //关门,没人上楼

lift.onClosing();

setTimeout(function () { //关门之后进入上升状态

// liftMoveUp();

// isStill=false; //不再被检测

isStill = true; //继续检测

isCheck = true;

}, 1000);

}, LIFTWAITTIME);

}, MANLEAVELIFTTIME);

} else { //没人下楼直接执行

var goin: Boolean = false; //没人上电梯

var r: Object = q;

while ((p = p.next) != undefined) { //查看要上楼的人走没走

if (p.to > p.from && liftCurrentDir != 1 || p.to < p.from && liftCurrentDir != 0) { //要上楼

goin = true;

RemoveQueue(q, p as Man); //从队列中删除

floorManNum[liftCurrentFloor]--;

if (p.next != undefined) delete p.next; //删除后续链接

EnQueue(liftQueue, p); //插入到电梯链表

p.onWalking();

move(p as MovieClip, p.x, p.y, LIFTX, p.y, GETINLIFTTIME);

p = r;

} else r = r.next; //比p慢一个节点

}

setTimeout(transMan, GETINLIFTTIME); //删除舞台上的显示对象并插入电梯中

if (goin == true) setTimeout(function () { //关门

lift.onClosing();

setTimeout(function () { //关门之后进入上升状态

// liftMoveUp();

// isStill=false; //不再被检测

isStill = true; //继续检测

isCheck = true;

}, 1000);

}, GETINLIFTTIME + LIFTWAITTIME);

else setTimeout(function () { //关门

lift.onClosing();

setTimeout(function () { //关门之后进入上升状态

// liftMoveUp();

// isStill=false; //不再被检测

isCheck = true;

isStill = true; //继续检测

}, 1000);

}, LIFTWAITTIME);

}

}, 1000); //开门之后干什么

return false; //处理了事件

} else {

isStill = true;

isCheck = true;

return true; //没有动作了，所有事件处理完

}

} //进入函数之后如果ischeck=true返回false，否则isstill=false,并检测

//检测完后isStill=isCheck=true，如果有动作，返回false，否则true

//以0进入函数为跳过第一条

//以下是时间驱动 定时检测电梯状态的函数并判断电梯向哪里运行

function liftClock(): void {

var sta: Number = 2; //表示电梯应该进行的动作 0上升1下降2不动

if (isStill == false) return; //电梯还在运动或正在检测

var mp: Object = liftQueue;

switch (liftCurrentDir) {

case 2: //停止状态

if (liftArrive(0) == false) {

isStill = false; //强制检查 如果有动作改变isfalse等待动作结束

break;

}

if (liftCurrentFloor != 1) { //没有停在1层

if (startCounting == false) { //如果没有开始计时

startCounting = true; //开始计时

liftcount = setTimeout(function (): void {

back1 = true;

}, SLEEPTIME); //sleeptime之后

}

}

while ((mp = mp.next) != undefined) {

if (mp.to > liftCurrentFloor) sta = 0; //上升

}

for (var i: int = liftCurrentFloor + 1; i < 5; i++) {

if (up[i] == 1 || down[i] == 1) sta = 0; //上升

}

if (sta == 0) {

liftMoveUp();

liftCurrentDir = 0;

} else { //先考虑上升再考虑下降

mp = liftQueue;

while ((mp = mp.next) != undefined) {

if (mp.to < liftCurrentFloor) sta = 1; //下降

}

for (var i: int = liftCurrentFloor - 1; i >= 0; i--) {

if (up[i] == 1 || down[i] == 1) sta = 1; //下降

}

if (sta == 1) {

liftCurrentDir = 1; //电梯下降

liftMoveDown();

} else if (back1 == true) { //回到1楼休眠

if (liftCurrentFloor == 0) {

liftMoveUp();

startCounting = false;

back1 = false;

} else {

if (liftCurrentFloor == 1) {

startCounting = false;

back1 = false;

} else liftMoveDown();

}

}

}

break;

case 1: //下降

becomeMoving();

if (liftArrive(0) == false) {

isStill = false; //强制检查 如果有动作改变isfalse等待动作结束

break; //有动作返回

}

while ((mp = mp.next) != undefined) {

if (mp.to < liftCurrentFloor) sta = 1; //下降

}

for (var i: int = liftCurrentFloor - 1; i >= 0; i--) {

if (up[i] == 1 || down[i] == 1) sta = 1; //下降

}

if (sta == 1) liftMoveDown();

else { //先考虑

mp = liftQueue;

while ((mp = mp.next) != undefined) {

if (mp.to > liftCurrentFloor) sta = 0; //上升

}

for (var i: int = liftCurrentFloor + 1; i < 5; i++) {

if (up[i] == 1 || down[i] == 1) sta = 0; //上升

}

if (sta == 0) {

liftCurrentDir = 0; //电梯上升

liftMoveUp();

} else { //既没有上升请求，也没有下降请求

liftCurrentDir = 2; //不动

}

}

break;

case 0: //上升状态

becomeMoving();

if (liftArrive(0) == false) {

isStill = false; //强制检查 如果有动作改变isfalse等待动作结束

break; //有动作返回

}

while ((mp = mp.next) != undefined) {

if (mp.to > liftCurrentFloor) sta = 0; //上升

}

for (var i: int = liftCurrentFloor + 1; i < 5; i++) {

if (up[i] == 1 || down[i] == 1) sta = 0; //上升

}

if (sta == 0) liftMoveUp();

else { //先考虑上升再考虑下降

mp = liftQueue;

while ((mp = mp.next) != undefined) {

if (mp.to < liftCurrentFloor) sta = 1; //下降

}

for (var i: int = liftCurrentFloor - 1; i >= 0; i--) {

if (up[i] == 1 || down[i] == 1) sta = 1; //下降

}

if (sta == 1) {

liftCurrentDir = 1; //电梯下降

liftMoveDown();

} else { //既没有上升请求，也没有下降请求

liftCurrentDir = 2; //不动

}

}

break;

}

}

//最后一个是新人到来播放相应动画的函数

function newManArrive(newman: Man): void { //立即发生的事件：新人来了

addChild(newman); //添加显示对象newman

newman.addEventListener("click", showClickInformation);

newman.x = MANBORNX;

floorManNum[newman.from]++; //当前层人数加一

switch (newman.from) { //人在那一层上电梯

case 0:

EnQueue(B1Queue, newman); //插入该层队列

newman.y = 700; //显示在该楼层位置

break;

case 1:

EnQueue(F1Queue, newman);

newman.y = 550; //显示在该楼层位置

break;

case 2:

EnQueue(F2Queue, newman);

newman.y = 400; //显示在该楼层位置

break;

case 3:

EnQueue(F3Queue, newman);

newman.y = 250; //显示在该楼层位置

break;

case 4:

EnQueue(F4Queue, newman);

newman.y = 100; //显示在该楼层位置

break;

}

newman.onWalking(); //正在行走的动画

showInformation(newman);

move(newman, newman.x, newman.y, floorManNum[newman.from] \* 100 + QUEUELINE, newman.y, WALKTOLIFTTIME);

setTimeout(function (): void {

if (maninit == true) manOnChoosing.gotoAndStop(2); //保证当前只选中一个人

else maninit = true;

manOnChoosing = newman;

newman.onChoosing();

}, WALKTOLIFTTIME); //经过WALKTOLIFTTIME时间后走到电梯门口并按下按钮

if (newman.to > newman.from) { //上楼

up[newman.from] = 1;

} else { //下楼

down[newman.from] = 1;

}

}

其余代码辅助这几个函数，不在此列出

总的层次关系是：人出现 -> 人按下某个电梯按钮 -> 电梯在某个时钟边缘检测到，并判断相应动作 -> 电梯楼层改变，重新判断相应动作，以免在进行上一个动作的过程中发生新的事件

因为人可能在任意时刻出现，电梯对应该进行的行为的检测应一直重复，比如在三楼半时收到四楼上楼的指令，必须改变之前的行为，变为停到四楼。

由于事件有发生的先后顺序，且必须，在事件发生的前一秒钟检查是否发生。（比如电梯关门的时候不能马上将上/下楼作为下一个行为，必须检测是否又有人来到该楼层按下按钮，即使在关门的最后一秒也不能断定关门后即可上下楼，所以这个事件就必须在到来的瞬间确定）这就需要不停的检测（比如开门动作完成后，检查是否有人要上楼，而关门时间的确定必须在上楼动作完成后，再次确认上楼这段时间内没有新人到来，才可以确定关门），代码有一些重复的部分（第二次检查），不够精简。应该可以更简短明白。

1. **调试分析**
2. 手动输入了一些数据，再输入不出错，且在合理范围内，程序均能处理。
3. 在写链表时，发现用p.next != NULL 无法作为结束条件，进行思考和尝试后发现，结束条件可用p.next == undefined。结尾置空时使用delete p.next即可。
4. 使用链表遍历应该从电梯上删除的人，将他们移动到楼的位置，然后注册一个删除函数使他们一定时间后从显示列表中移除，但是在这段时间后链表的指针可能已经指向别的数据，造成程序崩溃。最后将删除的人入栈，在一段时间后删除这个栈中的人即可。
5. 由于parent属于DisplayObject类，使用上一级的变量时应用( parent as MovieClip ).a，

同理Object和Man的转换。

1. 做时钟时，发现元件的旋转点不是影片剪辑的旋转点，而是该元件的注册点。
2. 为了进入电梯的人能跟电梯一起运动，把人添加到电梯的显示列表中。此时如果遍历电梯的链表，程序运行时会崩溃报错。应将人作为DisplayObject，判断其不属于电梯这个DisplayObjectContainer，但又在舞台显示列表中，再对其操作。
3. **用户手册**
4. 本程序需要在adobe flash player中运行（html文件，打开后自动进入浏览器加载flash player，如果你没有卸载这个东西）。也可能可以用暴风影音等视频播放软件打开（swf文件）。源文件用Adobe Animate打开（fla），也可以用Adobe Flash打开，需要cs3以上的版本（需要支持Action Script 3）。源代码我也放在了全部源代码.doc中。
5. 请先左键点击右上角“开始模拟”，看到下方时钟开始计时，此时可点击“自动生成人”自动生成人，或者点击“随机生成一个人”手动生成人。可以改下面的文本来创建一个指定的人（数据要合理）。写好数据后点击“生成指定的人”即可。最下方可以切换时钟到系统时钟或模拟时钟。当有人排队时，可以点击某一个人在右侧查看该人信息，并且此人变红，表示该人被选定，显示的信息为这个人的信息。绿色的为时间条，减少到0后此人离开，表示剩余等待时间。两个三角表示按钮，绿色为亮，表示有人按了这个按钮。
6. **测试结果**

程序运行了6分钟，没有什么问题。

