注意:

1. C++代码中，要使用数学函数，必须引用头文件#include<cmath>;
2. 未加特殊说明时，以下代码默认均为Java Script实现;
3. C++所需变量定义:

const double EP = 1E-10

* 数学

1. JS中，向下取整、向上取整和四舍五入函数

Math.ceil(); //向上取整xintiao

Math.floor(); //向下取整

Math.round(); //四舍五入

* 物理

1. 匀变速运动

* 需要获取的值

【运动开始-->某个时间点经历的时间g\_time】

【初始速度v0】

【加速度g\_accelSpeed】

【两次渲染之间的时间间隔delta】

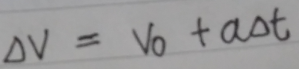
* 需要的数学知识——【微积分】





//微分





* 代码:【JS实现】

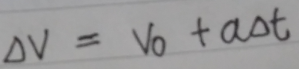
var g\_time= 0;//运动时长

var v0= 100;//初速度

var g\_accelSpeed= 100;

updateData:function(delta){

g\_time+= delta;

【原理：】

var vt= v0+ g\_time\* g\_accelSpeed;//时刻速度

【原理：】

var step= vt\* delta;//每一帧要移动的长度

this.x+= step;

this.y+= step;

}

1. 有方向的匀变速运动

基本上，和N0.1一致，需要改动两个地方:

* 需要获取的值

运动偏转方向g\_angle

* 更新位置: 【JS实现】

this.x+= setp\* Math.sin(g\_angle\*(Math.PI/ 180));

this.y+= step\* Math.cos(g\_angle\*(Math.PI/ 180));

* 几何

1. 点与点（可近似的看成一条线段上的两个端点）【JS实现】

* 判断:两个点是否相等

1. 【JS实现】

function pointEqualToPoint(point1, point2) {

return (point1.x == point2.x) && (point1.y == point2.y);

}

1. 【C++实现】

bool equal\_point(POINT p1,POINT p2)

{

return ( (abs(p1.x-p2.x)<EP)&&(abs(p1.y-p2.y)<EP) );

}

* 计算:关于原点的对称点

function pNeg(point){

return cc.p(-point.x, -point.y);

}

* 计算:向量加法

function pAdd(v1, v2) {

return cc.p(v1.x + v2.x, v1.y + v2.y);

}

* 计算:向量减法

function pSub(v1, v2) {

return cc.p(v1.x - v2.x, v1.y - v2.y);

}

* 计算:数乘

function pMult(point, floatVar) {

return cc.p(point.x \* floatVar, point.y \* floatVar);

}

* 计算:两向量的中心点

function pMidpoint(v1, v2) {

return cc.pMult(cc.pAdd(v1, v2), 0.5);

}

* 计算:点乘

function pDot(v1, v2) {

return v1.x \* v2.x + v1.y \* v2.y;

}

* 计算:叉乘

function pCross(v1, v2) {

return v1.x \* v2.y - v1.y \* v2.x;

}

* 计算:三个点的叉积

function crossProduct(p1, p2, p3){

return (p1.x\* p3.y+ p2.x\* p1.y+ p3.x\* p2.y- p1.x\* p2.y- p2.x\* p3.y- p3.x\* p1.y);

}

* 计算:模的平方

function pLengthSQ(v) {

return cc.pDot(v, v);

}

* 计算:两点之间距离的平方

function pDistanceSQ(point1, point2){

return cc.pLengthSQ(cc.pSub(point1,point2));

}

* 计算:模

function pLength(v) {

return Math.sqrt(cc.pLengthSQ(v));

}

* 计算:两点之间的距离

1. 【JS实现】

function pDistance(v1, v2) {

return cc.pLength(cc.pSub(v1, v2));

}

1. 【C++实现】

double dist(POINT p1,POINT p2)

{

return( sqrt( (p1.x-p2.x)\*(p1.x-p2.x)+(p1.y-p2.y)\*(p1.y-p2.y) ) );

}

* 计算:单位向量

function pNormalize(v) {

return cc.pMult(v, 1.0 / cc.pLength(v));

}

* 计算:两向量之间的角弧度

//允许误差

POINT\_EPSILON = parseFloat('1.192092896e-07F');

function pAngle(a, b) {

var angle = Math.acos(cc.pDot(cc.pNormalize(a), cc.pNormalize(b)));

if (Math.abs(angle) < POINT\_EPSILON) return 0.0;

return angle;

}

* 计算:两个点组成的线段的角度:

//范围【0~360】

function getAngle(startPos, endPos){

var vector= cc.pSub(endPos, startPos);

var length= cc.pLength(vector);

var angle= Math.floor(Math.acos(vector.x/ length)\* 180/ Math.PI);

angle+= (vector.y< 0)? 180:0;

return angle;

}

* 扩展:八方向算法
  + 原理:

**指针或者触摸点的位置，在目标的八方向中的哪个方向 == (指针或者触摸点的位置)和（目标所在位置）的连线的角度。**

**所在的方向= Math.floor(角度/45);**

1. 点与线段【JS实现】
2. 线段【JS实现】

* 计算: 单位时间内,沿着某个线段移动的位移
* 原理:

**x方向上的位移= speed\* cos(radian);**

**y方向上的位移= speed\* sin(radian);**

* 首先,求出线段的弧度【既是求出两点之间的角度】;
* 示例:

/\*

func:

单位时间内,沿着某个角度移动的位移

principle:

x方向上的位移= speed\* cos(radian);

y方向上的位移= speed\* sin(radian);

@delta:单位时间

@speed:速度（每秒钟移动的像素个数）

@angle:角度

@return 单位时间内的位移。

\*/

function MoveAlongLine(delta, speed, angle){

var pos= cc.p(0, 0);

var radian= angleToRadian(angle);

pos.x= delta\* speed\* Math.cos(radian);

pos.y= delta\* speed\* Math.sin(radian);

return pos;

}

* 扩展:自动寻路算法
* 原理:

**目标在【（目标所在的位置）和(指针或触摸点的位置)组成的线段】上移动。**

1. 线段和线段【JS实现】

* 判断:是否相交

function IsLineIntersectLine(begPos, endPos, p1, p2){

if(crossProduct(begPos, endPos, p1) \* crossProduct(begPos, endPos, p2)> 0)

return false;

if(crossProduct(p1, p2, begPos) \* crossProduct(p1, p2, endPos)> 0)

return false;

return true;

}

1. 矩形【JS实现】

* 获取:矩形右上角的X坐标【即矩形X轴最右边的值】

function rectGetMaxX(rect) {

return (rect.x + rect.width);

}

* 获取:矩形中心点的X坐标【即矩形X轴的中点】

function rectGetMidX(rect) {

return (rect.x + rect.width / 2.0);

}

* 获取:矩形左下角的X坐标【即矩形X轴最左边的值】

function rectGetMinX(rect) {

return rect.x;

}

* 获取:矩形右上角的Y坐标【即矩形Y轴最上面的值】

function rectGetMaxY(rect) {

return(rect.y + rect.height);

}

* 获取:矩形中心点的Y坐标【即矩形Y轴中点】

function rectGetMidY(rect) {

return rect.y + rect.height / 2.0;

}

* 获取:矩形左下角的Y坐标【即矩形Y轴下方向最小值】

function rectGetMinY(rect) {

return rect.y;

}

1. 点与矩形【JS实现】

* 判断:点是否在矩形中

function rectContainsPoint(rect, point) {

return (point.x >= rectGetMinX(rect) && point.x <= rectGetMaxX(rect) &&

point.y >= rectGetMinY(rect) && point.y <= rectGetMaxY(rect)) ;

}

1. 线段与矩形

* 判断:线段是否与矩形相交

1. 两步曲之一:线段是否和矩形的四条边相交
2. 两步曲之二:线段是否和线段相交

/\*

参数说明:

begPos——线段的起点

endPos——线段的终点

rect——矩形【锚点为(0,0)】

\*/

function IsLineIntersectRect(begPos, endPos, rect){

var points=[//顺时针【矩形的四条边】

cc.p(rect.x, rect.y),

cc.p(rect.x, rect.y+ rect.height),

cc.p(rect.x+ rect.width, rect.y+ rect.height),

cc.p(rect.x+ rect.width, rect.y)

];

//线段和矩形的四条边组成的线段比较

for(var i=0;i< points.length; ++i){

var p1= points[i];

var p2= points[(i+1)%points.length];

if(IsLineIntersectLine(begPos, endPos, p1, p2)){//相交

return true;

}

}

return false;

}

1. 矩形与矩形

* 获取:能包含两个矩形的最小矩形

function rectUnion(rectA, rectB) {

var rect = cc.rect(0, 0, 0, 0);

rect.x = Math.min(rectA.x, rectB.x);

rect.y = Math.min(rectA.y, rectB.y);

rect.width = Math.max(rectA.x + rectA.width, rectB.x + rectB.width) - rect.x;

rect.height = Math.max(rectA.y + rectA.height, rectB.y + rectB.height) - rect.y;

return rect;

}

* 获取:两个矩形的重叠部分

function rectIntersection(rectA, rectB) {

var intersection = cc.rect(

Math.max(cc.rectGetMinX(rectA), cc.rectGetMinX(rectB)),

Math.max(cc.rectGetMinY(rectA), cc.rectGetMinY(rectB)),

0, 0);

intersection.width = Math.min(cc.rectGetMaxX(rectA), cc.rectGetMaxX(rectB)) - cc.rectGetMinX(intersection);

intersection.height = Math.min(cc.rectGetMaxY(rectA), cc.rectGetMaxY(rectB)) - cc.rectGetMinY(intersection);

return intersection;

}

* 判断:相等

function rectEqualToRect(rect1, rect2) {

return (rect1.x == rect2.x) && (rect1.y == rect2.y) && (rect1.width == rect2.width) && (rect1.height == rect2.height);

}

* 判断:一个矩形是否在另外一个矩形内

function rectIntersectsRect(ra, rb) {

var maxax = ra.x + ra.width,

maxay = ra.y + ra.height,

maxbx = rb.x + rb.width,

maxby = rb.y + rb.height;

return !(maxax < rb.x || maxbx < ra.x || maxay < rb.y || maxby < ra.y);

}

* 判断:是否有交集

function rectOverlapsRect(rectA, rectB) {

return !((rectA.x + rectA.width < rectB.x) ||

(rectB.x + rectB.width < rectA.x) ||

(rectA.y + rectA.height < rectB.y) ||

(rectB.y + rectB.height < rectA.y));

}

1. 角度(angle)和弧度(radian)之间的转化:

* 角度-->弧度

/\*

func:

角度转换为弧度

@angle:角度

@return:弧度

\*/

function angleToRadian(angle){

return (angle\* Math.PI/ 180);

}

* 弧度-->角度

/\*

func:

弧度转换为角度

@angle:弧度

@return:角度

\*/

function radianToAngle(radian){

return (radian\* 180/ Math.PI);

}

* 算法

1. 校验信息:

* 代码实现:

/\*

func:

简单校验手机号是否合法

return:不合法时，返回true

\*/

function verifyPhone(phone){

return (!/^(13[\d]{9}|15[\d]{9}|17[\d]{9}|18[\d]{9})$/.test(phone));

}

/\*

func:

校验姓名、地址是否合法

return:不合法时，返回true

\*/

function verifyNameOrAddress(msg){

return (!msg.trim());

}

/\*

func:

校验身份证号

id:身份证号

\*/

function verifyIDCard(id){

return (!/^[1-9]\d{5}[1-9]\d{3}((0\d)|(1[0-2]))(([0|1|2]\d)|3[0-1])\d{3}[0-9|X|x]$/.test(id));

}

/\*

func:

校验某个字符是否为英文

ch:要校验的字符

\*/

function judgeEnglish(ch)

{

var Regx = /^[A-Za-z]\*$/;

return (ch.length== 1)&&(Regx.test(ch));

}

/\*

func:

校验某个字符是否为中文

ch:要校验的字符

\*/

function judgeChinese(ch)

{

return (ch.length== 1)&&((ch.charCodeAt()>= 0x4e00)&&(ch.charCodeAt()<= 0x9fa5));

}

/\*

func:

校验某个字符是否为数字

ch:要校验的字符

\*/

function judgeChinese(ch)

{

var Regx = /^[0-9]\*$/;

return (ch.length== 1)&&(Regx.test(ch));

}

/\*

func:

校验某个字符串是否仅有字母和数字组成

str:要校验的字符

\*/

function checkNum(str {

var Regx = /^[A-Za-z0-9]\*$/;

return Regx.test(str);

}

/\*

func:

校验车架号是否正确

str:要校验的字符

\*/

function judgeFrameNumber(frameNumber)

{

return (frameNumber.length== 17)&&(checkNum(frameNumber));

}

/\*

func:

固定格式的日期比较大小(日期格式为:年-月-日)

bigDate:比较大的日期

smallDate:比较小的日期

\*/

function judgeDate(bigDate, smallDate)

{

for(var i=0; i<3; i++)

{

if(i== 2){

var bigNum= parseInt(bigDate);

var smallNum= parseInt(smallDate);

if(bigNum> smallNum){

return true;

}

}else{

var bigNum= parseInt(bigDate.substring(0, bigDate.indexOf("-")));

var smallNum= parseInt(smallDate.substring(0, smallDate.indexOf("-")));

bigDate= bigDate.substring(bigDate.indexOf("-")+ 1, bigDate.length);

smallDate= smallDate.substring(smallDate.indexOf("-")+ 1, smallDate.length);

if(bigNum> smallNum){

return true;

}

}

}

return false;

}

/\*

func:

车牌号的粗略校验

一个字符为中文或者英文，第二个字符一定为英文 后五位具有数字或者字母组成

\*/

function judgeCarNumber(carNumber)

{

var firstChar= carNumber.substring(0, 1);

var secondChar= carNumber.substring(1, 2);

var lastStr= carNumber.substring(2, carNumber.length);

return (carNumber.length== 7)&&(judgeChinese(firstChar)||judgeEnglish(firstChar))&&(judgeEnglish(secondChar))&&(checkNum(lastStr));

}

* 提交按钮的单击事件:

var name= $("#name").val();

var address= $("#address").val();

var phone= $("#phone2").val();

if(verifyNameOrAddress(name)){

$("#tip1").css("display", "block");

return;

}

if(verifyPhone(phone)){

var tip= $("#tip2");

tip.css("display", "block");

return;

}

if(verifyNameOrAddress(address)){

$("#tip3").css("display", "block");

return;

}

**/\*调用后台的action，传值\*/**

* 在action的回调函数中，添加验证是否传值成功:

var tip= $("#tip");

if(msg== "repeat"){

tip.css("display", "block");

tip.html("手机号重复！");

return;

}else if(msg== "error"){

tip.css("display", "block");

tip.html("网络异常,稍后重试！");

return;

}

1. 屏幕旋转提示-H5监听:

var supportsOrientationChange = "onorientationchange" in window,

orientationEvent = supportsOrientationChange ? "orientationchange" : "resize";

// 监听事件

window.addEventListener(orientationEvent, function() {

if(window.orientation != 0){

alert("童鞋们,请竖屏玩耍！");

}

}, false);

1. 围成一个圆【JS实现】

* 【根据圆心坐标、圆的半径、圆上的物品个数】摆放指定的物品。

/\*

功能:几个元素围成一个圆

参数:

posCenter—圆心坐标

radius—圆的半径

num—元素个数

\*/

function GetPosition(posCenter, radius, num){

//将圆平分成num份

var angle= 360/num;

//创建并初始化一个二维数组

var array=new Array(num);

//遍历

for(var i=0; i< array.length; ++i){

//动态创建二维数组

array[i]= new Array(2);

//初始化

array[i][0]=Math.floor(posCenter.x+ Math.cos(i\*angle\* Math.PI/ 180)\* radius);

array[i][1]=Math.floor(posCenter.y+ Math.sin(i\*angle\* Math.PI/ 180)\* radius);

}

return array;

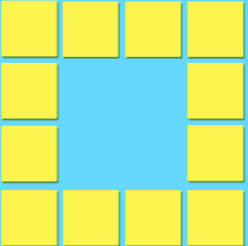
}

* 调用方式:

var center={x:100, y:100};

GetPosition(center, 100, 2);

1. 类似下图方式，摆放元素:【JS实现】



* 格式变量定义:

var g\_smileNums= 7;//笑脸图标数

var g\_cols= 4;//行数

var g\_rows= 4;//列数

var g\_seqArray= [];//奖品摆放位置数组

* 创建奖品摆放位置

//create the sequence of the award

function createSequence(nums){

//fill the array,not winning in the front,winning in the back

var array= [];

//not awarding

for(var i=0;i<g\_smileNums;++i){

array.push(0);

}

for(i=1;i<=nums-g\_smileNums;++i){

array.push(i);

}

//disrupted the array

array.sort(function(){ return Math.random()<0.5?-1:1;});

return array;

}

* 根据索引,获取该项在二维数组中的行列号

function getItemPosition(index){

//x-col y-row

var pos= cc.p(0,0);

//regional judgement

if(0<=index&&index<g\_cols){//the top edge

pos.x= index;

pos.y= 0;

}else if(index<(g\_cols+g\_rows-2)){//the right-middle edge

pos.x= g\_cols-1;

pos.y= index-g\_cols+1;

}else if(index<(g\_cols\*2+g\_rows-2)){//the bottom edge

pos.x= (2\*g\_cols+g\_rows-2)-index-1;

pos.y= g\_rows-1;

}else{//the left-middle edge

var numbers= 2\*g\_cols+ 2\*(g\_rows-2);

pos.x= 0;

pos.y= numbers-index;

}

return pos;

}

* 创建转盘上摆放的精灵

//创建精灵

function initItemsOnTable(){

//calculate the number of the items

var numbers= 2\*g\_cols+ 2\*(g\_rows-2);

g\_seqArray= createSequence(numbers);

for(var i=0;i<numbers;++i){

//position

var pos= getItemPosition(i);

var index= g\_seqArray[i];

if(index==0){

index=8;

}

var item= new Sprite("#award"+index+".png");

var size= item.getContentSize();

//center

var offset= cc.p(cc.winSize.width/2-g\_cols/2\*(size.width+20),cc.winSize.height\*0.6);

item.setPosition(cc.pAdd(cc.p(pos.x\*(size.width+20),-pos.y\*(size.height+20)),offset));

this.addChild(item,2,i+20);

}

}

1. 获取字符串链接中的某个值:【JS实现】

"http://219.234.5.130/game.html?openid=13522490962"

* 方法一:正则分析法

function getQueryString(name) {

var reg = new RegExp("(^|&)" + name + "=([^&]\*)(&|$)", "i");

var r = window.location.search.substr(1).match(reg);

if (r != null) return unescape(r[2]); return null;

}

举例:

var g\_openID= getQueryString(openid);

* 方法二:

<Script language="javascript">

function GetRequest() {

var url = location.search; //获取url中"?"符后的字串

var theRequest = new Object();

if (url.indexOf("?") != -1) {

var str = url.substr(1);

strs = str.split("&");

for(var i = 0; i < strs.length; i ++) {

theRequest[strs[i].split("=")[0]]=unescape(strs[i].split("=")[1]);

}

}

return theRequest;

}

</Script>

调用方法：

<Script language="javascript">

var Request = new Object();

Request = GetRequest();

var 参数1,参数2,参数3,参数N;

参数1 = Request['参数1'];

参数2 = Request['参数2'];

参数3 = Request['参数3'];

参数N = Request['参数N'];

</Script>

1. 查找某特定字符在字符串中的首次出现的位置:

* 定义和用法

indexOf() 方法:

可返回某个指定的字符串值在字符串中首次出现的位置。

* 语法

stringObject.indexOf(searchvalue,fromindex)

* 参数 描述

searchvalue 必需。规定需检索的字符串值。

fromindex 可选的整数参数。规定在字符串中开始检索的位置。它的合法取值是 0 到 stringObject.length - 1。如省略该参数，则将从字符串的首字符开始检索。

* 说明

该方法将从头到尾地检索字符串 stringObject，看它是否含有子串 searchvalue。开始检索的位置在字符串的 fromindex 处或字符串的开头（没有指定 fromindex 时）。如果找到一个 searchvalue，则返回 searchvalue 的第一次出现的位置。stringObject 中的字符位置是从 0 开始的。

* 提示和注释

注释：indexOf() 方法对大小写敏感！

注释：如果要检索的字符串值没有出现，则该方法返回 -1。

1. 截取字符串

* 定义和用法

substr() 方法:

可在字符串中抽取从 start 下标开始的指定数目的字符。

* 语法

stringObject.substr(start,length)

* 参数 描述

start 必需。要抽取的子串的起始下标。必须是数值。如果是负数，那么该参数声明从字符串的尾部开始算起的位置。也就是说，-1 指字符串中最后一个字符，-2 指倒数第二个字符，以此类推。

length 可选。子串中的字符数。必须是数值。如果省略了该参数，那么返回从 stringObject 的开始位置到结尾的字串。

* 返回值

一个新的字符串，包含从 stringObject 的 start（包括 start 所指的字符） 处开始的 length 个字符。如果没有指定 length，那么返回的字符串包含从 start 到 stringObject 的结尾的字符。

* 提示和注释

注释：substr() 的参数指定的是子串的开始位置和长度，因此它可以替代 substring() 和 slice() 来使用。

* 实例:
  + 截取特定格式的字符串: 'abc&123'

g\_code= temp.substr(0, temp.indexOf('&'));

g\_password= temp.substr(temp.indexOf('&')+ 1, temp.length);

1. 打乱数组

* JS实现

var seq= [1, 2, 3];

seq.sort(function(){return Math.random()<0.5?-1:1;});

* C++实现

1. 如果无特殊需要，可以直接使用系统提供的函数:

#include<algorithm>

int a[10] = {1,2,3,4,5,6,7,8,9,10};

int\* a\_begin = a;

int\* a\_end = a + sizeof(a)/sizeof(int);

random\_shuffle(a\_begin,a\_end);

1. 自定义随机交换两个值:

int a[10] = {1,2,3,4,5,6,7,8,9,10};

for(int i=0;i<10;i++) swap(a[i],a[rand()%10]);

* 总结:

算法不严密，搞不好的话根本就排出来和原来一样，而且全部打乱的几率很小。

1. 高效的数值交换方法

C++/JS中，不借助临时变量，仅仅使用【位异或】来交换两个变量的值:

void Swap(int\* a, int\* b)

{

\*a ^= \*b;

\*b ^= \*a;

\*a ^= \*b;

}

//调用

int a= 10,b=1;

Swap(&a, &b);

* 总结:

虽然JS/C++都可以使用【位异或】运算，来达到上述功能。但是因为JS中，没有指针(\*)和取址符(&)，所以JS中不能自定义上述函数。

1. 一维、二维数组的转换

注意:数组的下标都是从0开始

* 一维-->二维

列号= 下标/ 列数;

行号= 下标% 列数;

注意:不同的语言，在进行除法运算时，机制不同，因此要注意这一点。

* 二维-->一维

下标= 行号\* 列数+ 列号;

1. 在二维数组最后添加一条记录:

var arr= new Array(2);//列数为2

arr[0]= target;//目标

arr[1]= true;//默认,可以被切中

self.child.push(arr);

1. 将字符串转换为一维数组:

var str= "1111";

var arr= str.split("");

1. 文本自动换行【JS实现】

* 环境:

var self= this;

var label= new cc.LabelTTF("10000000000000000000000000", "微软雅黑", 30);

label.attr({

x:cc.winSize.width\* 0.5,

y:cc.winSize.height\* 0.5,

color:cc.color(255, 0, 0)

});

self.addChild(label, 1, 1);

* 方法一:调用其私有接口

label.\_setBoundingWidth(200);

* 方法二: 自我实现
  + 代码实现

/\*

func:

长字符串——自动换行

principle；

获取字体对象的宽度。超出的部分添加换行符"\r\n"。

@limitWidth:限制文本的显示区域宽度

@target:字体对象

\*/

function AutoLinefeed(limitWidth,target){

var str= target.getString();//字体对象中的文本

var width= target.width;//获取文本的总宽度

var singleWidth= Math.ceil(width/ str.length);//单个字符的宽度

var numPreRow= Math.floor(limitWidth/ singleWidth);//每行中的字符数目

var rows= Math.ceil(width/(singleWidth\* numPreRow));//需要几行，才可以将所有字符， 按照要求显示完全

var temp= str;

str= "";

for(var i=0;i< rows; ++i){

if(i== (rows- 1)){

str+= temp.substr((i\* numPreRow), (temp.length- i\*numPreRow));

}else{

str+= temp.substr((i\* numPreRow), numPreRow)+ "\r\n";

}

}

target.setString(str);

}

* + 接口调用:

**AutoLinefeed(200, label);**

1. 动画播放算法:

* 代码实现:

/\*

func:

纹理种类

\*/

if(typeof g\_picType== "undefined"){

var g\_picType= {

plist:0,

single:1

};

}

/\*

func:

两种纹理图片，创建动画。

type:纹理图片的种类

firstPath:起始动画的纹理路径

id:起始动画的ID

num:动画由多少帧组成

delay:延时,即每两帧动画之间的播放时间间隔

bRestoreOriginal:播放完成之后，是否回到首帧

times:动画播放次数

target:被执行的目标

\*/

function CreateAnimate(type, firstPath, id, num, delay, bRestoreOriginal, times, target){

var path= firstPath.substring(0, firstPath.length- 5);

//func:单个图片——方法一

var animation= new cc.Animation();

if(type== g\_picType.single){

for(var i= id; i<= num; ++i){

animation.addSpriteFrameWithFile(path+ i+ ".png");

}

}else{

for(i= id; i<= num; ++i){

var frame= cc.spriteFrameCache.getSpriteFrame(path+ i+ ".png");

animation.addSpriteFrame(frame);

}

}

animation.setDelayPerUnit(delay);

animation.setRestoreOriginalFrame(bRestoreOriginal);

var animate= cc.animate(animation);

if(times== 0){

target.runAction(animate.repeatForever());

}else{

target.runAction(animate.repeat(times));

}

}

* 接口调用:

**CreateAnimate(g\_picType.single, res.car1\_png, 1, 3, 1/6, true, 0, car);**

* 说明:

因为菜单已经被封装，所以菜单项做动画时，不能像精灵一样播放动画，会报错误，需要自我实现:

其中:

self.m\_btn\_start是要播放的菜单项

self.m\_curID是要显示的动画帧的ID

要播放的动画帧为2

//利用定时器，播放动画帧

self.schedule(self.changeMenuSprite, 0.16);

changeMenuSprite:function(delta)

{

var self= this;

self.m\_curID++;

if(self.m\_curID > 2){

self.m\_curID= 1;

}

self.m\_btn\_start.setNormalSpriteFrame("#ksyx\_" + self.m\_curID + ".png");

},

1. 切换图片:

* 精灵(Sprite):
  + 单个文件:

setTexture(cc.textureCache.addImage("res/stars.png"));

* + 序列帧:

setDisplayFrame(cc.spriteFrameCache.getSpriteFrame("runner1.png"));

* 菜单(Menu):
  + 单个文件:

setNormalImage(cc.Sprite.create(res.a\_btnOn));

* + 序列帧:

setNormalSpriteFrame("#ksyx\_" + self.m\_curID + ".png");

1. 单方向移动算法:

* 移动的同时,不透明度逐渐降低
  + 代码实现:

/\*

func:

单方向移动——移动的同时,不透明度逐渐降低

@srcPos:起始位置

@desPos:终点位置

@time:单方向移动时间

@times:动作执行次数

@target:动作执行目标

\*/

function SingleFade(srcPos, desPos, time, times, target){

var move= cc.moveTo(time, desPos);

var fadeOut= cc.fadeOut(time);

var spawn= cc.spawn(move, fadeOut);

var seq= cc.sequence(spawn, cc.callFunc(function(pSender){

pSender.setPosition(srcPos);

pSender.setOpacity(255);

}));

if(times== 0){

target.runAction(seq.repeatForever());

}else{

target.runAction(seq.repeat(times));

}

}

* + 接口调用:

**SingleFade(****cc.p(cc.winSize\* 0.5, 0), cc.p(cc.winSize.height\* 0.5, cc.winSize.height\* 0.5), 1, 0, target);**

* 单方向移动算法-移动的同时，缩小
  + 代码实现:

/\*

func:

单方向移动算法-移动的同时，缩小

@srcPos:起始位置

@desPos:终点位置

@srcScale:起始缩放系数

@desScale:终止缩放系数

@time:单方向运动时间

@times:运动次数

@target:目标

\*/

function SingleScale(srcPos, desPos, srcScale, desScale, time, times, target){

var move= cc.moveTo(time, desPos);

var scale= cc.scaleTo(time, desScale);

var spawn= cc.spawn(move, scale);

var seq= cc.sequence(spawn, cc.callFunc(function(pSender){

target.setPosition(srcPos);

target.setScale(srcScale);

}));

if(times== 0){

target.runAction(seq.repeatForever());

}else{

target.runAction(seq.repeat(times));

}

}

* + 接口调用:

**SingleScale(cc.p(cc.winSize.width\* 0.85, cc.winSize.height\* 0.5),**

**cc.p(cc.winSize.width\* 0.75, cc.winSize.height\* 0.5),**

**1, 0.5, 0.5, 0, circle2);**

* 单方向移动算法-移动到目标位置，重回到起点【不做动作】
  + 代码实现:

/\*

func:

单方向移动算法-移动到目标位置，重回到起点【不做动作】

@srcPos:起始位置

@desPos:终点位置

@time:单方向运动时间

@times:运动次数

@target:目标

\*/

function SingleReturn(srcPos, desPos, time, times, target){

var move= cc.moveTo(time, desPos);

var seq= cc.sequence(move, cc.callFunc(function(pSender){

target.setPosition(srcPos);

}));

if(times== 0){

target.runAction(seq.repeatForever());

}else{

target.runAction(seq.repeat(times));

}

}

* + 接口调用:

SingleReturn(srcPos, desPos, 0.5, 0, wood);

* 单方向移动算法-移动的同时，缩小&&降低不透明度
  + 代码实现:

/\*

func:

单方向移动算法-移动的同时，缩小&&降低不透明度

@srcPos:起始位置

@desPos:终点位置

@srcScale:起始缩放系数

@desScale:终止缩放系数

@time:单方向运动时间

@times:运动次数

@target:目标

\*/

function SingleScaleAndFade(srcPos, desPos, srcScale, desScale, time, times, target){

var move= cc.moveTo(time, desPos);

var scale= cc.scaleTo(time, desScale);

var fadeOut= cc.fadeOut(time);

var spawn= cc.spawn(move, scale, fadeOut);

var seq= cc.sequence(spawn, cc.callFunc(function(pSender){

target.setPosition(srcPos);

target.setScale(srcScale);

target.setOpacity(255);

}));

if(times== 0){

target.runAction(seq.repeatForever());

}else{

target.runAction(seq.repeat(times));

}

}

* + 接口调用:
* 移动到超过终点的位置，然后再移动到终点位置
  + 代码实现:

/\*

func:

单方向移动——移动到超过终点的位置，然后再移动到终点位置

@midPos:中间位置

@desPos:终点位置

@time:单方向移动时间

@times:动作执行次数

@target:动作执行目标

\*/

function SingleMove(midPos, desPos, time, times, target){

var move1= cc.moveTo(time, midPos);

var move2= cc.moveTo(time\* 0.3, desPos);

var seq= cc.sequence(move1, move2);

if(times== 0){

target.runAction(seq.repeatForever());

}else{

target.runAction(seq.repeat(times));

}

}

* + 接口调用:

**SingleMove(cc.p(cc.winSize\* 0.5, cc.winSize.height\* 0.6), cc.p(cc.winSize.height\* 0.5, cc.winSize.height\* 0.5), 1, 0, target);**

1. 晃动算法【JS实现】

* 代码实现:

/\*

func:目标执行晃动效果

principle:一次晃动需要三个过程:

1、从中心位置-->一个方向，移动step距离;

2、向另一个方向，移动step\* 2的距离;

3、当前位置-->中心位置，移动step距离;

@direction:晃动方向——【水平true,还是竖直false】

@step:单方向偏移距离【正负符号不同时,初始运动方向不同】

@time:单方向偏移时间

@times:晃动次数

@target:要执行晃动的目标

\*/

function Shake(direction, step, time, times, target){

var pos= direction?cc.p(step, 0):cc.p(0, step);

var move1= cc.moveBy(time, pos);//步骤1

var move3= move1.clone();//步骤3

pos= cc.pMult(pos, -2);

var move2= cc.moveBy(time\* 2, pos);

var seq= cc.sequence(move1, move2, move3);

if(times== 0){

target.runAction(seq.repeatForever());

}else{

target.runAction(seq.repeat(times));

}

}

* 接口调用:

**Shake(true, 20, 0.1, 0, phone);**

1. 摆动算法:【JS实现】

* 代码实现:

/\*

func:目标执行摆动效果

principle:一次晃动需要三个过程:

1、从中心位置-->一个方向的最大摆动位置，转动angle角度;

2、向另一个方向，转动angle\* 2角度;

3、当前最大摆动位置-->中心位置，转动angle角度;

@angle:单方向偏移距离【正负符号不同时,初始运动方向不同】

@time:单方向偏移时间

@times:摆动次数【0—不停止】

@target:要执行晃动的目标

\*/

function Swing(angle, time, times, target){

var move1= cc.rotateBy(time, angle);//步骤1

var move3= move1.clone();//步骤3

var move2= cc.rotateBy(time\* 2, angle\* (-2));

var seq= cc.sequence(move1, move2, move3);

if(times== 0){

target.runAction(seq.repeatForever());

}else{

target.runAction(seq.repeat(times));

}

}

* 接口调用:

**Swing(20, 0.1, 0, phone);**

1. 树苗成长、摇摆算法:

* 原理:

1、锚点设置:(0.5, 0)

2、树木由小变大的同时，执行摇摆算法；

* 同理:木马的锚点，也是(0.5, 0)
* 示例:

/\*

func:树苗成长以后,随风摇摆

@desScale:最终尺寸

@timeGrow:成长时间

@times:执行次数

@target:执行目标

\*/

function GrowUp(desScale, timeGrow, target){

var scale= cc.scaleTo(timeGrow, desScale);

target.runAction(scale);

}

/\*

func:目标执行摆动效果

principle:一次晃动需要三个过程:

1、从中心位置-->一个方向的最大摆动位置，转动angle角度;

2、向另一个方向，转动angle\* 2角度;

3、当前最大摆动位置-->中心位置，转动angle角度;

@angle:单方向偏移距离【正负符号不同时,初始运动方向不同】

@time:单方向偏移时间

@times:摆动次数【0—不停止】

@target:要执行晃动的目标

\*/

function swing(angle, time, times, target){

var move1= cc.rotateTo(time, angle);//步骤1

var move3= cc.rotateTo(time, 0);//步骤3

var move2= cc.rotateTo(time\* 2, angle\* (-2));

var seq= cc.sequence(move1, move2, move3);

if(times== 0){

target.runAction(seq.repeatForever());

}else{

target.runAction(seq.repeat(times));

}

}

1. 心跳算法

* 原理:

隔一段时间,放大;再隔一段时间,还原;重复即可。【可以使用一个定时器，这次执行放大操作，下一次执行还原操作。】

* 示例:

/\*

func:心跳动画

principle:

隔一段时间,放大;

再隔一段时间,还原;

重复即可

@srcScale:初始尺寸

@desScale:最终尺寸

@time:跳动一下，执行时间

@times:执行次数

@target:执行目标

\*/

function HeartBeat(srcScale, desScale, time, times, target){

var delay1= cc.delayTime(time);

var delay2= delay1.clone();

var seq1= cc.sequence(delay1, cc.callFunc(function(pSender){

pSender.setScale(desScale);

}));

var seq2= cc.sequence(delay2, cc.callFunc(function(pSender){

pSender.setScale(srcScale);

}));

var seq3= cc.sequence(seq1, seq2);

if(times!=0){

target.runAction(seq3.repeat(times));

}else{

target.runAction(seq3.repeatForever());

}

}

* 接口调用:

**HeartBeat(1, 1.1, 0.5, 0, crown);**

1. 盖章算法:

* 原理:

由很大的缩放系数,短时间内突变成正常缩放系数，即可实现盖章效果。反之，由正常缩放系数，短时间内突变成很大，然后消失，即可实现飞出屏幕的效果。

* 代码实现:

/\*

func:

盖章算法

principle:

由很大的缩放系数,短时间内突变成正常缩放系数，即可实现盖章效果。

反之，由正常缩放系数，短时间内突变成很大，然后消失，即可实现飞出屏幕的效果。

@srcScale:起始缩放系数

@desScale:最终缩放系数

@time:单方向移动时间

@target:动作执行目标

\*/

function Seal(srcScale, desScale, time, target){

target.setScale(srcScale);

var scale= cc.scaleTo(time, desScale);

target.runAction(scale);

}

* 接口调用:

**Seal(5, 1, 0.2, target);**

1. 碰撞检测算法:

/\*

func:

碰撞检测方式

\*/

if(typeof g\_detectionWay== "undefined"){

var g\_detectionWay= {

AnchorPosInRect:0,//物体1的锚点是否在物体2的矩形区域内

RectIntersectsRect:1,//两个物体是否相交

RectContainsRect:2//物体2是否包含物体1

};

}

/\*

func:

两个物体碰撞检测

@target1:物体1

@target2:物体2

@detectionWay:碰撞检测方式

\*/

function CollisionDetection(target1, target2, detectionWay){

var target1Size= cc.size(target1.getContentSize().width\* target1.getScaleX(), target1.getContentSize().height\* target1.getScaleY());

var target2Size= cc.size(target2.getContentSize().width\* target2.getScaleX(), target2.getContentSize().height\* target2.getScaleY());

var pos= target2.convertToNodeSpace(target1.getPosition());

var rect1= cc.rect(pos.x, pos.y, target1Size.width, target1Size.height);

var rect2= cc.rect(0, 0, target2Size.width, target2Size.height);

switch(detectionWay){

case g\_detectionWay.AnchorPosInRect:

return cc.rectContainsPoint(rect2, pos);

case g\_detectionWay.RectIntersectsRect:

return cc.rectIntersectsRect(rect2, rect1);

case g\_detectionWay.RectContainsRect:

return cc.rectContainsRect(rect2, rect1);

}

}

1. 显示和隐藏动画:

* 显示:
  + 1、由某点-->另一点,移动的同时,放大:
    - 实现:

/\*

func:

显示——移动的同时,放大

@desPos:终点坐标,

@desScale:最终的缩放系数

@time:执行时间

@target:目标

\*/

function Show(desPos, desScale, time, target){

var scale= cc.scaleTo(time, desScale);

var move= cc.moveTo(time,desPos);

var spawn= cc.spawn(scale, move);

var seq= cc.sequence(spawn, cc.callFunc(function(pSender){

CallBack\_Show();

}));

target.runAction(seq);

}

/\*

func:

显示成功后的回调函数

\*/

function CallBack\_Show(){

}

* + - 接口调用:

**Show(cc.p(cc.winSize.width\* 0.5, cc.winSize.height\* 0.5), 1, 0.1, popup);**

* 隐藏:
  + 1、由某点-->另一点,移动的同时,缩小
    - 实现:

/\*

func:

隐藏——移动的同时,缩小

@desPos:终点坐标

@desScale:最终的缩放系数

@time:执行时间

@target:目标

\*/

function Hide(desPos, desScale, time, target){

var scale= cc.scaleTo(time, desScale);

var move= cc.moveTo(time,desPos);

var spawn= cc.spawn(scale, move);

var seq= cc.sequence(spawn, cc.callFunc(function(pSender){

CallBack\_Hide();

}));

target.runAction(seq);

}

/\*

func:

隐藏成功后的回调函数

\*/

function CallBack\_Hide(){

}

* + - 接口调用:

**Hide(cc.p(cc.winSize.width, 0), 0, 0.1, popup);**

1. 切换文本算法:

* 功能:

文本做小-->大-->小的动作,在执行完由小-->大的动作之后，更改文本内容。

* 代码实现:

/\*

func:

文本做小-->大-->小的动作,在执行完由小-->大的动作之后，更改文本内容

@srcScale:起始缩放系数

@desScale:最终缩放系数

@time:动作执行时间

@times:动作执行次数

@target:动作执行目标

@callback:回调函数

\*/

function ChangeText(srcScale, desScale, time, times, target, callback){

if(callback== undefined){callback= null;}

var scale1= cc.scaleTo(time, desScale);

var scale2= cc.scaleTo(time, srcScale);

var seq= cc.sequence(scale1, cc.callFunc(callback),scale2);

if(times== 0){

target.runAction(seq.repeatForever());

}else{

target.runAction(seq.repeat(times));

}

}

function ChangeMsg(pSender){

g\_iScore++;

pSender.setString(g\_iScore);

}

* 接口调用:

**ChangeText(1, 2, 0.5, 0, label, ChangeMsg);**

1. 几个对象都满足某一条件时,才可以执行某项操作【JS实现】

/\*

func:几个对象都满足某一条件时,才可以

principle:

遍历几个对象，有一个对象不满足时[本实例以《是否<=3》为判定条件],就返回false.

@array:几个对象组成的数组

\*/

function bAllSuccess(array){

for(var i=0;i<array.length; ++i){

if(array[i]<=3){

return false;

}

}

return true;

}

//调用

var array= [1, 4];

bAllSuccess(array)

1. JS数组中的特殊排序算法:

* 缺省时排序：

array.sort();

结果：0,1,10,11,12,13,14,2,3,4,5,6,7,8,9

* 升序：

array.sort(function(a,b){return a>b?1:-1});

结果：0,1,2,3,4,5,6,7,8,9,10,11,12,13,14

* 降序：

array.sort(function(a,b){return a<b?1:-1});

结果：14,13,12,11,10,9,8,7,6,5,4,3,2,1,0

* 混乱排列：

array.sort(function(){return Math.random()<0.5?0:1;});

结果：7,0,13,3,4,5,6,12,8,9,11,10,1,2,14

1. 往返运动，记录运动的次数【JS】

/\*

func:往返运动中，记录运动的次数

principle:

往返运动中，虽然某个值呈现波浪式变化，但是在单程运动中，该值是唯一的。

因此，只要判定运动中的该值是否超越某一条界限，如果超过了，就加一。

问题:在单程运动中，只要超过了该界限，那么以下的短时间运动，也会超越，因此应该加一个开关。

说明:

该实例中,以位置的X坐标是否超过(中心位置的X坐标+ 30)

\*/

//是否超过界限

var g\_bOverRange= false;

//次数

var g\_times= 0;

function record(target){

var layer= target.getParent();

var label= layer.getChildByTag(1);

var posX= target.getPositionX();

if(posX> (cc.winSize.width\* 0.5+ 30)){

if(!g\_bOverRange){

g\_bOverRange= true;

g\_times++;

label.setString(g\_times);

}

}else{

g\_bOverRange= false;

}

}

1. 鼠标按下或者按住屏幕

* 拖动精灵或者滑动《游戏规则》:
* 原理:

1. 点击时，获取此时的位置(x,y),然后时刻获取鼠标或触摸点的位置，进而得到移动距离。
2. 设定:精灵的位置= 精灵的初始位置+ 偏移量;

* 实现:
  + C++:

在MouseMove()的回调函数，获取鼠标的位置。

* + JS:

在onTouchMoved(touch, event)的回调函数中，直接获取偏移量:

var delta= touch.getDelta();//偏移量

1. 添加监听机制:

/\*

func:监听机制

\*/

function addTouch(target){

var listener= cc.EventListener.create({

event:cc.EventListener.TOUCH\_ONE\_BY\_ONE,

swallowTouches:true,

onTouchBegan:onTouchBegan,

onTouchMoved:onTouchMoved,

onTouchEnded:onTouchEnded

});

cc.eventManager.addListener(listener, target);

}

//开始点击

function onTouchBegan(touch, event){

var target= event.getCurrentTarget();

var size= target.getContentSize();

var rect= cc.rect(0, 0, size.width, size.height);

var locationPos= target.convertToNodeSpace(touch.getLocation());

if(cc.rectContainsPoint(rect, locationPos)){

//开启蓄力操作

target.schedule(target.timer, 0.1);

return true;

}

return false;

}

//移动

function onTouchMoved(touch, event){

var delta= touch.getDelta();

var target= event.getCurrentTarget().getChildByTag(1);

//拖动算法

Drag(delta, target);

}

//释放

function onTouchEnded(touch, event){

var target= event.getCurrentTarget();

//关闭蓄力操作

target.unschedule(target.timer);

}

1. 拖动算法:

/\*

func:拖动算法

principle:

要移动的位置= 当前位置+ 移动位置

@

\*/

function Drag(delta, target){

target.setPosition(cc.pAdd(target.getPosition(), delta));

}

* 按住时，精灵移动;释放时,精灵停止【按住蓄力】:
* 原理:

设置一个定时器，点击时，开启，释放时，关闭。在定时器中，移动精灵或者能量值增加。

* 实现:

1. 设定蓄力定时器:

//定时器-【蓄力功能】

timer:function(delta){

g\_times++;

var label= this.getChildByTag(1);

label.setString(g\_times);

}

1. 在onTouchBegan函数中，添加:

//开启蓄力操作

target.schedule(target.timer, 0.1);

1. 在onTouchEnded函数中，添加:

//关闭蓄力操作

target.unschedule(target.timer);

1. 物体沿着抛物线运动:

抛物线方程有两种:

1. y= ax^2+bx+c

原理:

根据抛物线方程ax^2+bx+c=y，已知(x1, y1)、(x2, y2)、(x3, y3),得方程组:

ax1^2+bx1+c=y1

ax2^2+bx2+c=y2

ax3^2+bx3+c=y3

解方程组得抛物线的a,b,c

a = ((x1- x3)\* (y1- y2)- (x1- x2)\* (y1- y3))/(x1- x2)\* (x1- x3)\* (x2- x3);

b = ((y1- y2)-a\* (x1\* x1- x2\* x2))/(x1- x2);

c = y1-a\* x1\* x1-b\* x1;

1. **计算抛物线方程系数：**

/\*

func:

计算抛物线方程系数

存储原则:

NO.1:系数a

NO.2:系数b

NO.3:系数c

\*/

function CalculateCoefficient(startPos, endPos, height){

var arr= [0, 0, 0];

var centerPos= cc.p(0.5\* (startPos.x+ endPos.x), startPos.y+ height);

//系数:a

arr[0]= ((startPos.y- centerPos.y)\* (startPos.x- endPos.x)- (startPos.y- endPos.y)\* (startPos.x- centerPos.x))/((startPos.x- centerPos.x)\* (startPos.x- endPos.x)\* (centerPos.x- endPos.x));

//系数:b

arr[1]= ((startPos.y- centerPos.y)- arr[0]\*(startPos.x\* startPos.x- centerPos.x\* centerPos.x))/(startPos.x- centerPos.x);

//系数:c

arr[2]= (startPos.y- arr[0]\* startPos.x\* startPos.x- arr[1]\* startPos.x);

return arr;

}

1. **由抛物线方程，计算运动的坐标点:**

/\*

func:

由抛物线方程，计算运动的坐标点

return:运动的坐标点

\*/

function Parabola(arr, xStep){

var pos= cc.p(0, 0);

pos.x= xStep;

pos.y= arr[0]\* xStep\* xStep+ arr[1]\*xStep+ arr[2];

return pos;

}

1. **自定义类:**

var CParabolaSprite= cc.Sprite.extend({

startPos:cc.p(0, 0),//抛体运动的起点

endPos:cc.p(0, 0),//抛体运动的终点

offsetPos:cc.p(0, 0),//位置偏移量

risingHeight:0,//上升高度

coefficient:[],//抛物线系数[y= a\*x\*x+ b\*x + c]

posX:0,//X轴位移

stepX:0,//X轴瞬时位移

/\*

func:

由抛物线方程，计算运动的坐标点

parameter:

@startPos:抛物线起始抛点

@endPos:抛物线终止抛点

@height:抛物线的高度,

@stepX:X轴位移

@offsetPos:位置偏移量

\*/

Init:function(startPos, endPos, height, stepX, offsetPos){

this.startPos= startPos;

this.endPos= endPos;

this.risingHeight= height;

this.stepX= ((stepX== undefined)?1:stepX);

this.offsetPos= ((offsetPos== undefined)?cc.p(0, 0):offsetPos);

this.coefficient= CalculateCoefficient(startPos, endPos, height);

this.scheduleUpdate();

},

update:function(delta){

this.posX+= this.stepX;

var pos= Parabola(this.coefficient, this.posX);

/\*

//往返运动

if(this.posX> this.endPos.x){

pos= this.startPos;

this.posX= 0;

}

\*/

this.setPosition(cc.pAdd(pos, this.offsetPos));

}

});

1. **调用接口:**

1、在project.json文件中,添加该文件的引用:"src/CParabolaSprite.js"

2、调用接口:

initPlayer:function(){

var player= new CParabolaSprite(res.football\_png);

this.addChild(player, 1 ,1);

player.Init(cc.p(0, 0), cc.p(200, 100), 600, 2, cc.p(200, 200));

}

1. y^2=2px

原理:

**根据抛物线方程y^2=2px，已知(x1, y1),得方程组:**

**标准方程:**

**右开口抛物线：y^2=2px**

**左开口抛物线：y^2= -2px**

**上开口抛物线：x^2=2py**

**下开口抛物线：x^2=-2py**

**[p为焦准距（p>0）]**

**特点:**

**在抛物线y^2=2px中，焦点是(p/2，0），准线的方程是x= -p/2，离心率e=1，范围：x≥0；**

**在抛物线y^2= -2px 中，焦点是( -p/2，0），准线的方程是x=p/2，离心率e=1，范围：x≤0；**

**在抛物线x^2=2py 中，焦点是（0，p/2），准线的方程是y= -p/2,离心率e=1，范围：y≥0；**

**在抛物线x^2= -2py中，焦点是（0，-p/2），准线的方程是y=p/2，离心率e=1，范围：y≤0；**

**抛物线四种方程的异同**

**共同点：**

**①原点在抛物线上;**

**②对称轴为坐标轴;**

**③准线与对称轴垂直，垂足与焦点分别对称于原点，它们与原点的距离都等于一次项系数的绝对值的1/4**

**不同点：**

**①对称轴为x轴时，方程右端为±2px，方程的左端为y^2；对称轴为y轴时，方程的右端为±2py，方程的左端为x^2；**

**②开口方向与x轴（或y轴）的正半轴相同时，焦点在x轴（y轴）的正半轴上，方程的右端取正号；开口方向与x（或y轴）的负半轴相同时，焦点在x轴（或y轴）的负半轴上，方程的右端取负号。**

1. **示例:**

/\*

func:

计算抛物线方程中的准焦距p

\*/

function CalculateP(pos, direction, orientation){

var p= 0;

if(!direction){//竖直方向

if(!orientation){//开口向下

p= (pos.x\* pos.x)/(-2\* pos.y);

}else{//开口向上

p= (pos.x\* pos.x)/(2\* pos.y);

}

}else{//水平方向

if(!orientation){//开口向左

p= (pos.y\* pos.y)/(-2\* pos.x);

}else{//开口向右

p= (pos.y\* pos.y)/(2\* pos.x);

}

}

return p;

}

/\*

func:

由抛物线方程，计算运动的坐标点

parameter:

@p:准焦点

@posXOrY:目标在X轴或者Y轴上的位置

@direction:抛物线开口方向,默认竖直方向

@orientation:抛物线开口方向,默认向下

return:运动的坐标点

\*/

//Todo:没必要，每次计算传值

function Parabola(p, posXOrY, direction, orientation){

var pos= cc.p(0, 0);

if(!direction){//竖直方向

pos.y= Math.abs(posXOrY);

pos.y\*= (orientation)? 1: (-1);

pos.x= (posXOrY<0? -1: 1)\*(Math.sqrt(2\* p\* pos.y));

}else{//水平方向

pos.x= Math.abs(posXOrY);

pos.x\*= (orientation)? 1: (-1);

pos.y= (posXOrY<0? -1: 1)\*(Math.sqrt(2\* p\* pos.x));

}

return pos;

}

//计算方程有两种方法:

//1、已知抛物线上的某点(x1, y1);

//2、已知焦准距p

var CParabolaSprite= cc.Sprite.extend({

way:false,//默认为第1种方程

pos:cc.p(0, 0),//物线上的某点(x1, y1)

p:0,//准焦距

direction:false,//抛物线开口方向,默认竖直方向

orientation:false,//抛物线开口方向,默认向下

offsetPos:cc.p(0, 0),//位置的偏移量

step:0,//每帧的偏移值

posXOrY:0,//目标当前在X轴或者Y轴中的位置

bCallBack:false,//设置可否回调

callback:null,//回调接口

/\*

func:

由抛物线方程，计算运动的坐标点

principle:

已知抛物线上的某点(x1, y1)或者准焦距p;

parameter:

@p:物线上的某点(x1, y1)或者焦准距p

@direction:抛物线开口方向【true- 水平 false- 竖直】

@orientation:抛物线开口方向【true- 向上或者向右 false- 向下或者向左】

@offsetPos:位置的偏移值

@step:每帧的偏移值

\*/

InitWay:function(p, direction, orientation){

this.direction= direction;

this.orientation= orientation;

if(cc.isObject(p)){//抛物线上的某点

this.way= false;

this.pos= p;

this.p= CalculateP(this.pos, direction, orientation);

}else{//准焦点

this.way= true;

this.p= p\* (orientation? 1: (-1));

}

this.scheduleUpdate();

},

/\*

func:

设置位置的偏移量

\*/

setOffsetPos:function(offsetPos){

this.offsetPos= (offsetPos== undefined)? cc.p(0, 0):offsetPos;

},

/\*

func:

设置每帧的偏移量

\*/

setStep:function(step){

this.step= (step== undefined)?100:step;

},

/\*

func:

设置可否回调

\*/

setCallback:function(callback){

if(callback== undefined){

this.bCallBack= false;

}else{

this.bCallBack= true;

this.callback= callback;

}

},

update:function(delta){

this.posXOrY+= this.step\* delta;

//在父节点相对坐标系中的位置

var pos= Parabola(this.p, this.posXOrY, this.direction, this.orientation);

pos= cc.pAdd( pos, this.offsetPos);

//在世界坐标系中的位置

var pos1= cc.pAdd(pos, this.getParent().getPosition());

//检测是否越界

if((pos1.x<=-50)|| (pos1.y<= -50)){

this.unscheduleUpdate();

if(this.bCallBack){//可否回调

this.callback(this.getParent());

}

this.removeFromParent(true);

return;

}

this.setPosition(pos);

}

});

**//调用**

**1、在project.json文件中,添加该文件的引用:"src/CParabolaSprite.js"**

**2、调用接口:**

**initPlayer:function(){**

**var player= new CParabolaSprite(res.football\_png);**

**this.addChild(player, 1 ,1);**

**//方法1、已知点cc.p(50, 50)在抛物线上**

**player.InitWay1(cc.p(50, 50), true, true, cc.p(320, 480), 2);**

**//方法2、已知抛物线的点焦距p【确保p值为正数】**

**player.InitWay2(25, true, true, cc.p(320, 480), 2);**

**}**

1. 在Cocos中添加微信头像,并缩放成固定大小:

* 定义一个全局变量的<img>标签,并设置其纹理路径:

var g\_image= new Image();

g\_image.src= "http://wx.qlogo.cn/mmopen/KydxAIB52xnFDwghvDYRqGwPyAdialHGr5G5c7EhUFXM283xSBHMsHKsBo73qQibwOwwicPdWkkLicHoK2MQYabibCzkEeawBQzNE/0";

* 添加头像精灵，并缩放成固定的尺寸:

var portraits= new cc.Sprite(g\_image.src);

portraits.attr({

x:cc.winSize.width\* 0.5,

y:cc.winSize.height\* 0.5,

scaleX:100/g\_image.width,

scaleY:100/g\_image.height

});

self.addChild(portraits, 2, 2);

1. 手机号

有时，需要预留玩家的信息，例如:手机号等等。此时，应该需要验证一下手机号是否符合格式，此时可以使用正则表达式。

1. 获取玩家输入的手机号:

//获取输入框的数据

var phone= document.getElementById("phone").value;

1. 利用正则表达式,进行验证:

//失败

if(!/^(13[\d]{9}|15[\d]{9}|17[\d]{9}|18[\d]{9})$/.test(phone)){

}

1. 因为有时玩家在输入时，会有意无意的添加一些空白字符，所以，在验证之前必须将一些空白字符先删除掉:

function rid(d){

while(d.indexOf(" ")!=-1){

d=d.replace(" ","");

}

return d;

}

phone= rid(phone);

1. 用户输入信息 仅限中文

现在网上大多数用于判断中文字符的Unicode码是:U+4E00~U+9FA5，这个范围是只是“中日韩统一表意文字”这个区间，但不是全部，如果要全部包含，则还要他们的扩展集、部首、象形字、注间字母等等;

2E80－A4CF　加上　F900-FAFF　加上　FE30-FE4F

* 2E80－A4CF

包含了中日朝部首补充、康熙部首、表意文字描述符、中日朝符号和标点、日文平假名、日文片假名、注音字母、谚文兼容字母、象形字注释标志、注音字母扩展、中日朝笔画、日文片假名语音扩展、带圈中日朝字母和月份、中日朝兼容、中日朝统一表意文字扩展A、易经六十四卦符号、中日韩统一表意文字、彝文音节、彝文字根

* F900-FAFF

中日朝兼容表意文字

* FE30-FE4F

中日朝兼容形式

所以，一般用4E00－9FA5已经可以，如果要更广，则用2E80－A4CF || F900-FAFF　||　FE30-FE4F。

* 实现:

1. 获取玩家输入信息:

var question= $("#question");

var str=question.attr("value");

1. 判定:

function bAllChinese(str){

for(var i=0;(i<str.length);++i){

if((str[i].charCodeAt()<= 0x4e00)|| (str[i].charCodeAt()>= 0x9fa5)){

return false;

}

}

return true;

}

* 类

1. 游戏规则类:【JS实现】

* 功能:

**长的游戏规则,可滑动查看，底部有一个返回按钮。**

* 原理:

**方法一、利用手机的屏幕有限，显示的精灵只能显示一部分，因此，可以通过【onTouchMoved中的touch.getDelta()】改变精灵的位置，来实现拖动《游戏规则》查看内容;**

**方法二、利用setTextureRect来控制精灵的显示区域;**

**方法三、利用裁剪节点cc.ClippingNode,来控制精灵的显示区域;**

* 示例:
  + 方法一:【详见CRuleSprite.js】

var CRuleSprite= cc.Sprite.extend({

posYLimit:0,//限制区域

Init:function(path){

this.initBg();

this.addMenu(path);

},

initBg:function(){

this.attr({

anchorX:0,

anchorY:1,

x:0,

y:cc.winSize.height

});

this.addListener();

this.posYLimit= this.getContentSize().height;

},

addMenu:function(path){

var menu= new cc.Menu();

menu.setPosition(cc.p(0, 0));

this.addChild(menu, 1, 1);

//Return

var Return= new cc.MenuItemImage(

path,null,

this.onReturn, this

);

Return.attr({

x:cc.winSize.width\* 0.5,

y:this.posYLimit\* 0.02

});

menu.addChild(Return, 1, 1);

},

addListener:function(){

var self= this;

var listener= cc.EventListener.create({

event:cc.EventListener.TOUCH\_ONE\_BY\_ONE,

swallowTouches:true,

onTouchBegan:self.onTouchBegan,

onTouchMoved:self.onTouchMoved

});

cc.eventManager.addListener(listener, this);

},

onTouchBegan:function(touch, event){

var target= event.getCurrentTarget();

var size= target.getContentSize();

var rect= cc.rect(0,0, size.width, size.height);

var locationPos= target.convertToNodeSpace(touch.getLocation());

return cc.rectContainsPoint(rect,locationPos);

},

onTouchMoved:function(touch, event){

var target= event.getCurrentTarget();

target.moveByHand(touch.getDelta());

},

moveByHand:function(delta){

var desPosY= this.getPositionY()+ delta.y;

if(desPosY<= cc.winSize.height){//限制精灵-过分向下滑动

desPosY= cc.winSize.height;

}else if(desPosY>= this.posYLimit){//限制精灵-过分向上滑动

desPosY= this.posYLimit;

}

this.setPositionY(desPosY);

},

onReturn:function(pSender){

//点击《返回》按钮,直接删除该精灵

this.removeFromParent(true);

}

});

* + 接口调用:

**var rule= new CRuleSprite(res.bg\_rule\_jpg);**

**rule.Init(res.btn\_return\_png);**

**this.addChild(rule, 1, 6);**

* + 方法二:

var CRuleSprite= cc.Sprite.extend({

size:cc.size(0, 0),//精灵的尺寸

canvasSize:cc.size(0, 0),//窗体的有效尺寸

curPosY:0,

Init:function(path){

this.attr({

anchorX:0,

anchorY:1,

x:0,

y:cc.winSize.height

});

this.size= this.getContentSize();

//menu

var menu= new cc.Menu();

menu.setPosition(0, 0);

this.addChild(menu,1, 1);

//Return

var Return= new cc.MenuItemImage(

path,null,

this.onReturn,this

);

Return.setVisible(false);

Return.setPosition(cc.winSize.width\* 0.5, 50);

menu.addChild(Return,1,1);

this.curPosY= 0;

//canvas的尺寸

this.canvasSize= cc.view.getDesignResolutionSize();

this.setTextureRect(cc.rect(0, this.curPosY, this.canvasSize.width, this.canvasSize.height));

this.addSelfListener();

},

onReturn:function(pSender){

var scene= cc.director.getRunningScene();

scene.removeAllChildren(true);

scene.addChild(new MenuLayer(),g\_LayerTag.BgLayerTag,g\_LayerTag.BgLayerTag);

},

addSelfListener:function(){

var self= this;

var listener= cc.EventListener.create({

event:cc.EventListener.TOUCH\_ONE\_BY\_ONE,

swallows:true,

onTouchBegan:self.onTouchBegan,

onTouchMoved:self.onTouchMoved

},this);

cc.eventManager.addListener(listener,this);

},

onTouchBegan:function(touch,event){

var target= event.getCurrentTarget();

var size= target.getContentSize();

var rect= cc.rect(0,0,size.width,size.height);

var locationPos= touch.getLocation();

return cc.rectContainsPoint(rect,locationPos);

},

onTouchMoved:function(touch,event){

var target= event.getCurrentTarget();

var Return= target.getChildByTag(1).children[0];

var delta= touch.getDelta();

target.curPosY+=delta.y;

if(target.curPosY<0){

target.curPosY=0;

}else if(target.curPosY> (target.size.height- target.canvasSize.height)){

target.curPosY=(target.size.height- target.canvasSize.height);

}else if(target.curPosY> (target.size.height- target.canvasSize.height- 50)){

Return.setVisible(true);

}else{

Return.setVisible(false);

}

target.setTextureRect(cc.rect(0, target.curPosY, target.canvasSize.width, target.canvasSize.height));

}

});

* + 接口调用:

**var rule= new CRuleSprite(res.bg\_rule\_jpg);**

**rule.Init(res.btn\_return\_png);**

**this.addChild(rule, 1, 6);**

1. 分享引导或者游戏引导类

* 功能:

**点击一下，该精灵消失。**

* 原理:

**在原来的cc.Sprite类的基础上,添加监听机制,onTouchEnded()中，释放该精灵即可。**

* 实现:

var CGuide= cc.Sprite.extend({

bCallback:false,//是否需要回调

callback:null,//回调函数

/\*

func:

初始化，添加监听机制

@time:渐现时间,值为0时,不执行渐现动作

@arrItem:菜单项数组——[纹理路径, 位置]

设置arrItem时，一定要time

\*/

Init:function(time, arrItem){

var self= this;

self.setPosition(cc.winSize.width\* 0.5, cc.winSize.height\* 0.5);

if(arrItem== undefined){

self.addTouch();

}else{

self.addMenu(arrItem);

}

if((time!= 0)&&(time!= undefined)){

self.Enter();

}

},

/\*

func:

添加菜单项

@arrItem:菜单项数组

\*/

addMenu:function(arrItem){

var self= this;

var menu= new cc.Menu();

menu.setPosition(cc.p(0, 0));

self.addChild(menu, 1, 1);

var item= new cc.MenuItemImage(

arrItem[0],null,

self.onSelected,self

);

item.setPosition(arrItem[1]);

menu.addChild(item);

},

/\*

func:

监听机制

\*/

addTouch:function(){

var self= this;

var listener= cc.EventListener.create({

event:cc.EventListener.TOUCH\_ONE\_BY\_ONE,

swallowTouches:true,

onTouchBegan:self.onTouchBegan,

onTouchEnded:self.onTouchEnded

});

cc.eventManager.addListener(listener, self);

},

/\*

func:

开始触摸——判断是否触摸在有效区域

\*/

onTouchBegan:function(touch, event){

var target= event.getCurrentTarget();

var size= target.getContentSize();

var rect= cc.rect(0, 0, size.width, size.height);

var locationPos= target.convertToNodeSpace(touch.getLocation());

return cc.rectContainsPoint(rect, locationPos);

},

/\*

func:

结束触摸——回调函数

\*/

onTouchEnded:function(touch, event){

var target= event.getCurrentTarget();

target.onSelected();

},

/\*

func:

被选中

\*/

onSelected:function(){

var self= this;

//添加需要回调的相应代码

if(self.bCallback){

self.callback(self.getParent());

}else{

self.removeSelf();

}

},

/\*

func:

设置回调函数

@callback:回调函数

\*/

SetCallback:function(callback){

var self= this;

if(callback== undefined){

self.bCallback= false;

}else{

self.bCallback= true;

self.callback= callback;

}

},

/\*

func:

删除自身

@time:渐隐动作执行时间

\*/

removeSelf:function(time){

var self= this;

if((time!= 0)&&(time!= undefined)){

self.Exit(time);

}else{

self.removeFromParent(true);

}

},

/\*

func:

显示时，渐现

@time:渐现时间

\*/

Enter:function(time){

var self= this;

var fadeIn= cc.fadeIn(time);

self.runAction(fadeIn);

},

/\*

func:

退出时，渐隐

@time:渐隐时间

\*/

Exit:function(time){

var self= this;

var fadeOut= cc.fadeOut(time);

var seq= cc.sequence(fadeOut, cc.callFunc(function(pSender){

pSender.removeFromParent(true);

}));

self.runAction(fadeOut);

}

});

* 接口调用:

**var guide= new CGuide(res.guide\_png);**

**guide.Init();**

**this.addChild(guide, 1, 2);**

1. 精灵的5种显示方式类

/\*

func:

精灵的显示:不全-->全

精灵的隐藏:全-->不全

interface:

popup.ShowOrHide(true, 1, g\_wayTag.Center);

\*/

/\*

要显示的方式

\*/

if(typeof g\_wayTag== "undefined"){

g\_wayTag= {

Center:0,//居中

LeftTop:1,//左上

LeftDown:2,//左下

RightTop:3,//右上

RightDown:4//右下

}

}

/\*

配置数据

行数[下标:0~4]-居中、左上、左下、右上、右下

列数:

0-精灵的锚点

1-精灵的位置

2-在精灵上要裁剪的初始位置

3-c1系数,

4-c2系数

\*/

var g\_clipIni= [

[cc.p(0.5, 0.5), cc.p(0, 0), cc.p(0.5, 0.5), 0.5, 2, 1, 1, -2, cc.p(0, 0)], //居中

[cc.p(0, 1), cc.p(-0.5, 0.5), cc.p(0, 0), 1, 1, 0, 0, -1, cc.p(-0.5, 0.5)], //左上

[cc.p(0, 0), cc.p(-0.5, -0.5), cc.p(0, 1), 1, 1, 0, 1, -1, cc.p(-0.5, -0.5)], //左下

[cc.p(1, 1), cc.p(0.5, 0.5), cc.p(1, 0), 1, 1, 1, 0, -1, cc.p(0.5, 0.5)], //右上

[cc.p(1, 0), cc.p(0.5, -0.5), cc.p(1, 1), 1, 1, 1, 1, -1, cc.p(0.5, -0.5)] //右下

];

var g\_anchorID= 0;

var g\_xyID= 1;

var g\_initPosID= 2;

var g\_show\_c0ID= 3;

var g\_show\_c3ID= 4;

var g\_hide\_c1ID= 5;

var g\_hide\_c2ID= 6;

var g\_hide\_c3ID= 7;

var g\_hide\_c4ID= 8;

var CClipSprite= cc.Sprite.extend({

size:cc.size(0,0),//尺寸

fHSpeed:0,//每秒钟移动的像素个数【横向】

fVSpeed:0,//每秒钟移动的像素个数【纵向】

time:0,//显示或者隐藏的时间

bShow:false,//是否为显示

way:g\_wayTag.Center,//动画方式

c1:0,//系数1

c2:0,//系数2

c3:0,//系数3

c4:cc.p(0, 0),//系数4

/\*

func:

根据显示或者隐藏的方式不同，设置不同的锚点、初始位置和已经在精灵上的裁剪位置。

\*/

getPos:function(){

this.setAnchorPoint(g\_clipIni[this.way][g\_anchorID]);

this.setPosition(cc.pAdd(this.getPosition(), cc.p(g\_clipIni[this.way][g\_xyID].x\* this.size.width, g\_clipIni[this.way][g\_xyID].y\* this.size.height)));

return cc.p(g\_clipIni[this.way][g\_initPosID].x\* this.size.width, g\_clipIni[this.way][g\_initPosID].y\* this.size.height);

},

/\*

func:

获取在竖直和水平的速度【每秒移动多少个像素】

principle:

竖直速度= 在该方向上要移动的距离(居中显示或隐藏时，为一半;其他方式,均为整个)/执行时间

水平速度= 在该方向上要移动的距离(居中显示或隐藏时，为一半;其他方式,均为整个)/执行时间

\*/

getSpeed:function(){

var coefficient= g\_clipIni[this.way][g\_show\_c0ID];

this.fHSpeed= coefficient\* this.size.width/ this.time;

this.fVSpeed= coefficient\* this.size.height/ this.time;

},

/\*

func:

计算4个系数

\*/

calculateCoefficient:function(){

if(this.bShow){

this.c1= -1;

this.c2= -1;

this.c3= g\_clipIni[this.way][g\_show\_c3ID]

}else{

this.c1= g\_clipIni[this.way][g\_hide\_c1ID];

this.c2= g\_clipIni[this.way][g\_hide\_c2ID];

this.c3= g\_clipIni[this.way][g\_hide\_c3ID];

this.c4= g\_clipIni[this.way][g\_hide\_c4ID];

}

},

/\*

func:

执行方式:显示 Or 隐藏

@bShow:显示true, 隐藏false

@time:动画执行时间

@way:动画执行方式

\*/

ShowOrHide:function(bShow, time, way){

this.bShow= bShow;

this.time= time;

this.way= way;

this.size= this.getContentSize();

this.calculateCoefficient();

this.getSpeed();

if(bShow){

var pos= this.getPos();

this.setTextureRect(cc.rect(pos.x, pos.y, 0, 0));

}

this.scheduleUpdate();

},

update:function(delta){

var HStep= (this.fHSpeed\* delta);

var VStep= (this.fVSpeed\* delta);

var rect= this.getTextureRect();

if(!this.bShow){

this.x+= (HStep\* this.c4.x);

this.y+= (VStep\* this.c4.y);

}

rect.x+= HStep\* this.c1;

rect.y+= VStep\* this.c2;

rect.width+= HStep\* this.c3;

rect.height+= VStep\* this.c3;

if(rect.x<= 0){rect.x= 0;}

if(rect.y<= 0){rect.y= 0;}

if(rect.width<= 0){rect.width= 0;}

if(rect.height<= 0){rect.height= 0;}

this.time-= delta;

if(this.time< 0){

this.unscheduleUpdate();

if(!this.bShow){

/\*

有时候,虽然时间为空，但是width,height为负数，所以有时候会残留一块。

方法一、如果只是暂时隐藏，最好加上下面一行代码

this.setTextureRect(cc.rect(rect.x, rect.y, 0, 0));

方法二、如果是删除,可以直接加上下面一行代码

this.removeFromParent(true);

方法三、也可以先判定width,height,再做裁剪:

if(rect.width<= 0){rect.width= 0;}

if(rect.height<= 0){rect.height= 0;}

\*/

this.removeFromParent(true);

return;

}else{

rect= cc.rect(0, 0, this.size.width, this.size.height);

}

}

this.setTextureRect(rect);

}

});

1. 卷轴类

* 说明:

有水平、竖直两种显示或者隐藏方式，从中间-->两边、一边-->另一边、另一边-->一边，因此一共有2\* 3= 6种效果。

* 实现:

/\*

func:

卷轴动画显示

【竖直】

1、从中间向两边展开;

2、从上向下展开;

3、从下向上展开;

【水平】

4、从中间向两边展开;

5、从左向右展开;

6、从右向左展开;

卷轴动画隐藏

【竖直】

1、从中间向两边隐藏;

2、从上向下隐藏;

3、从下向上隐藏;

【水平】

4、从中间向两边隐藏;

5、从左向右隐藏;

5、从右向左隐藏;

\*/

if(typeof g\_wayTag== "undefined"){

var g\_wayTag={

V\_CenterToSide:0,

V\_TopToBottom:1,

V\_BottomToTop:2,

H\_CenterToSide:3,

H\_LeftToRight:4,

H\_RightToLeft:5

}

}

/\*

配置数据

行数(由0~2):

【竖直】

1、从中间向两边展开;

2、从上向下展开;

3、从下向上展开;

列数(由0~...):

锚点-卷轴中间图的坐标位置-卷轴上边X轴坐标-卷轴下边X轴坐标-系数c1-系数c2

centerXY-side1XY-side2XY-textureRect

\*/

var g\_rouleauIni= [

//竖直方向

[cc.p(0.5, 0.5), cc.p(0, 0), 0, 0, 0.5, 0.5, cc.p(0, 0), cc.p(0, 1), cc.p(0, -1), cc.rect(0, -1, 0, 2)],

[cc.p(0, 1), cc.p(-0.5, 0.5), 0.5, 0.5, 0, 1, cc.p(0, 0), cc.p(0, 0), cc.p(0, -1), cc.rect(0, 0, 0, 1)],

[cc.p(0, 0), cc.p(-0.5, -0.5), 0.5, 0.5, 1, 1, cc.p(0, 0), cc.p(0, 1), cc.p(0, 0), cc.rect(0, -1, 0, 1)],

//水平方向

[cc.p(0.5, 0.5), cc.p(0, 0), 0, 0, 0.5, 0.5, cc.p(0, 0), cc.p(-1, 0), cc.p(1, 0), cc.rect(-1, 0, 2, 0)],

[cc.p(0, 0.5), cc.p(-0.5, 0), -0.1, 0.1, 0, 1, cc.p(0, 0), cc.p(0, 0), cc.p(1, 0), cc.rect(-1, 0, 1, 0)],

[cc.p(1, 0.5), cc.p(0.5, 0), -0.1, 0.1, 1, 1, cc.p(0, 0), cc.p(-1, 0), cc.p(0, 0), cc.rect(-1, 0, 1, 0)]

];

var g\_anchorID= 0;

var g\_xyID= 1;

var g\_side1XID= 2;

var g\_side2XID= 3;

var g\_c1ID= 4;

var g\_c2ID= 5;

var g\_centerXYID= 6;

var g\_side1XYID= 7;

var g\_side2XYID= 8;

var g\_textureRectID= 9;

var CRouleau = cc.Node.extend({

way:g\_wayTag.V\_CenterToSide,//动画执行方式

center:null,//中间的精灵,

side1:null,//卷轴的一边,

side2:null,//卷轴的另一边,

pos:cc.p(0, 0),//节点所处位置

speed:0,//每秒钟的移动的像素个数

size:cc.size(0, 0),//卷轴中间部分的尺寸

time:0,//动画执行时间

step:0,//两卷轴之间的间隙

bShow:true,//显示,还是隐藏

bEnable:false,//可不可以被隐藏,避免显示不全时,玩家就隐藏出错

/\*

func:

初始化动画显示方式

@centerPicPath:中间图片路径;

@sidePic1Path:一边的卷轴图片;

@sidePic2Path:另一边的卷轴图片;

@way:动画执行方式

@time:动画执行时间

\*/

Init:function(centerPicPath, sidePic1Path, sidePic2Path, way, time){

this.way= way;

this.time= time;

this.pos= this.getPosition();

this.addRouleau(centerPicPath, sidePic1Path, sidePic2Path);

},

addRouleau:function(centerPicPath, sidePic1Path, sidePic2Path){

//Center

var center= new cc.Sprite(centerPicPath);

var side1= new cc.Sprite(sidePic1Path);

var side2= new cc.Sprite(sidePic2Path);

this.center= center;

this.side1= side1;

this.side2= side2;

this.size=center.getContentSize();

this.ShowRouleau();

},

ShowRouleau:function(){

this.bShow= true;

if(this.way< 3){//竖直

//初始化裁剪

this.step= (this.side1.getContentSize().height+ this.side2.getContentSize().height)\* 0.5;

this.center.setTextureRect(cc.rect(0, this.center.getContentSize().height\* (g\_rouleauIni[this.way][g\_c1ID]), this.size.width, 0));

}else{

this.step= (this.side1.getContentSize().width+ this.side2.getContentSize().width)\* 0.5;

this.center.setTextureRect(cc.rect(this.center.getContentSize().width\* g\_rouleauIni[this.way][g\_c1ID], 0, 0, this.size.height));

}

//中间

this.center.attr({

anchorX:g\_rouleauIni[this.way][g\_anchorID].x,

anchorY:g\_rouleauIni[this.way][g\_anchorID].y,

x:this.pos.x+ (this.size.width\* g\_rouleauIni[this.way][g\_xyID].x),

y:this.pos.y+ (this.size.height\* g\_rouleauIni[this.way][g\_xyID].y)

});

this.addChild(this.center);

var pos= this.center.getPosition();

if(this.way< 3){//竖直方向

this.side1.attr({

x:pos.x+ (this.size.width\* g\_rouleauIni[this.way][g\_side1XID]),

y:pos.y+ this.step\* (0)

});

this.addChild(this.side1);

this.side2.attr({

x:pos.x+ (this.size.width\* g\_rouleauIni[this.way][g\_side2XID]),

y:pos.y- this.step\* (0)

});

this.addChild(this.side2);

this.speed= (this.size.height\* g\_rouleauIni[this.way][g\_c2ID])/ this.time;

}else{//水平方向

this.side1.attr({

x:pos.x+ (this.step\* g\_rouleauIni[this.way][g\_side1XID]),

y:pos.y+ this.size.height\* (0)

});

this.addChild(this.side1);

this.side2.attr({

x:pos.x+ (this.step\* g\_rouleauIni[this.way][g\_side2XID]),

y:pos.y+ this.size.height\* (0)

});

this.addChild(this.side2);

this.speed= (this.size.width\* g\_rouleauIni[this.way][g\_c2ID])/ this.time;

}

this.scheduleUpdate();

},

HideRouleau:function(time){

if(!this.bEnable) return;

this.bShow= false;

this.time= time;

if(this.way< 3) {//竖直方向

this.speed= (this.size.height\* g\_rouleauIni[this.way][g\_c2ID])/ this.time;

}else{

this.speed= (this.size.width\* g\_rouleauIni[this.way][g\_c2ID])/ this.time;

}

this.scheduleUpdate();

},

update:function(delta){

var rect= this.center.getTextureRect();

var step= this.speed\* delta;

if(!this.bShow){

step\*= (-1);

}

this.center.x+= step\* g\_rouleauIni[this.way][g\_centerXYID].x;

this.center.y+= step\* g\_rouleauIni[this.way][g\_centerXYID].y;

//同步移动卷轴

this.side1.x+= step\* g\_rouleauIni[this.way][g\_side1XYID].x;

this.side1.y+= step\* g\_rouleauIni[this.way][g\_side1XYID].y;

this.side2.x+= step\* g\_rouleauIni[this.way][g\_side2XYID].x;

this.side2.y+= step\* g\_rouleauIni[this.way][g\_side2XYID].y;

rect.x+= step\* g\_rouleauIni[this.way][g\_textureRectID].x;

rect.y+= step\* g\_rouleauIni[this.way][g\_textureRectID].y;

rect.width+= step\* g\_rouleauIni[this.way][g\_textureRectID].width;

rect.height+= step\* g\_rouleauIni[this.way][g\_textureRectID].height;

//检测是否超出边界

if(rect.x<= 0){rect.x= 0;}

if(rect.y<= 0){rect.y= 0;}

if(rect.width<= 0){rect.width= 0;}

if(rect.height<= 0){rect.height= 0;}

this.time-= delta;

if(this.time<= 0){

this.bEnable= this.bShow;

this.unscheduleUpdate();

if(!this.bShow){

this.removeFromParent(true);

return;

}else{

rect= cc.rect(0, 0, this.size.width, this.size.height);

}

}

this.center.setTextureRect(rect);

}

});

* 接口调用:

//显示

var rouleau= new CRouleau();

rouleau.Init(res.rouleau2\_png, res.rouleau1\_png, res.rouleau1\_png, g\_wayTag.V\_TopToBottom, 5);

rouleau.setPosition(cc.p(cc.winSize.width\* 0.25 , cc.winSize.height\* 0.75));

this.addChild(rouleau, 1, 2);

//原方式隐藏

rouleau.HideRouleau(1);

1. 霓虹灯类

* 说明:

通过使用setColor函数,实现五颜六色的效果。会遭遇到一个问题:

手机的性能或者显卡不同,图片上可能不只是纯色,有时还会夹杂着黑色等等。颜色不纯。

【尤其是黄色和白色的时候,黑色效果更明显。】

上面可以放置文字。太阳光作为背景。

* 资源:

一条光柱图片

* 效果:



* 实现:

var CNeon= cc.Node.extend({

removeArr:[],//存储数组

colorArr:[],//颜色数组

startColor:cc.color(0, 0, 0),//起始颜色

endColor:cc.color(0, 0, 0),//终止颜色

num:0,//太阳光的数目

angle:0,//角度范围

radius:0,//半径

index:0,//当前颜色的下标

bRotated:false,//是否旋转

step:0,//每帧角度变化

bDifference:false,//颜色是否不同[true:不同 false:相同]

/\*

func:

初始化

@path:太阳光纹理路径信息

@radian:半径

@num:太阳光的数量

@angle:角度范围

@colorArr:颜色变化数组或者起始颜色

@endColor:终止颜色或者颜色变化时间,默认为0.3

\*/

Init:function(path, radius, num, angle, colorArr, endColor, time){

this.removeArr= new Array(0);

this.colorArr= new Array(0);

this.radius= radius;

this.num= num;

this.angle= angle;

var timer= 0;

if(time== undefined){

this.colorArr= colorArr;

timer= endColor;

}else{

if(time== undefined){

}else{

timer= time;

}

this.startColor= colorArr;

this.endColor= endColor;

this.calculateColor();

}

this.addItems(path);

this.schedule(this.updateColor, timer);

},

/\*

func:

1、按照要求,添加相应数量的光柱;

2、设置每一条光柱的颜色、位置和旋转角度;

@path:光柱的纹理路径

\*/

addItems:function(path){

var angle= this.angle/ this.num;

if(this.angle!= 360){

angle= this.angle/ (this.num- 1);

}

for(var i=0; i< this.num; ++i){

var light= new cc.Sprite(path);

//设置锚点-左部中间

light.setAnchorPoint(cc.p(0, 0.5));

//设置旋转角度

light.setRotation(angle\* i\* (-1));

//设置灯光的位置

light.setPosition(this.calculatePos(angle\* i));

light.setColor(this.colorArr[0]);

this.addChild(light, 1);

this.removeArr.push(light);

}

},

/\*

func:

设置是否旋转

@angle:旋转角度偏移量【正数- 顺时针旋转 负数- 逆时针旋转】

\*/

SetBRotated:function(angle){

this.bRotated= true;

this.step= angle;

},

/\*

func:

设置每条光柱是否相同

\*/

SetBDifference:function(){

this.bDifference= true;

},

/\*

func:

已知旋转到某个角度,计算其对应的位置

principle:

X= 半径\* cos(旋转角度的弧度表示)

Y= 半径\* sin(旋转角度的弧度表示)

@angle:旋转角度

\*/

calculatePos:function(angle){

var pos= cc.p(0, 0);

var radian= angle\* Math.PI/ 180;

pos.x= this.radius\* Math.cos(radian);

pos.y= this.radius\* Math.sin(radian);

return pos;

},

/\*

func:计算颜色数组

principle:

颜色是由RGB组成的,因此两种颜色的渐变,就是RGB的渐变。

渐变间隔r= (终止颜色的r- 起始颜色的r)/ 光柱数目

渐变间隔g= (终止颜色的g- 起始颜色的g)/ 光柱数目

渐变间隔b= (终止颜色的b- 起始颜色的b)/ 光柱数目

\*/

calculateColor:function(){

var r= 0, g= 0, b= 0;

r= (this.endColor.r- this.startColor.r)/ this.num;

g= (this.endColor.g- this.startColor.g)/ this.num;

b= (this.endColor.b- this.startColor.b)/ this.num;

for(var i=0;i< this.num; ++i){

var color= cc.color(this.startColor.r+ Math.floor(r\* i), this.startColor.g+ Math.floor(g\* i), this.startColor.b+ Math.floor(b\* i));

this.colorArr.push(color);

}

},

/\*

func:

1、遍历每一条光柱，并设置每一条对应的颜色;

2、

\*/

updateColor:function(delta){

//数组下标, 循环遍历

this.index= ((this.index+1)%this.colorArr.length);

//遍历每一条光柱,并设置每一条对应的颜色

for(var i=0;i< this.removeArr.length;++i){

//设置颜色

if(this.bDifference){//不同

//数组下标, 循环遍历

this.index= ((this.index+1)%this.colorArr.length);

}

this.removeArr[i].setColor(this.colorArr[this.index]);

//旋转

if(this.bRotated){

//计算目标角度的位置

this.removeArr[i].setPosition(this.calculatePos(this.removeArr[i].getRotation()\*(-1)+ this.step));

//旋转物体

this.removeArr[i].setRotation(this.removeArr[i].getRotation()+ this.step);

}

}

}

});

* 接口调用:

**var color= [**

**cc.color(255, 255, 255),**

**cc.color(255, 255, 0)**

**];**

**方式1:**

**sunlight1.Init(res.light\_png, 100, 10, 180, color, 0.3);**

**方式2:**

**sunlight2.Init(res.light\_png, 10, 10, 360, cc.color(255, 255, 255), cc.color(0, 0, 0), 0.1);**

**//设置是否旋转**

**sunlight.SetBRotated(2);**

**//设置每条光柱是否统一**

**sunlight.SetBDifference();**

1. 资源加载场景类

* 修改Resources.js文件, 加载场景所需要的资源数组:

//Loading页面所需要的资源

var preRes = {

loadBgLogo\_png:"res/loadBgLogo.png",

loadProcessUp\_png:"res/loadProcessUp.png",

loadProcessDown\_png:"res/loadProcessDown.png"

};

var g\_preRes = [];

for(var i in preRes){

g\_preRes.push(preRes[i]);

}

* Main.js中,接口调用:

**MyLoaderScene.preLoad(g\_preRes, g\_resources, function () {**

**cc.director.runScene(new GameScene());**

**}, this);**

* 实现:

var MyLoaderScene = cc.Scene.extend({

\_length : 0,//要加载的资源的数量

\_count : 0,//成功加载资源的数量

bgLayer:null,

onEnter: function () {

cc.Node.prototype.onEnter.call(this);

},

onExit: function () {

cc.Node.prototype.onExit.call(this);

},

init : function(){

this.initLayer();

this.initBg();

this.initProcessBar();

this.startLoading();

return true;

},

/\*

func:

添加cc.LayerColor

reason:

如果直接在cc.Scene中，添加除cc.Layer之外的子节点,

不同时机,可能会出现闪屏现象。因此需要添加一个cc.LayerColor

\*/

initLayer:function(){

// layer

var bgLayer = cc.LayerColor.create(cc.color(0, 0, 0));

bgLayer.setPosition(cc.visibleRect.bottomLeft);

this.addChild(bgLayer, 0);

this.bgLayer= bgLayer;

},

/\*

func:

加载页面的背景图

\*/

initBg:function(){

//Bg

var bg = new cc.Sprite(preRes.loadBgLogo\_png);

bg.attr({

x:cc.winSize.width/2,

y:cc.winSize.height/2

});

this.bgLayer.addChild(bg, 1);

},

/\*

func:

进度条【前景和后景】

principle:

锚点:从左到右【cc.p(0, 0.5)】

位置:获取后景的尺寸(如果加载成功了以后,获取真实尺寸。否则获取的尺寸为cc.size(0, 0))

\*/

initProcessBar:function(){

var ProcessBarDown = new cc.Sprite(preRes.loadProcessDown\_png);

ProcessBarDown.attr({

x:cc.winSize.width/2,

y:cc.winSize.height/2

});

this.bgLayer.addChild(ProcessBarDown, 10);

var bgSize= ProcessBarDown.getContentSize();

this.ProcessBarUp = new cc.Sprite(preRes.loadProcessUp\_png);

this.ProcessBarUp.attr({

anchorX:0,

anchorY:0.5,

x:2,

y:bgSize.height\* 0.5,

scaleX:0,

scaleY:1

});

ProcessBarDown.addChild(this.ProcessBarUp, 11);

},

initWithResources: function (resources, cb) {

if(typeof resources == "string") resources = [resources];

this.resources = resources || [];

this.\_length= this.resources.length;

this.cb = cb;

},

preLoading:function(preRes, resources, cb){

var self= this;

if(typeof preRes == "string") preRes = [preRes];

cc.loader.load(preRes, function(){

self.initWithResources(resources, cb);

self.init();

});

},

startLoading: function () {

var self = this;

var res = self.resources;

self.\_count = 0;

cc.loader.load(res, function(result, count){

self.\_count++;

self.updatePercent();

}, function(){

self.ProcessBarUp.setScaleX(1);

if(self.cb){

self.cb();

}

});

},

updatePercent: function () {

var self = this;

var count = self.\_count;

var length = self.\_length;

var percent = (count / length \* 100) | 0;

percent = Math.min(percent, 100);

self.ProcessBarUp.setScaleX(percent\* 0.01);

}

});

/\*

func:

预加载资源接口

\*/

MyLoaderScene.preLoad = function(preRes, resources, cb){

var myLoaderScene = new MyLoaderScene();

myLoaderScene.preLoading(preRes, resources, cb);

cc.director.runScene(myLoaderScene);

return myLoaderScene;

};

1. 精灵滚动类

* 代码实现:

/\*

func:精灵移动方向

\*/

if(typeof g\_moveDirection== "undefined"){

var g\_moveDirection= {

LeftToRight:0,//从左到右

RightToLeft:1,//从右到左

TopToBottom:2,//从上到下

BottomToTop:3//从上到下

};

}

/\*

数据配置

锚点anchor-参数1-参数2-参数3-参数4

\*/

if(typeof g\_dataInI== "undefined"){

var g\_dateInI= [

[cc.p(0, 0.5), 0, 1, 0, 0],//从左到右

[cc.p(1, 0.5), 1, -1, 0, 0],//从右到左

[cc.p(0.5, 1), 0, 0, 1, -1],//从左到右

[cc.p(0.5, 0), 0, 0, 0, 1]//从左到右

];

}

var g\_AnchorID= 0;

var g\_a1ID= 1;

var g\_a2ID= 2;

var g\_a3ID= 3;

var g\_a4ID= 4;

var CRollNode= cc.Node.extend({

spriteArr:[],//精灵数组

size:cc.size(0, 0),//尺寸

Scale:cc.p(0,0),//缩放比例系数

moveDirection:0,//移动方向

speed:0,//移动速度【像素/s】

times:0,//轮播次数

times\_callback:0,//回调次数

bCallBack:false,//是否回调

callback:null,//回调函数

/\*

@path:纹理路径

@pos:位置信息

@scale:缩放比例系数

@direction:移动方向

\*/

Init:function(path, pos, scale, direction){

var self= this;

self.times\_callback= 0;

self.spriteArr= new Array(0);

self.moveDirection= direction;

self.Scale= scale;

//获取要轮播的图片尺寸

var texture= cc.textureCache.addImage(path);

var size =self.size= texture.getContentSize();

//可显示区域尺寸

var visibleSize= cc.winSize;

//计算要实现轮播所必须的精灵数

var num=self.calculateNum(size, visibleSize)+ 1;

self.initSprite(texture, num, pos, scale);

},

/\*

func:设置移动数据

@speed:移动速度【像素/s】

@times:轮播次数【times为0时,一直执行】

@callback:回调函数

\*/

setFormat:function(speed, times, callback){

var self= this;

self.speed= speed;

self.times= times;

if(callback== undefined){

self.bCallBack= false;

}else{

self.bCallBack= true;

self.callback= callback;

}

self.scheduleUpdate();

},

/\*

func:

计算要实现轮播所必须的精灵数

@size:被切割的图的尺寸；

@visibleSize:屏幕适配方案尺寸

\*/

calculateNum:function(size, visibleSize){

var self= this;

var num= 0;

if(self.moveDirection<2){//水平方向

num= Math.ceil(visibleSize.width/ size.width);

}else{//竖直方向

num= Math.ceil(visibleSize.height/ size.height);

}

return num;

},

/\*

func:

填写精灵数组

@texture:被切割的纹理

@num:要切割成多少块

@pos:偏移位置

@scale:精灵要缩放尺寸

\*/

initSprite:function(texture, num, pos, scale){

var self= this;

for(var i=0; i< num; ++i){

var wave= new cc.Sprite(texture);

wave.attr({

x:cc.winSize.width\* (g\_dateInI[self.moveDirection][g\_a1ID])+self.size.width\* i\* (g\_dateInI[self.moveDirection][g\_a2ID])+ pos.x,

y:cc.winSize.height\* (g\_dateInI[self.moveDirection][g\_a3ID])+self.size.height\* i\* (g\_dateInI[self.moveDirection][g\_a4ID])+ pos.y,

anchorX:g\_dateInI[self.moveDirection][g\_AnchorID].x,

anchorY:g\_dateInI[self.moveDirection][g\_AnchorID].y,

scaleX:scale.x,

scaleY:scale.y

});

self.addChild(wave);

self.spriteArr.push(wave);

}

},

/\*

func:

更新位置

\*/

update:function(delta){

var self= this;

for(var i= 0;i<self.spriteArr.length; ++i){

if(self.moveDirection== g\_moveDirection.LeftToRight) {//从左到右

self.spriteArr[i].x +=self.speed\* delta\* (-1);

if(self.spriteArr[i].x<= (-self.size.width\*self.Scale.x)){

self.spriteArr[i].x+= (self.size.width\*self.Scale.x\*self.spriteArr.length)\* (1);

self.callBack();

}

}else if(self.moveDirection== g\_moveDirection.RightToLeft){

self.spriteArr[i].x +=self.speed\* delta\* (1);

if(self.spriteArr[i].x>= (self.size.width\*self.Scale.x+ cc.winSize.width)){

self.spriteArr[i].x+= (self.size.width\*self.Scale.x\*self.spriteArr.length)\* (-1);

self.callBack();

}

}else if(self.moveDirection== g\_moveDirection.TopToBottom){

self.spriteArr[i].y +=self.speed\* delta\* (1);

if(self.spriteArr[i].y>= (self.size.height\*self.Scale.y+ cc.winSize.height)){

self.spriteArr[i].y+= (self.size.height\*self.Scale.y\*self.spriteArr.length)\* (-1);

self.callBack();

}

}else if(self.moveDirection== g\_moveDirection.BottomToTop){

self.spriteArr[i].y +=self.speed\* delta\* (-1);

if(self.spriteArr[i].y<= (self.size.height\*self.Scale.y\* (-1))){

self.spriteArr[i].y+= (self.size.height\*self.Scale.y\*self.spriteArr.length)\* (1);

self.callBack();

}

}

}

},

/\*

func:

位置改变——相当于精灵轮播一次结束

\*/

callBack:function(){

var self= this;

self.times\_callback++;

if((self.times!= 0)&&(self.times==self.times\_callback)){

self.times= 0;

self.unscheduleUpdate();

self.callback();

}

/\*

上述if语句相当于if(this.times==self.times\_callback)

因为当this.times为0时，this.times\_callback只要进入该函数，

self.times\_callback的值均不为0。

\*/

}

});

* 调用接口:

**var rollNode= new CRollNode();**

**self.addChild(rollNode, 1, 2);**

**rollNode.Init(res.wave\_png, cc.p(0, cc.winSize.height\* 0.5), cc.p(1, 1), g\_moveDirection.RightToLeft);**

**rollNode.setFormat(1000, 0);**

1. 图片从中间分裂的动画【心碎、孔雀开屏、展翅、开花】

* 代码实现:

/\*

分裂方式

\*/

if(typeof g\_splitWay== "undefined"){

var g\_splitWay= {

topCenter:0,//上中

bottomCenter:1,//下中

leftCenter:2,//左中

rightCenter:3//右中

};

}

/\*

运动方向

\*/

if(typeof g\_directionWay== "undefined"){

var g\_directionWay= {

Single:0,//单方向

Restore:1,//还原

Round:2//往返运动

}

}

/\*

数据配置

anchor——一边

anchor——另一边

c1——系数c1

c2——系数c2

\*/

if(typeof g\_dataIni== "undefined"){

var g\_dataIni= [

[cc.p(1, 0), cc.p(0, 0), -1, 1], //上中

[cc.p(1, 1), cc.p(0, 1), 1, -1],//下中

[cc.p(0, 0), cc.p(0, 1), -1, 1], //左中

[cc.p(1, 0), cc.p(1, 1), 1, -1]//右中

];

}

var g\_side1ID= 0;

var g\_side2ID= 1;

var g\_c1ID= 2;

var g\_c2ID= 3;

var CHeartBroken= cc.Node.extend({

way:null,//分裂方式

direction:null,//运动方向

removeArr:[],//分裂元素数组

totalNum:0,//等分成几块

range:0,//角度张开的最大范围【0-180？】

clipArr:[],//裁切数组

times:0,//动作动作次数

times\_callback:0,//动作回调次数

bCallback:false,//可否回调

callback:null,//回调函数

/\*

@path:要分裂的纹理路径

@way:分裂方式

@direction:运动方向

@num:分裂成几块【确保大于等于2】

\*/

Init:function(path, way, direction, num){

var self= this;

self.removeArr= new Array(0);

self.clipArr= new Array(0);

self.way= way;

self.direction= direction;

self.totalNum= num;

//裁切位置数组

var bEven=((num%2)&&(self.way>=2))?true:false;

self.initArray(bEven);

for(var i=0;i< num; ++i){

var anchorPos= (i<num\* 0.5)?g\_dataIni[self.way][g\_side1ID]:g\_dataIni[self.way][g\_side2ID];

var sprite= new cc.Sprite(path);

sprite.setAnchorPoint(anchorPos);

self.addChild(sprite);

var size= sprite.getContentSize();

var step= size.height/ num;

var rect= cc.rect(0, step\* i, size.width, step);

if(self.way>= 2){

sprite.setPosition(cc.p(size.width\* 0.5\* g\_dataIni[self.way][g\_c1ID], step\* self.clipArr[num- i-1]));

}else{

step= size.width/ num;

rect= cc.rect(step\* i, 0, step, size.height);

sprite.setPosition(cc.p(step\* self.clipArr[i], size.height\* 0.5\* g\_dataIni[self.way][g\_c1ID]));

}

sprite.setTextureRect(rect);

self.removeArr.push(sprite);

}

},

/\*

func:

开始执行分裂动画

@time:单方向动画执行时长

@times:动画执行次数

@range:角度张开的最大范围

\*/

runAnimation:function(time, times, range){

var self= this;

self.times= times;

self.range= range;

var angle= (1/(self.totalNum- 1))\* self.range;

//一边的最大偏转角度

var mostAngle= self.range\* 0.5\*(-1);

for(var i=0;i< self.totalNum; ++i){

var seq= null;

var rotate= cc.rotateTo(time,(mostAngle+ i\* angle)\*(g\_dataIni[self.way][g\_c2ID]));

if(self.direction== g\_directionWay.Round){

//往返方向

var rotate1= cc.rotateTo(time,0);

seq= cc.sequence(rotate1, rotate, cc.callFunc(self.callback\_action));

}else{

//单方向

seq= cc.sequence(rotate, cc.callFunc(self.callback\_action));

}

//往返

if(times== 0){

self.removeArr[i].runAction(seq.repeatForever());

}else{

self.removeArr[i].runAction(seq.repeat(times));

}

}

},

/\*

func:

初始化数组-裁切位置

@bEven:是否为偶数

\*/

initArray:function(bEven){

var self= this;

var half= self.totalNum\* 0.5;

var num= Math.floor(half)\* (-1);

num+= (bEven?0.5:1);

var judgeNum= Math.floor(half);

judgeNum-= (bEven?0:1);

for(var i=0;i< self.totalNum;++i){

self.clipArr.push(num);

if(i== judgeNum) continue;

num++;

}

},

/\*

func:

精灵执行一次动作,之后的回调函数

principle:

如果一直执行动作，每次执行完成之后，回调函数

否则，执行完指定次数的动作之后，才回调函数

\*/

callback\_action:function(pSender){

var self= pSender.getParent();

if(self.direction== g\_directionWay.Restore){

pSender.setRotation(0);

}

if(self.bCallback){

self.times\_callback++;

if(self.times== 0){

if(self.times\_callback%self.totalNum==0){

self.callback();

}

}else{

if(self.times\_callback== self.times\* self.totalNum){

self.callback();

}

}

}

},

/\*

func:

设置是否进行回调函数

@callback:回调函数

\*/

setBCallback:function(Callback){

var self= this;

if(Callback== undefined){

self.bCallback= false;

}else{

self.bCallback= true;

self.callback= callback;

}

}

});

* 接口调用:

**var heart= new CHeartBroken();**

**heart.setPosition(cc.winSize.width\* 0.5, cc.winSize.height\* 0.5);**

**this.addChild(heart);**

**//【初始化】纹理路径- 分裂方式- 往返方向- 被切割的块数**

**heart.Init(res.crown\_png, g\_splitWay.topCenter, g\_directionWay.Round, 10);**

**//【动画】单方向动画执行时长- 执行次数- 角度张开的最大范围**

**heart.runAnimation(2, 0, 40);**

1. 精灵裂成两半:

* 代码实现:

var CSplitNode= cc.Node.extend({

child:[],//分裂成几等分

bCallBack:false,//设置可否回调

callback:null,//回调接口

totalNum:0,//分裂总数

ctor:function(){

this.\_super();

this.bCallBack= false;

this.callback= null;

this.totalNum= 2;

},

/\*

func:

初始化函数

@path:纹理路径

@percent:切割点占图片尺寸的百分比

\*/

Init:function(path, percent){

var sprite1= new CParabolaSprite(path);

var sprite2= new CParabolaSprite(path);

var size= sprite1.getContentSize();

//切线的角度小于正负45度 水平

//切线的角度大于45度 竖直

//竖直切下来

//var percent= Math.random();

var rect1= cc.rect(0, 0, size.width\* percent, size.height);

var rect2= cc.rect(size.width\* percent, 0, size.width\* (1- percent), size.height);

sprite1.setAnchorPoint(cc.p(1, 0.5));

sprite2.setAnchorPoint(cc.p(0, 0.5));

sprite1.setTextureRect(rect1);

sprite2.setTextureRect(rect2);

sprite1.setPosition(cc.p(size.width\* ((1)\* percent- 0.5), 0));

sprite2.setPosition(cc.p(size.width\* ((1)\* percent- 0.5), 0));

//抛物线运动

//当参数4为cc.p(0, 0)时,抛物线的顶点为极为分割点

sprite1.InitWay(20, false, false);

sprite2.InitWay(20, false, false);

//为了确保分割点在切中目标的居中位置，可以设置cc.p(0,0)或者不设置

sprite1.setOffsetPos(cc.p(size.width\* ((1)\* percent- 0.5), 0));

sprite2.setOffsetPos(cc.p(size.width\* ((1)\* percent- 0.5), 0));

//设置每帧偏移量

sprite1.setStep(-500);

sprite2.setStep(500);

//设置回调函数

sprite1.setCallback(this.callBack);

sprite1.setCallback(this.callBack);

this.addChild(sprite1);

this.addChild(sprite2);

},

/\*

func:

设置可否回调

@callback:回调函数

\*/

setCallback:function(callback){

if(callback== undefined){

this.bCallBack= false;

}else{

this.bCallBack= true;

this.callback= callback;

}

},

/\*

func:

回调函数

@self:本身类的对象

\*/

callBack:function(self){

if(self.bCallBack){

self.callback();

}

}

});

1. 切痕节点

* 代码实现:

//切痕的tag

var g\_maxZOrder= 10000;

/\*

说明:

child是二维数组,每一列中的元素为:

0- CScratchNode的子节点

1- 该子节点能否被切中;

\*/

var g\_childID= 0;

var g\_canCutID= 1;

var CScratchNode= cc.Node.extend({

child:[],//cc.Node对象中所有的子节点

scratch:null,//划痕

bShow:false,//划痕是否在显示

size:cc.size(0, 0),

beginPos:cc.p(0, 0),//触摸的起点位置

curPos:cc.p(0,0),//当前点

preSize:cc.size(0, 0),//划痕的尺寸

fadeTime:0,//绘制出来，消失时间【0-不隐藏】

bCollision:false,//是否开启碰撞检测

callback:null,//切中后的回调函数

bRemove:true,//切中后是否隐藏,默认隐藏

percent:0,//起始触摸点在相对于切中目标的水平百分比

ctor:function(){

this.\_super();

this.beginPos= cc.p(0, 0);

this.fadeTime= 0;

this.child= new Array(0);

this.bShow= false;

},

/\*

func:

1、初始化纹理路径;

2、设置划痕有效监听区域的宽和高;

@path:划痕纹理路径

@size:有效的监听区域

\*/

Init:function(path, size){

this.size= size;

//划痕节点

var scratch= new cc.Sprite(path);

this.preSize= scratch.getContentSize();

scratch.setAnchorPoint(cc.p(0, 0.5));//设置锚点

scratch.setScale(0 ,1);//设置尺寸

this.addChild(scratch, g\_maxZOrder, g\_maxZOrder);

this.scratch=scratch;

//添加监听机制

this.addTouch();

},

/\*

func:监听机制

\*/

addTouch:function(){

var self= this;

var listener= cc.EventListener.create({

event:cc.EventListener.TOUCH\_ONE\_BY\_ONE,

swallowTouches:true,

onTouchBegan:self.onTouchBegan,

onTouchMoved:self.onTouchMoved,

onTouchEnded:self.onTouchEnded

});

cc.eventManager.addListener(listener, self);

},

/\*

func:

在有效范围内点击,切痕初始化

\*/

onTouchBegan:function(touch, event){

var target= event.getCurrentTarget();

var rect= cc.rect(target.size.width\* (-0.5), target.size.height\* (-0.5), target.size.width, target.size.height);

var locationPos= target.convertToNodeSpace(touch.getLocation());

if(cc.rectContainsPoint(rect, locationPos)){

target.reset(touch.getLocation());

}

return true;

},

/\*

func:

当触摸点在有效区域中时,改变切痕的缩放系数和旋转角度。

否则,也就是在有效区域外,隐藏。

\*/

onTouchMoved:function(touch, event){

var target= event.getCurrentTarget();

var rect= cc.rect(target.size.width\* (-0.5), target.size.height\* (-0.5), target.size.width, target.size.height);

var locationPos= target.convertToNodeSpace(touch.getLocation());

if(cc.rectContainsPoint(rect, locationPos)){

target.Moving(touch.getLocation());

}else{

target.fade();

}

},

/\*

释放鼠标或者手指时,切痕隐藏。

\*/

onTouchEnded:function(touch, event){

var target= event.getCurrentTarget();

target.fade();

},

/\*

func:

切痕初始化

principle:

1、停止切痕的所有动作;

2、设置切痕的不透明度为255;

3、设置切痕的显示比例;

4、设置切痕的位置;

\*/

reset:function(pos){

this.beginPos= pos;

this.curPos= pos;

this.bShow= true;

this.scratch.stopAllActions();

this.scratch.setScale(0, 1);

this.scratch.setOpacity(255);

this.scratch.setPosition(this.convertToNodeSpace(pos));

},

/\*

func:

修改切痕的缩放系数和旋转角度

principle:

前提:已经在有效区域中，移动

如果切痕已经显示,那么修改切痕的缩放系数和旋转角度;

否则,也就是移动时切痕并没有显示出来【从外部移动到有效区域内】,为了此种情况下

也可以显示切痕添加了this.reset(pos);。如果不需要此种效果,直接将其注释掉即可。

缩放系数X= (起点和终点之间的距离【cc.pDistance(pos, this.beginPos)】)/ 切痕纹理的宽度;

旋转角度angle= 起点和终点之间的角度 转化为 精灵的旋转角度【this.getAngle(this.beginPos, pos)】

@pos:当前触摸点的位置

\*/

Moving:function(pos){

if(!this.bShow){//未显示

this.reset(pos);

return;

}

this.curPos= pos;

//两点之间的距离

var distance= cc.pDistance(pos, this.beginPos);

var XScale= distance/this.preSize.width;

this.scratch.setScaleX(XScale);

//角度

var angle= this.getAngle(this.beginPos, pos);

this.scratch.setRotation(angle);

//碰撞检测

if(this.bCollision){

this.isCollision();

}

},

/\*

func:

根据起点、终点，精灵应该转动的角度

principle:

利用三角函数:cos(angle)= x/ 斜边的长度

获取由起点-->终点的向量:displacement= cc.pSub(endPos, startPos);

斜边的长度= 该向量的长度:length= cc.pLength(displacement);

弧度= Math.acos(displacement.x/ length);

角度= 弧度\* 180/ Math.PI

转化为精灵的旋转角度:angle\*= (displacement.y< 0)?1:(-1);

@startPos:起点

@endPos:终点

\*/

getAngle:function(startPos, endPos){

//位移

var displacement= cc.pSub(endPos, startPos);

var length= cc.pLength(displacement);

var angle= Math.floor(Math.acos(displacement.x/ length)\* 180/ Math.PI);

angle\*= (displacement.y< 0)?1:(-1);

return angle;

},

/\*

func:

设置渐隐时间

@time:划痕渐隐时间- time为0时,不隐藏

\*/

setFadeTime:function(time){

this.fadeTime= (time== undefined)?0:time;

},

/\*

func:

划痕绘制成功以后，渐隐

principle:

如果划痕已经显示&&渐隐时间不为0,

1、划痕隐藏;

2、当前点重置;

\*/

fade:function(){

if(this.fadeTime&&this.bShow){

var fadeOut= cc.fadeOut(this.fadeTime);

this.scratch.runAction(fadeOut);

this.bShow= false;

this.curPos= cc.p(-1, -1);

}

},

/\*

func:

获取划痕当前是否处于显示中

\*/

getBShow:function(){

return this.bShow;

},

/\*

func:

获取当前位置

\*/

getCurPos:function(){

return this.curPos;

},

/\*

func:

设置切中后，是否移除

\*/

setBRemove:function(bRemove){

this.bRemove= (bRemove== undefined)?false:bRemove;

},

/\*

func:

开启碰撞检测的回调接口

@callback:切中后的回调接口

\*/

startCollision:function(callback){

this.bCollision= true;

//回调函数接口

this.callback= callback;

},

/\*

func:

碰撞检测

principle:

判定是否切中有两种方法

方法一:

判定(起点和终点组成的线段)是否和目标矩形相交

方法二:

判定当前点是否在目标矩形中

【缺点:有时切痕在目标上，但是游戏逻辑处理会当成 没有切中】

【情景再现:在目标的一端开始触摸,绕过目标,到目标的另一端】

获取cc.Node节点中的所有所有子节点，并遍历。如果子节点是切痕,continue。

否则,判定是否和子节点是否切中。

\*/

isCollision:function(){

var self= this;

for(var i=0; i< self.child.length;++i){

if(!self.child[i][g\_canCutID]) continue;//不能被切中

var size= self.child[i][g\_childID].getContentSize();

var rect= cc.rect(0, 0, size.width, size.height);

var pos= self.child[i][g\_childID].convertToNodeSpace(self.curPos);

if(cc.rectContainsPoint(rect, pos)){//切中以后，处理

self.child[i][g\_canCutID]= false;

self.calculatePercent(size, self.child[i][g\_childID]);

self.callback(self.percent, cc.pAdd(self.getParent().getPosition(), self.getPosition()), self.child[i][g\_childID].tag);

if(self.bRemove){

//从数组中移除，同时重新遍历当前位置

self.child[i][g\_childID].removeFromParent(true);

i--;

}

}

}

},

/\*

func:

添加到子节点数组中

因为tag一定要有,所以三个参数，均不可能为空。

\*/

addToChild:function(target, zOrder, tag){

var self= this;

self.addChild(target, zOrder, tag);

//二维数组中添加新元素

var arr= new Array(2);

arr[g\_childID]= target;//目标

arr[g\_canCutID]= true;//默认,可以被切中

self.child.push(arr);

},

/\*

func:

计算，在被切中目标左下角点为原点的坐标系中的位置

principle:

获取相对于被切中目标的位置= target.convertToNodeSpace(this.beginPos);

注意:此时被切中目标的锚点是cc.p(0.5, 0.5)，是相对于被切中目标的中心位置。

因此要转换为目标左下角坐标系中,pos= cc.pAdd(pos, cc.p(size.width\* 0.5, size.height\* 0.5));

原理有错误，代码是正解 但是解释不通。

@size:被切中目标的尺寸

@target:被切中的目标

\*/

calculatePercent:function(size, target){

var pos= target.convertToNodeSpace(this.beginPos);

pos= cc.pAdd(pos, cc.p(0, size.height\* 0.5));

this.percent= pos.x/ size.width;

}

});

* 接口调用:

**var self= this;**

**var scratchNode= new CScratchNode();**

**scratchNode.setPosition(cc.p(cc.winSize.width\* 0.5, cc.winSize.height\* 0.5));**

**self.addChild(scratchNode, 1, 2);**

**//子节点**

**var start= new cc.Sprite(res.btn\_play\_png);**

**scratchNode.addToChild(start, 1, 1);**

**//初始化设置**

**scratchNode.Init(res.slideTrack\_png, cc.size(start.getContentSize().width\* 2, start.getContentSize().height\* 2));**

**//绘制成功以后是否要隐藏**

**scratchNode.setFadeTime(1);**

**//开启碰撞检测**

**scratchNode.startCollision(self.callback\_Menu);**

1. 进度条类

该进度条类，使用了缩放和可视区域的两种方法。测试时，此种方法在三星9100和华为两款手机上，不能正常显示。

* 代码实现:

//显示方式

if(typeof g\_way== "undefined"){

var g\_way= {

Scale:0,

VisibleRect:1

};

}

//进度条的方向

if(typeof g\_direction== "undefined"){

var g\_direction={

//水平

H\_LeftToRight:0,

H\_RightToLeft:1,

//竖直

V\_BottomToTop:2,

V\_TopToBottom:3

};

}

/\*

配置数据

锚点- 缩放系数- 位置信息

\*/

var g\_iniProcessBar= [

[cc.p(0, 0.5), cc.p(0, 1), cc.p(0, 0.5)],//从左到右

[cc.p(1, 0.5), cc.p(0, 1), cc.p(1, 0.5)],//从右到左

[cc.p(0.5, 0), cc.p(1, 0), cc.p(0.5, 0)],//从下到上

[cc.p(0.5, 1), cc.p(1, 0), cc.p(0.5, 1)]//从上到下

];

var g\_anchorID= 0;

var g\_scaleID= 1;

var g\_posID= 2;

/\*

func:

进度条类

\*/

var CProcessBar= cc.Node.extend({

way:g\_way.Scale,

direction:g\_direction.H\_LeftToRight,//进度条的方式

processBarUp:null,//上面的进度条

preSize:cc.size(0, 0),//初始宽高尺寸

preRect:cc.rect(0, 0, 0, 0),//原来的可视区域尺寸

percent:0,//进度条的显示进程

bCallback:false,//显示完全以后,是否回调

callback:null,//回调函数

offsetPos: cc.p(0,0),//偏移量

deltaPerPercent:0,//每个percent的增量

/\*

@way:进度条的显示方式

@direction:方向

@backPath:进度条背景纹理

@forthPath:进度条显示纹理

@percent:初始时的当前进度

\*/

Init:function(way, direction, backPath, forthPath, percent){

var self= this;

self.direction= direction;

self.way= way;

//水平-从左到右

var processBarDown= new cc.Sprite(backPath);

self.addChild(processBarDown, 1, 1);

var bgSize= processBarDown.getContentSize();

var processBarUp= new cc.Sprite(forthPath);

self.processBarUp= processBarUp;

processBarDown.addChild(processBarUp, 1, 1);

//获取初始尺寸

self.preSize= processBarUp.getContentSize();

self.preRect= cc.rect(0, 0, 0, 0);

processBarUp.attr({

anchorX:g\_iniProcessBar[self.direction][g\_anchorID].x,

anchorY:g\_iniProcessBar[self.direction][g\_anchorID].y,

x:bgSize.width\* (g\_iniProcessBar[self.direction][g\_posID].x+ 0.013),

y:bgSize.height\* (g\_iniProcessBar[self.direction][g\_posID].y)

});

if(self.way== g\_way.Scale){

processBarUp.setScale(g\_iniProcessBar[self.direction][g\_scaleID].x, g\_iniProcessBar[self.direction][g\_scaleID].y);

}else{

processBarUp.setTextureRect(self.preRect);

}

if(percent== undefined){

self.percent= 0;

}else{

self.percent= percent;

self.updateProcess();

}

},

/\*

func:

设置是否回调

@callback:回调函数

\*/

setCallback:function(callback){

var self= this;

if(callback== undefined){

self.bCallback= false;

}else{

self.bCallback= true;

self.callback= callback;

}

},

/\*

func:

设置前面的进度条,相对于后面的进度条的位置

@pos:相对位置

\*/

setOffsetPos:function(pos){

var self= this;

self.offsetPos= (pos==undefined)?cc.p(0, 0):pos;

self.processBarUp.setPosition(cc.pAdd(self.processBarUp.getPosition(), self.offsetPos));

},

/\*

func:

获取当前进度

\*/

getPercent:function(){

var self= this;

return self.percent;

},

/\*

func:

设置当前进度

@percent:当前进度

\*/

setPercent:function(percent){

var self= this;

self.percent= percent;

if(self.percent>= 100){

self.percent= 100;

self.Over();

}else if(self.percent<= 0){

self.percent= 0;

self.Over();

}

self.updateProcess();

},

/\*

func:

进度为100%时，回调

\*/

Over:function(){

var self= this;

if(self.bCallback){

self.callback(self.getParent());

}

},

/\*

func:

更新进度条的显示比例

\*/

updateProcess:function(){

var self= this;

if(self.way== g\_way.Scale){

if(self.direction>= 2){

self.processBarUp.setScaleY(0.01\* self.percent\* self.getScaleY());

}else{

self.processBarUp.setScaleX(0.01\* self.percent\* self.getScaleX());

}

}else{

if(self.direction== g\_direction.H\_LeftToRight){

self.preRect.x= self.preSize.width\* self.percent\* 0.01\* (0);

self.preRect.y= 0;

self.preRect.width= self.preSize.width\* self.percent\* 0.01;

self.preRect.height= self.preSize.height;

self.processBarUp.x= 0;

}else if(self.direction== g\_direction.H\_RightToLeft){

self.preRect.x= self.preSize.width\* self.percent\* 0.01\* (1);

self.preRect.y= 0;

self.preRect.width= self.preSize.width;

self.preRect.height= self.preSize.height;

self.processBarUp.x= self.preSize.width\* (self.percent\* 0.01+ 1);

}else if(self.direction== g\_direction.V\_TopToBottom){

self.preRect.x= 0;

self.preRect.y= 0;

self.preRect.width= self.preSize.width;

self.preRect.height= self.preSize.height\* (1- self.percent\* 0.01);

self.processBarUp.y= self.preSize.height;

}else if(self.direction== g\_direction.V\_BottomToTop){

self.preRect.x= 0;

self.preRect.y= self.preSize.height\* (self.percent\* 0.01);

self.preRect.width= self.preSize.width;

self.preRect.height= self.preSize.height\* (1-self.percent\* 0.01);

self.processBarUp.y= 0;

}

self.processBarUp.setTextureRect(self.preRect);

}

}

});

* 接口调用:

**initProcessBar:function(){**

**var self= this;**

**var processBar= new CProcessBar();**

**processBar.Init(g\_way.VisibleRect, g\_direction.H\_LeftToRight, res.loadProcessDown\_png, res.loadProcessUp\_png, 0);**

**processBar.setOffsetPos(cc.p(-3, 0));**

**processBar.setCallback(self.GameOver);**

**processBar.attr({**

**x:cc.winSize.width\* 0.5,**

**y:cc.winSize.height\* 0.9**

**});**

**self.addChild(processBar, 1, 1);**

**var item= [processBar, 1];**

**self.arrProcessBar.push(item);**

**}**

1. 游戏类-拼图

* 代码实现:

/\*

func:

晃动算法

principle:

一次晃动需要三个过程:

1、从中心位置-->一个方向，移动step距离;

2、向另一个方向，移动step\* 2的距离;

3、当前位置-->中心位置，移动step距离;

@direction:晃动方向——【水平true,还是竖直false】

@step:单方向偏移距离【正负符号不同时,初始运动方向不同】

@time:单方向偏移时间

@times:晃动次数

@target:要执行晃动的目标

\*/

function shake(direction, step, time, times, target){

var pos= direction?cc.p(step, 0):cc.p(0, step);

var move1= cc.moveBy(time, pos);//步骤1

var move3= move1.clone();//步骤3

pos= cc.pMult(pos, -2);

var move2= cc.moveBy(time\* 2, pos);

var seq= seq= cc.sequence(move1, move2, move3);

target.runAction(seq.repeat(times));

}

/\*

说明:

拼图游戏

interface:

initClipping:function(){

var clipNode= new CClipNode();

clipNode.attr({

x:cc.winSize.width\* 0.5,

y:cc.winSize.height\* 0.5

});

clipNode.Init(res.bg\_clipping\_png, 3, 3, 10);

clipNode.SetShakeFormat(10, 0.02, 5);

//添加监听机制,开始游戏

clipNode.startGame();

this.addChild(clipNode, 2, 2);

}

\*/

var CClipNode= cc.Node.extend({

size:cc.size(0, 0),//要切的图片的尺寸

leftTopPos:cc.p(0,0),//左上角点

clipWidth:0,//小块的宽

clipHeight:0,//小块的高

row:0,//行数

col:0,//列数

hStep:0,//水平方向上的间隔

vStep:0,//竖直方向上的间隔

arrItems:[],//拼图数组

bSelected:false,//是否被选中

bShake:false,//被选中时, 是否要晃动。默认是不晃动

step:0,//单方向移动频率

time:0,//单方向时间

times:0,//晃动次数

bCallback:false,//是否回调

callback:null,//回调函数

/\*

func:

初始化要裁剪的各种块

@path:裁剪纹理;

@row:裁剪行数

@col:裁剪列数

@hStep:水平方向上的间隔

@vStep:竖直方向上的间隔

\*/

Init:function(path, row, col, hStep, vStep){

var self= this;

self.arrItems= new Array(0);

//行数和列数

self.row= row;

self.col= col;

//水平和竖直间隔

self.hStep= hStep;

self.vStep= (vStep== undefined)? self.hStep: vStep;

//创建图块数组

self.createItemArray(path);

},

/\*

func:

设置晃动格式数据

@step:单方向移动频率

@time:单方向时间

@times:晃动次数

\*/

SetShakeFormat:function(step, time, times){

var self= this;

self.bShake= true;

self.step= step;

self.time= time;

self.times= times;

},

/\*

func:

设置游戏结束时是否回调

@callback:回调函数

\*/

SetCallback:function(callback){

var self= this;

if(callback== undefined){

self.bCallback= false;

}else{

self.bCallback= true;

self.callback= callback;

}

},

/\*

func:

一维数组，存储裁剪后的图块

@path:纹理的路径信息

\*/

createItemArray:function(path){

var self= this;

//原图

var texture= cc.textureCache.addImage(path);

//原图尺寸

self.size= texture.getContentSize();

//要裁剪的小块的尺寸

self.clipWidth= Math.floor(self.size.width/ self.col);

self.clipHeight= Math.floor(self.size.height/ self.row);

//该剪切区域所占的总尺寸

var size= cc.size(self.size.width+ self.hStep\*(self.col- 1), self.size.height+ self.vStep\* (self.row- 1));

//左上角区域

self.leftTopPos= cc.p(- 0.5\* size.width, 0.5\* size.height);

//设置裁剪区域

var rect= cc.rect(0, 0, 0, 0);

//在原图上,按照要求裁剪

for(var i= 0; i< self.row; ++i){//行数

for(var j= 0; j< self.col; ++j){//列数

//在原图中,裁剪位置

var pos= cc.p(j\* self.clipWidth, i\* self.clipHeight);

rect= cc.rect(pos.x, pos.y, self.clipWidth, self.clipHeight);

var sprite= new cc.Sprite();

sprite.setAnchorPoint(cc.p(0, 1));

sprite.setTexture(texture);

sprite.setTextureRect(rect);

self.addChild(sprite, 1, i\* self.col+ j);

self.arrItems.push(sprite);

}

}

self.chaoticArray();

},

/\*

func:

混乱数组,避免出现天糊

\*/

chaoticArray:function(){

var self= this;

//此种方法可以避免出现天糊

do{

//打乱原始数组

self.arrItems.sort(function(){return Math.random()<0.5?-1:1});

}while(self.bGameOver());

self.setItemPos();

},

/\*

func:

设置图块的位置

\*/

setItemPos:function(){

var self= this;

for(var i= 0;i< self.row; ++i){//行数

for(var j= 0;j< self.col; ++j){//列数

var pos= self.calculatePos\_ByRowCol(i, j);

self.arrItems[i\* self.col+ j].setPosition(self.calculatePos\_ByRowCol(i, j)) ;

}

}

},

startGame:function(){

var self= this;

for(var i= 0;i< self.row; ++i){//行数

for(var j= 0;j< self.col; ++j){//列数

self.addTouch(self.arrItems[i\* self.col+ j]);

}

}

},

/\*

func:

计算图块的位置

@row:图块所属的行

@col:图块所属的列

\*/

calculatePos\_ByRowCol:function(row, col){

var self= this;

return cc.pAdd(cc.p(col\* (self.clipWidth+ self.hStep), -row\*(self.clipHeight + self.vStep)), self.leftTopPos);

},

/\*

func:

计算图块的位置

@id:数组removeArr中的下标

\*/

calculatePos\_ByID:function(id){

var self= this;

//目标所在的行数和列数

var row= Math.floor(id/self.col);

var col= id%self.col;

return cc.pAdd(cc.p(col\* (self.clipWidth+ self.hStep), -row\*(self.clipHeight + self.vStep)), self.leftTopPos);

},

//监听机制

addTouch:function(target){

var self= this;

var listener= cc.EventListener.create({

event:cc.EventListener.TOUCH\_ONE\_BY\_ONE,

swallowTouches:true,

onTouchBegan:self.onTouchBegan,

onTouchMoved:self.onTouchMoved,

onTouchEnded:self.onTouchEnded

});

cc.eventManager.addListener(listener, target);

},

onTouchBegan:function(touch, event){

var target= event.getCurrentTarget();

var size= target.getContentSize();

var rect= cc.rect(0, 0, size.width\* target.getScale(), size.height\* target.getScale());

var locationPos= target.convertToNodeSpace(touch.getLocation());

if((false==target.getParent().bSelected)&&(cc.rectContainsPoint(rect, locationPos))){

target.getParent().onSelected(target);

return true;

}else{

return false;

}

},

onTouchMoved:function(touch, event){

var target= event.getCurrentTarget();

var self= target.getParent();

self.moving(touch.getDelta(), target);

},

onTouchEnded:function(touch, event){

var target= event.getCurrentTarget();

var self= target.getParent();

target.stopAllActions();

self.reset(touch.getLocation(), target);

},

/\*

func:

选中目标,晃动算法

@target:执行目标

\*/

onSelected:function(target){

var self= this;

self.bSelected= true;

target.setLocalZOrder(target.getLocalZOrder()+ 1);

//是否可以被选中

if(self.bShake){

shake(true, self.step, self.time, self.times, target);

}

},

/\*

func:

实现拖动图块,移动的效果

@delta:相对位移

@target:被选中的目标

\*/

moving:function(delta, target){

//避免缩放造成的移动间距

delta= cc.pMult(delta, 1/(target.getScale()\* target.getParent().getScale()));

target.setPosition(cc.pAdd(target.getPosition(), delta));

},

/\*

func:

1、还原目标的ZOrder;

2、交换图片位置;

3、判定游戏是否结束;

@pos:释放鼠标时的位置，也就是要移动到的位置

@target:被选中的目标

\*/

reset:function(pos, target){

var self= this;

self.bSelected= false;

target.setLocalZOrder(1);

//选中的目标，在数组中的下标

var tag= target.tag;

for(var i=0;i< self.arrItems.length; ++i){

if(self.arrItems[i].tag== tag){

tag= i;

break;

}

}

self.changePos(tag, self.JudgeTag(tag, pos));

//判定:游戏是否结束

if(self.bGameOver()){

self.GameOver();

}

},

/\*

func:

判定移动到那个位置,如果移动到无效区域,就返回自己的ID

@tag:被选中的目标,在数组中的下标位置

@pos:释放鼠标时的位置，也就是要移动到的位置

\*/

JudgeTag:function(tag, pos){

var self= this;

var id= tag;

var rect= cc.rect(0, 0, self.clipWidth, self.clipHeight);

for(var i=0;i< self.arrItems.length; ++i){

if(i== id) continue;

var locationPos= self.arrItems[i].convertToNodeSpace(pos);

if(cc.rectContainsPoint(rect, locationPos)){

id= i;

break;

}

}

return id;

},

/\*

func:

原始位置上的图块和目标位置上的图块 交换位置

@srcID:初始目标的ID

@desID:终点目标的ID

\*/

changePos:function(srcID, desID){

var self= this;

self.arrItems[srcID].setPosition(self.calculatePos\_ByID(desID));

if(desID!= srcID){

self.arrItems[desID].setPosition(self.calculatePos\_ByID(srcID));

//同时调换数组顺序

var temp= self.arrItems[srcID];

self.arrItems[srcID]= self.arrItems[desID];

self.arrItems[desID]= temp;

}

},

/\*

func:

游戏是否结束

\*/

bGameOver:function(){

var self= this;

for(var i=0; i<self.arrItems.length; ++i){

if(i!= self.arrItems[i].tag){

return false;

}

}

return true;

},

/\*

func:

游戏结束

\*/

GameOver:function(){

var self= this;

alert("Game Over!");

if(self.bCallback){

self.callback(self);

}

}

});

* 接口调用:

**initClipping:function(){**

**var clipNode= new CClipNode();**

**clipNode.attr({**

**x:cc.winSize.width\* 0.5,**

**y:cc.winSize.height\* 0.5**

**});**

**clipNode.Init(res.bg\_clipping\_png, 3, 3, 10);**

**clipNode.SetShakeFormat(10, 0.02, 5);**

**//添加监听机制,开始游戏**

**clipNode.startGame();**

**this.addChild(clipNode, 2, 2);**

**}**

1. 游戏类—寻找房祖名

* 代码实现:

/\*

func:

图块的对应的tag

\*/

if(typeof g\_picTag== "undefined"){

var g\_picTag={

correct:1,//正确的

wrong:2//错误的

};

}

var CFindNode= cc.Node.extend({

arrItems:null,//二维数组

centerPos:null,//放置位置【中间点的位置】

leftTopPos:null,//图块的起始位置【左上角的位置】

leftBottomPos:null,//背景层的位置【左下角】

numPerRow:0,//每一行的图块数目

numPerCol:0,//每一行的图块数目

correctPicPath:null,//正确图片的纹理路径

wrongPicPath:null,//错误的纹理路径

cScaleX:1,//初始缩放比例

cScaleY:1,

visibleSize:null,//图块要显示的区域大小

sizeItem:cc.size(0, 0),//图块的尺寸，

bCallback:false,//选中后，是否需要回调

callback:null,//选中后的回调函数,

times\_callback:0,//回调的次数

/\*

func:

初始化

@numPerRow:每一行或者每一列中图块数

@pos:放置位置，锚点为cc.p(0.5, 0.5)

@rect:区域范围

@correctPicPath:正确图片的纹理路径

@wrongPicPath:错误的纹理路径

\*/

Init:function(numPerRow, pos, size, correctPicPath, wrongPicPath){

var self= this;

self.visibleSize= size;

self.arrItems= new Array(0);

self.numPerRow= numPerRow;

self.numPerCol= numPerRow;

self.correctPicPath= correctPicPath;

self.wrongPicPath= wrongPicPath;

self.centerPos= pos;

var texture= cc.textureCache.addImage(self.correctPicPath);

self.sizeItem= texture.getContentSize();

self.addBgColor();

self.addItems();

},

/\*

func:

下一关

@delta:图块的增量 默认为1

\*/

changeNextCensorship:function(delta){

var self= this;

self.removeAllChildren();

if(delta== undefined){

self.numPerRow++;

self.numPerCol++;

}else{

self.numPerRow+= delta;

self.numPerCol+= delta;

}

self.addBgColor();

self.addItems();

self.startGame();

},

/\*

func:

底层色

\*/

addBgColor:function(){

var self= this;

self.calculateLeftBottomPos();

var colorLayer= new cc.LayerColor();

colorLayer.attr({

x:self.leftBottomPos.x,

y:self.leftBottomPos.y,

width:self.visibleSize.width,

height:self.visibleSize.height,

color:self.calculateBgColor()

});

self.addChild(colorLayer, 0, 0);

},

/\*

func:

计算背景的随机颜色

\*/

calculateBgColor:function(){

var r= Math.floor(Math.random()\* 255);

var g= Math.floor(Math.random()\* 255);

var b= Math.floor(Math.random()\* 255);

return cc.color(r, g, b);

},

/\*

func:

监听机制

\*/

addListener:function(target){

var self= this;

var listener= cc.EventListener.create({

event:cc.EventListener.TOUCH\_ONE\_BY\_ONE,

swallowTouches:true,

onTouchBegan:self.onTouchBegan

});

cc.eventManager.addListener(listener, target)

},

onTouchBegan:function(touch, event){

var target= event.getCurrentTarget();

var pos= touch.getLocation();

var self= target.getParent();

if(self.collisionDetection(pos, target)){

self.onSelected((target.tag== g\_picTag.correct), target);

}

return false;

},

/\*

func:

某点是否在某个目标的矩形区域内

@pos:世界坐标系中的位置

@target:目标

\*/

collisionDetection:function(pos, target){

var self= this;

//相对坐标系中的位置

var locationPos= target.convertToNodeSpace(pos);

var rect= cc.rect(0, 0, self.sizeItem.width, self.sizeItem.height);

return cc.rectContainsPoint(rect, locationPos);

},

/\*

func:

添加小图快

\*/

addItems:function(){

var self= this;

self.calculateScale();

self.calculateLeftTopPos();

self.arrItems= new Array(self.numPerRow\* self.numPerCol);

//正确的

self.arrItems[0]= (self.createItem(g\_picTag.correct));

for(var i= 1;i< self.arrItems.length;++ i){

self.arrItems[i]= (self.createItem(g\_picTag.wrong));

}

self.showItems();

},

/\*

func:

混乱数组，并且添加到该节点下

\*/

showItems:function(){

var self= this;

self.arrItems.sort(function(){return Math.random()< 0.5?1:-1;});

for(var i=0;i< self.arrItems.length; ++i){

self.setItemPos(i);

self.addChild(self.arrItems[i], 1);

}

},

/\*

func:

计算图块的缩放比例

\*/

calculateScale:function(){

var self= this;

self.cScaleX= self.visibleSize.width/ (self.numPerCol\* self.sizeItem.width);

self.cScaleY= self.visibleSize.height/ (self.numPerRow\* self.sizeItem.height);

},

/\*

func:

计算:最左上角图块的位置

\*/

calculateLeftTopPos:function(){

var self= this;

var size= self.sizeItem;

var totalSize= cc.size(self.cScaleX\* size.width\* self.numPerCol, self.cScaleY\* size.height\* self.numPerRow);

//图块组成的二维数组,组成的矩形区域中,左上角位置

self.leftTopPos= cc.p(self.centerPos.x- totalSize.width\* 0.5, self.centerPos.y+ totalSize.height\* 0.5);

//图块组成的二维数组，下标为0的图块的位置

self.leftTopPos= cc.pAdd(self.leftTopPos, cc.p(self.cScaleX\* size.width\* 0.5, self.cScaleX\* size.height\* (-0.5)));

},

/\*

func:

计算:可视区域的左下角的位置

\*/

calculateLeftBottomPos:function(){

var self= this;

//图块组成的二维数组

self.leftBottomPos= cc.pSub(self.centerPos, cc.p(self.visibleSize.width\* 0.5, self.visibleSize.height\* 0.5));

},

/\*

func:

创建单个的图块

@tag:图块的tag

\*/

createItem:function(tag){

var self= this;

var path= ((tag==g\_picTag.wrong)?self.wrongPicPath:self.correctPicPath);

var item= new cc.Sprite(path);

item.setScale(self.cScaleX, self.cScaleY);

item.tag= tag;

return item;

},

/\*

func:

根据该图块在二维数组中的行号和列号，设置该图块的位置

\*/

setItemPos:function(id){

var self= this;

var row= Math.floor(id/ self.numPerCol);

var col= Math.floor(id% self.numPerCol);

self.arrItems[id].setPosition(cc.pAdd(self.leftTopPos, cc.p(self.cScaleX\* self.sizeItem.width\* col, self.cScaleX\* self.sizeItem.height\* row\* (-1))));

},

/\*

func:

开始游戏

\*/

startGame:function(){

var self= this;

for(var i= 0;i< self.arrItems.length;++ i){

self.addListener(self.arrItems[i]);

}

},

setCallback:function(callback){

var self= this;

if(callback== undefined){

self.bCallback= false;

}else{

self.bCallback= true;

self.callback= callback;

}

},

/\*

func:选择结果

@result:结果

\*/

onSelected:function(result, target){

var self= this;

if(!self.bCallback){//如果没有回调函数

self.changeNextCensorship();

}else if(result){//有回调函数&&选中

self.times\_callback= 0;

for(var i=0;i< self.arrItems.length; ++i){

var c= Math.floor(Math.random()\*2)?1:(-1);

Shake(1, 20\* c, 0.1, 2, self.arrItems[i], self.action\_callback);

}

}else{

//有回调函数但是没有被选中

Shake(1, 20, 0.1, 2, target);

}

},

action\_callback:function(pSender){

var self= pSender.getParent();

self.times\_callback++;

if(self.times\_callback== self.arrItems.length){

self.times\_callback= 0;

self.callback(self);

}

}

});

/\*

func:目标执行晃动效果

principle:一次晃动需要三个过程:

1、从中心位置-->一个方向，移动step距离;

2、向另一个方向，移动step\* 2的距离;

3、当前位置-->中心位置，移动step距离;

@direction:晃动方向——【水平true,还是竖直false】

@step:单方向偏移距离【正负符号不同时,初始运动方向不同】

@time:单方向偏移时间

@times:晃动次数

@target:要执行晃动的目标

@callback:回调函数

\*/

function Shake(direction, step, time, times, target, callback){

var pos= direction?cc.p(step, 0):cc.p(0, step);

var move1= cc.moveBy(time, pos);//步骤1

var move3= move1.clone();//步骤3

pos= cc.pMult(pos, -2);

var move2= cc.moveBy(time\* 2, pos);

var seq= cc.sequence(move1, move2, move3, cc.callFunc(callback));

target.runAction(seq.repeat(times));

}

* 接口调用:

**addItems:function(){**

**var self= this;**

**var findNode= new CFindNode();**

**findNode.Init(3, cc.p(cc.winSize.width\* 0.5, cc.winSize.height\* 0.475), cc.size(580, 580), res.success\_png, res.fail\_png);**

**self.addChild(findNode, 1, 3);**

**findNode.setCallback(self.onSelectedSuccess);**

**findNode.startGame();**

**},**

**//被选中后的回调函数**

**onSelectedSuccess:function(target){**

**target.changeNextCensorship();**

**}**

1. 游戏类-答题

* 实现:

/\*

问题数组中,

0-题干

1-...选项

\*/

var g\_questionID= 0;

var g\_optionID= 1;

/\*

文本对齐方式

\*/

if(typeof g\_alignWay== "undefined"){

var g\_alignWay= {

Left:0,//左对齐

Center:1//居中

};

}

var CQuestion= cc.Node.extend({

nodeWidth:0,//节点宽度

nodeHeight:0,//节点高度

arrQuestion:[],//题目数组

alignWay:g\_alignWay.Center,//文本对齐方式，默认为左对齐

optionNum:0,//选项的数目

questionPos:null,//题干位置

optionPos:null,//选项中间位置

questionColor:0,//题干的颜色

optionColor:0,//选项的颜色

questionFontSize:0,//题干的字号

optionFontSize:0,//选项的字号

callback:null,//选中之后是否回调

bDetection:false,//是否开启检测

/\*

@alignWay:对齐方式

@nodeWidth:设置节点宽度【文案超过时，自动换行】

@nodeHeight:设置节点高度【文本自动平分】

@id:题目的起始位置

@arrQuestion:问题数组

\*/

Init:function(alignWay, nodeWidth, nodeHeight, id, arrQuestion){

var self= this;

self.alignWay= alignWay;

self.arrQuestion= new Array(0);

self.nodeWidth= nodeWidth;

self.nodeHeight= nodeHeight;

//数组拷贝-[仅拷贝题干和选项]

self.arrQuestion= arrQuestion.slice(id, arrQuestion.length);

self.optionNum= (self.arrQuestion.length- 1);

self.addQuestion();

self.addOptions();

},

/\*

func:切换题目

@id:题目的起始位置

@arrQuestion:题目数组

@delay:切换的延时时间

\*/

switchQuestion:function(id, arrQuestion, delay){

var self= this;

self.arrQuestion= arrQuestion.slice(id, arrQuestion.length);

self.optionNum= (self.arrQuestion.length- 1);

self.scheduleOnce(function(){

self.bDetection= true;

self.removeAllChildren();

self.addQuestion();

self.addOptions();

self.resetQuestionFormat();

self.resetOptionFormat();

},(delay==undefined||delay<=0)?1:delay);

},

/\*

func:

设置题干的格式

@questionPos:题目的位置

@questionColor:题目的颜色

@questionFontSize:题干的字号

\*/

setQuestionFormat:function(questionPos, questionColor, questionFontSize){

var self= this;

self.questionPos= questionPos;

self.questionColor= questionColor;

self.questionFontSize= questionFontSize;

self.resetQuestionFormat();

},

/\*

func:

设置选项的格式

@optionPos:选项的中间位置

@optionColor:选项的颜色

@optionFontSize:选项的字号

\*/

setOptionFormat:function(optionPos, optionColor, optionFontSize){

var self= this;

self.optionPos= optionPos;

self.optionColor= optionColor;

self.optionFontSize= optionFontSize;

self.resetOptionFormat();

},

/\*

func:

设置检测机制开启Or关闭

@bDetection:开启或者关闭

\*/

setDetection:function(bDetection){

var self= this;

self.bDetection= (bDetection== undefined)?false:bDetection;

},

/\*

func:

重新设置题干

\*/

resetQuestionFormat:function(){

var self= this;

var question= self.getChildByTag(g\_questionID);

question.setPosition(self.questionPos);

question.setColor(self.questionColor);

question.setFontSize(self.questionFontSize);

AutoLinefeed(self.nodeWidth, question);

},

/\*

func:

重新设置选项

\*/

resetOptionFormat:function(){

var self= this;

//获取该字体的高度

var temp= new cc.LabelTTF(self.optionFontSize);

var height= temp.height\* self.optionNum;

//高度

var heightPer= (self.nodeHeight- height)/ self.optionNum;

var arrSeq= calculateArray(self.optionNum);

for(var i= 0; i< self.optionNum; ++i){

var desPos= cc.pAdd(self.optionPos, cc.p(0, heightPer\* arrSeq[i]));

var option= self.getChildByTag(g\_optionID+ i);

option.setPosition(desPos);

option.setColor(self.optionColor);

option.setFontSize(self.optionFontSize);

AutoLinefeed(self.nodeWidth\* 0.75, option);

}

},

//add the question

addQuestion:function(){

var self= this;

var question= new cc.LabelTTF("", "微软雅黑", 30);

self.addChild(question, 1, g\_questionID);

question.setString(self.arrQuestion[g\_questionID]);

},

//add the options

addOptions:function(){

var self= this;

for(var i=0; i< self.optionNum; ++i){

var option= new cc.LabelTTF("","微软雅黑", 30);

self.addChild(option, 1, g\_optionID+ i);

option.setString(self.arrQuestion[g\_optionID+ i]);

self.addRadioButton(cc.p(self.nodeWidth\* 0.75+ 30, 0), option);

if(self.alignWay== g\_alignWay.Left){

option.setAnchorPoint(cc.p(0, 0.5));

}

}

},

//单选框

addRadioButton:function(offset, target){

var self= this;

var radioButton= new cc.Sprite(res.radio\_noSelected);

radioButton.setPosition(cc.pAdd(offset, cc.p(0, radioButton.getContentSize().height\* 0.5+ 5)));

target.addChild(radioButton, 1, 1);

self.addTouch(radioButton);

},

//监听机制

addTouch:function(target){

var self= this;

var listener= cc.EventListener.create({

event:cc.EventListener.TOUCH\_ONE\_BY\_ONE,

swallowTouches:true,

onTouchBegan:self.onTouchBegan

});

cc.eventManager.addListener(listener, target);

},

//开始触摸

onTouchBegan:function(touch, event){

var target= event.getCurrentTarget();

var size= target.getContentSize();

var locationPos= target.convertToNodeSpace(touch.getLocation());

var rect= cc.rect(0, 0, size.width, size.height);

var option= target.getParent();

var self= option.getParent();

if((self.bDetection)&&(cc.rectContainsPoint(rect, locationPos))){

self.bDetection= false;

self.onSelected(option.tag, target);

}

return false;

},

/\*

func:

选中

@tag:被选中的选项索引

@pSender:对应的选项

\*/

onSelected:function(tag, pSender){

var self=this;

self.resetAllRadio();

pSender.setTexture(cc.textureCache.addImage(res.radio\_Selected));

if(self.callback!= null){

self.callback(tag);

}

},

//还原所有的图片

resetAllRadio:function(){

var self= this;

for(var i= 0; i< self.optionNum; ++i){

var option= self.getChildByTag(g\_optionID+ i);

var radio= option.getChildByTag(1);

radio.setTexture(cc.textureCache.addImage(res.radio\_noSelected));

}

},

/\*选中之后的回调函数\*/

setCallback:function(callback){

var self= this;

self.callback= (callback== undefined)?null:callback;

}

});

/\*

func:

生成此序列数组

0

1 -1

1 0 -1

2 1 -1 -2

2 1 0 -1 2

principle:

如果在偶数一行中的中间位置添加0 后一个数始终比前一个数小1

每一列的起始数值均为:Math.floor(行号\* 0.5)

\*/

function calculateArray(num){

var arr= new Array(0);

var totalNum= num%2?num:(num+ 1);

var began= Math.floor(totalNum\* 0.5);

for(var i= 0;i< totalNum; ++i){

arr.push(began);

began--;

}

if(num%2== 0){//偶数

arr.splice(Math.floor(totalNum\* 0.5), 1);

for(i=0; i<num; ++i){

arr[i]+= ((arr[i]>0)?(-0.5):0.5);

}

}

return arr;

}

/\*

func:

长字符串——自动换行

principle；

获取字体对象的宽度。超出的部分添加换行符"\r\n"。

@limitWidth:限制文本的显示区域宽度

@target:字体对象

\*/

function AutoLinefeed(limitWidth,target){

var str= target.getString();//字体对象中的文本

var width= target.width;//获取文本的总宽度

var singleWidth= Math.ceil(width/ str.length);//单个字符的宽度

var numPreRow= Math.floor(limitWidth/ singleWidth);//每行中的字符数目

var rows= Math.ceil(width/(singleWidth\* numPreRow));//需要几行，才可以将所有字符， 按照要求显示完全

var temp= str;

str= "";

for(var i=0;i< rows; ++i){

if(i== (rows- 1)){

str+= temp.substr((i\* numPreRow), (temp.length- i\*numPreRow));

}else{

str+= temp.substr((i\* numPreRow), numPreRow)+ "\r\n";

}

}

target.setString(str);

}

* 接口调用:

**addQuestion:function(){**

**var self= this;**

**var question= new CQuestion();**

**g\_iSelectedID= 0; //被选中的题目索引**

**//对齐方式—设置节点宽度【文案超过时，自动换行】—设置节点高度【文本自动平分】—题目的起始位置—问题数组**

**question.Init(g\_alignWay.Left, 500, 300, 1, g\_arrQuestion[g\_iSelectedID]);**

**//题目的位置—题目的颜色—题干的字号**

**question.setQuestionFormat(cc.p(cc.winSize.width\* 0.5, cc.winSize.height\* 0.74), cc.color(0, 0, 0), 40);**

**//选项的位置—选项的颜色—选项的字号**

**question.setOptionFormat(cc.p(cc.winSize.width\* 0.15, cc.winSize.height\* 0.5), cc.color(0, 0,0), 30);**

**//选中时的回调函数**

**question.setCallback(self.selected\_callback);**

**self.addChild(question, 1, 0.1);**

**//开启检测机制**

**question.setDetection(true);**

**},**

**//选中以后的回调函数**

**selected\_callback:function(tag){**

**var pSender= this;**

**g\_iSelectID= (tag- 1);**

**g\_bCorrect= ((g\_iSelectID== g\_arrQuestion[g\_iSelectedID][0])?1:0);**

**g\_iSelectedID++;**

**//题目的起始位置—题目数组—切换的延时时间**

**pSender.switchQuestion(1, g\_arrQuestion[g\_iSelectedID], 1);**

**}**

1. 手势操作类:

* 功能:

**手势操作-单指拖动,双指放大、缩小,双指旋转。**

* 原理:

**单指移动精灵**

**精灵要移动的目标位置= 当前位置+ 触摸移动的偏移量**

**双指操作-【旋转、缩小、放大】**

**1、获取前两个触摸点在目标坐标系统中的位置;**

**2、计算:**

**两点之间的**

**距离= ((x2- x1)\*(x2- x1)+ (y2- y1)\*(y2- y1))的开方**

**cos值= (y2- y1)/(距离)**

**弧度值= (Math.acos(cos值))**

**角度= 弧度值\* 180/ Math.PI;**

**注意:**

**当两个触摸点处于同一竖直线时，角度值可能为NaN。**

**判定角度值是否是NaN，不能使用==NaN，而应该使用函数isNaN(value)**

**3、如果self.initDistance的值为0，表示从未移动过，可以将对应值设置为初始值;**

**4、否则，执行缩放、旋转操作**

**缩放系数= （当前两点的距离）/初始距离\* 精灵初始缩放系数;**

**旋转角度= （当前两点的角度- 两点之间的初始角度）;**

**精灵角度= （旋转角度+ 精灵的初始角度）%360;**

* 实现:

var CGestureSprite= cc.Sprite.extend({

initDistance:0,//两个触摸点之间的初始距离

initAngle:0,//两个触摸点之间的初始角度

Angle:0,//精灵的初始角度

initScale:0,//精灵的初始缩放系数

callback\_Moving:null,//移动时的回调函数

callback\_End:null,//触摸结束时的回调函数

//派生类初始化函数

Init:function(){

var self= this;

self.callback\_Moving= null;

self.callback\_End= null;

self.addListener();

},

//添加多点触摸机制

addListener:function(){

var self= this;

var listener= cc.EventListener.create({

event:cc.EventListener.TOUCH\_ALL\_AT\_ONCE,

onTouchesBegan:self.onTouchesBegan,

onTouchesMoved:self.onTouchesMoved,

onTouchesEnded:self.onTouchesEnded

});

cc.eventManager.addListener(listener, self);

},

//func:同一时间点，触摸点的个数

onTouchesBegan:function(touches, event) {

var target = event.getCurrentTarget();

var size= target.getContentSize();

var rect= cc.rect(0, 0, size.width, size.height);

for(var i=0;i< touches.length;++i){

var locationPos= target.convertToNodeSpace(touches[i].getLocation());

if(!cc.rectContainsPoint(rect, locationPos)){

touches.splice(i, 1);

}

}

},

//func:同一时间点,多个触摸点移动

onTouchesMoved:function(touches, event) {

var target = event.getCurrentTarget();

var size= target.getContentSize();

var rect= cc.rect(0, 0, size.width, size.height);

var arrTemp=new Array(touches.length);

var num= ((touches.length>2)?2:touches.length);

for(var i=0;i< num;++i){

var locationPos= target.convertToNodeSpace(touches[i].getLocation());

//是否在有效范围内

if(cc.rectContainsPoint(rect, locationPos)){

arrTemp[i]= 1;

//单指移动——更新触摸点的位置显示

target.getParent().updatePos(i, locationPos);

}else{

arrTemp[i]= 0;

}

}

if(arrTemp[0]&&arrTemp[1]){//双指操作

target.DoubleFingers(touches[0], touches[1]);

}else if(arrTemp[0]){//单指拖动

target.Moving(touches[0].getDelta());

}

},

//func:多点触摸结束

onTouchesEnded:function(touches, event) {

var target = event.getCurrentTarget();

target.Clear();

},

/\*

func:

单指移动精灵

principle:

精灵要移动的目标位置= 当前位置+ 触摸移动的偏移量

@delta:触摸移动的偏移量

\*/

Moving:function(delta){

var self= this;

self.setPosition(cc.pAdd(self.getPosition(), delta));

},

/\*

func:

双指操作-【旋转、缩小、放大】

principle:

1、获取前两个触摸点在目标坐标系统中的位置;

2、计算:

两点之间的

距离= ((x2- x1)\*(x2- x1)+ (y2- y1)\*(y2- y1))的开方

cos值= (y2- y1)/(距离)

弧度值= (Math.acos(cos值))

角度= 弧度值\* 180/ Math.PI;

注意:

当两个触摸点处于同一竖直线时，角度值可能为NaN。

判定角度值是否是NaN，不能使用==NaN，而应该使用函数isNaN(value)

3、如果self.initDistance的值为0，表示从未移动过，可以将对应值设置为初始值;

4、否则，执行缩放、旋转操作

缩放系数= （当前两点的距离）/初始距离\* 精灵初始缩放系数;

旋转角度= （当前两点的角度- 两点之间的初始角度）;

精灵角度= （旋转角度+ 精灵的初始角度）%360;

\*/

DoubleFingers:function(touch1, touch2){

var self= this;

var locationPos1= self.convertToNodeSpace(touch1.getLocation());

var locationPos2= self.convertToNodeSpace(touch2.getLocation());

//获取两点之间距离

var pos= (locationPos2.x> locationPos1.x)?cc.pSub(locationPos2, locationPos1):cc.pSub(locationPos1, locationPos2);

var distance= Math.sqrt(pos.x\* pos.x+ pos.y\* pos.y);

var value= (Math.floor(pos.y))/(distance);

var angle= Math.acos(value)\* (180/ Math.PI);

angle= ((isNaN(angle))?0:angle);

if(self.initDistance== 0){//两点之间的初始值

self.initDistance=distance;

self.Angle= ((isNaN(self.getRotation()))?0:self.getRotation());//精灵的角度

self.initScale= self.getScale();

self.initAngle= angle;

}else{//变化【缩放(缩小、放大)、旋转】

//缩放

var percent= distance/self.initDistance;

self.setScale(percent\* self.initScale);

//旋转

angle-= ((self.initAngle- self.Angle));

self.setRotation(angle%360);

}

if(self.callback\_Moving!= null){

self.getParent().callback\_Moving(distance, Math.floor(self.initAngle), Math.floor(angle), Math.floor(self.Angle), Math.floor(self.getRotation()));

}

},

/\*

func:

触摸点结束，数据清空

\*/

Clear:function(){

var self= this;

self.initDistance= 0;//两点之间的距离

self.initAngle= 0;//两点之间的初始角度

self.Angle= 0;//精灵的初始角度

self.initScale= 0;//精灵的初始缩放系数

if(self.callback\_End!= null){

self.getParent().callback\_End(0, Math.floor(self.initAngle), 0, Math.floor(self.Angle), Math.floor(self.getRotation()));

}

},

/\*

func:

设置:触摸移动中的回调函数

\*/

SetCallback\_Moving:function(callback){

var self= this;

self.callback\_Moving= (callback== undefined)?null:callback;

},

/\*

func:

设置:触摸结束的回调函数

\*/

SetCallback\_Ended:function(callback){

var self= this;

self.callback\_End= (callback== undefined)?null:callback;

}

});

* 接口:

var self= this;

var title= new CGestureSprite(res.title\_png);

title.attr({

x:cc.winSize.width\* 0.5,

y:cc.winSize.height\* 0.5

});

self.addChild(title, 1, 0);

title.Init();

1. 算法: 监听机制

/\*

func:

监听机制

\*/

addTouch:function(target){

var self= this;

var listener= cc.EventListener.create({

event:cc.EventListener.TOUCH\_ONE\_BY\_ONE,

swallowTouches:true,

onTouchBegan:self.onTouchBegan,

onTouchEnded:self.onTouchEnded

});

cc.eventManager.addListener(listener, ((target==undefined)?self:target));

},

/\*

func:

开始触摸——判断是否触摸在有效区域

\*/

onTouchBegan:function(touch, event){

var target= event.getCurrentTarget();

var size= target.getContentSize();

var rect= cc.rect(0, 0, size.width, size.height);

var locationPos= target.convertToNodeSpace(touch.getLocation());

return cc.rectContainsPoint(rect, locationPos);

},

/\*

func:

结束触摸——回调函数

\*/

onTouchEnded:function(touch, event){

var target= event.getCurrentTarget();

target.addMile();

}

1. 算法: 生成此序列数组

0

1 -1

1 0 -1

2 1 -1 -2

2 1 0 -1 2

* + 原理:

如果在偶数一行中的中间位置添加0，后一个数始终比前一个数小1。每一列的起始数值均为:Math.floor(行号\* 0.5)

* + 实现:

function calculateArray(num){

var arr= new Array(0);

var totalNum= num%2?num:(num+ 1);

var began= Math.floor(totalNum\* 0.5);

for(var i= 0;i< totalNum; ++i){

arr.push(began);

began--;

}

if(num%2== 0){//偶数

arr.splice(Math.floor(totalNum\* 0.5), 1);

}

return arr;

}

* 举一反三:

答题项目时，有时选项的个数不定，为了均匀分配，需要动态设置其位置。序列数组必须要改成:

0

0.5 -0.5

1 0 -1

1.5 0.5 -0.5 -1.5

2 1 0 -1 2

* + 原理:

偶数时，>0的数值，-0.5，0<的的数值,加上0.5。

* + 实现:

function calculateArray(num){

var arr= new Array(0);

var totalNum= num%2?num:(num+ 1);

var began= Math.floor(totalNum\* 0.5);

for(var i= 0;i< totalNum; ++i){

arr.push(began);

began--;

}

if(num%2== 0){//偶数

arr.splice(Math.floor(totalNum\* 0.5), 1);

for(i=0; i<num; ++i){

arr[i]+= ((arr[i]>0)?(-0.5):0.5);

}

}

return arr;

}

1. 算法:求一个数值的正负号

var symbol= arr[i]/Math.abs(arr[i]);

1. 算法: 保留两位或者三位小数

num.toFixed(2)

1. 算法:设置菜单是否可用:

/\*

func:

set menu enabled

@target:menu

\*/

function setMenuEnabled(target){

for(var i= 0;i< target.getChildrenCount(); ++i){

target.children[i].setColor(cc.color(255, 255, 255, 255));

}

target.setEnabled(true);

}

/\*

func:

set item disEnabled

@target:menu item

\*/

function setItemDisEnabled(target){

var menu= target.getParent();

target.setColor(cc.color(128, 128, 128, 32));

menu.setEnabled(false);

}

/\*

func:

set menu enabled

@target:menu

\*/

function setMenuEnabled(target){

target.setEnabled(true);

}

/\*

func:

set item disEnabled

@target:menu item

\*/

function setItemDisEnabled(target){

var menu= target.getParent();

menu.setEnabled(false);

var scaleSmall= cc.scaleTo(0.2, 0.8);

var scaleBig= cc.scaleTo(0.2, 1);

var seq= cc.sequence(scaleSmall, scaleBig);

target.runAction(seq);

}

1. 算法:圆不变形

* 代码实现:

/\*

func:

目标不变形

@scale:缩放系数

@target:目标

\*/

function RoundOut(scale, target){

target.setScale(scale/ cc.view.getScaleX(), scale/ cc.view.getScaleY());

}

btn\_music.setScale(1, 1\* cc.view.getScaleX()/cc.view.getScaleY());

* 接口调用:

**RoundOut(1, popup);**

1. 算法:颠球动画

* 原理:

球向上跳的同时转动，球拍向下运动。

* 代码实现:

var jump1= cc.jumpTo(0.8, tennis.x, tennis.y, cc.winSize.height\* 0.5, 1);

var rotate1= cc.rotateBy(0.01,10);

tennis.runAction(jump1.repeatForever());

tennis.runAction(rotate1.repeatForever());

var jump2= cc.jumpTo(0.8, racket.x, racket.y, cc.winSize.height\* (- 0.08), 1);

var seq= cc.sequence(jump2);

racket.runAction(seq.repeatForever());

1. 算法:精灵是否被移除

car.isRunning()

1. 限制仅能在微信浏览器中打开:

* 原理:

方法一:

检测是否为微信浏览器，如果是，则打开。判定条件:

(cc.sys.isMobile && window.navigator.userAgent.indexOf("MicroMessenger") != -1),true:是微信浏览器。

方法二:

微信浏览器内部初始化成功以后，会调用**WeixinJSBridgeReady**接口，因此只要监听这个接口即可。

window.onload = function () {

//隐藏微信右上角菜单

document.addEventListener('WeixinJSBridgeReady',

function onBridgeReady() {

cc.game.onStart = function () {

//适配模式

var mode = cc.sys.isMobile && window.navigator.userAgent.indexOf("MicroMessenger") != -1 ? cc.ResolutionPolicy.EXACT\_FIT : cc.sys.isMobile ? cc.ResolutionPolicy.EXACT\_FIT : cc.ResolutionPolicy.SHOW\_ALL;

cc.view.setDesignResolutionSize(640, 960, mode);

cc.view.resizeWithBrowserSize(true);

//load resources

MyLoaderScene.preLoad(g\_preRes, g\_resources, function () {

cc.spriteFrameCache.addSpriteFrames(res.card\_plist, res.card\_png);

cc.director.runScene(new IndexScene());

}, this);

};

cc.game.run("gameCanvas");

});

};

1. 算法:判断手机号所属运营商

* Java实现:

String phone= "13522490962";//要校验的手机号

//移动手机号校验

String mobile\_regex= "^(134|135|136|137|138|139|147|150|151|152|157|158|159|178|182|183|184|187|188)[0-9]{8}$";

//联通手机号校验

String unicom\_regex= "^(130|131|132|145|155|156|176|185|186)[0-9]{8}$";

//电信手机号校验

String telecom\_regex= "^(133|153|177|180|181|189)[0-9]{8}$";

if(phone.matches(mobile\_regex)){//移动

}else if(phone.matches(unicom\_regex)){//联通

}else if(phone.matches(telecom\_regex)){//电信

}else{//未知运营商

}

* JavaScript实现:

/\*\*

\* 返回类型

1 中国移动

2 中国联通

3 中国电信

0 无法识别

\*\*/

function validate(phoneno){

var regex = /^(134|135|136|137|138|139|147|150|151|152|157|158|159|178|182|183|184|187|188)[0-9]{8}$/;

if(regex.test(phoneno)){

alert("中国移动！");

return 1;

}

regex = /^(130|131|132|145|155|156|176|185|186)[0-9]{8}$/;

if(regex.test(phoneno)){

alert("中国联通！");

return 2;

}

regex = /^(133|153|177|180|181|189)[0-9]{8}$/;

if(regex.test(phoneno)){

alert("中国电信！");

return 3;

}

return 0;

}

1. 算法: 在区间内【0,iRange）生成一个长度为iLength的随机整数数组，不重复

原理:

逐步遍历，生成iLength个不同的整数。随机出一个整数后，在已生成的数组中查找，如果找不到，说明新生成的整数没有重复，否则一直循环随机。

实现:

//在区间内【0,iRange）生成一个长度为iLength的随机整数数组，不重复

createDifferentArray:function(iLength, iRange)

{

var temp= new Array(0);

var num= 0;

//随机抽取iLength个不同的整数

for(var i=0; i< iLength; ++i)

{

do{

num= Math.floor(Math.random()\* iRange);

}while((-1)!= temp.indexOf(num));

temp.push(num);

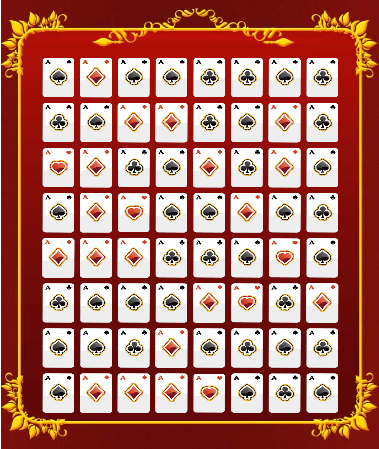
}

return temp;

}

1. 算法:在一定范围内,生成指定的行列数的精灵

效果:



原理:

缩放系数:

X轴方向上的缩放系数——(有效区域的宽-缝隙的尺寸)/(每一行有几列\* 子节点的原始宽度)

Y轴方向上的缩放系数——(有效区域的高- 缝隙的尺寸)/(每一列有几行\* 子节点的原始高度);

左下角点的起始位置:

可视区域的最左下角位置——(二维数组的中间位置)- (0.5\* 视区域尺寸)

最下角的子节点的起始位置——(可视区域的最左下角位置)+ (0.5\* (缩放系数\* 子节点的原始尺寸));

生成二维数组:

根据子节点所在的行列号，设置子节点的位置(包括间隔)，同时设置其缩放比例。

实现:

1. 设置可视区域尺寸、水平方向和竖直方向上的间隙:

self.iVStep= iVStep;

self.iHStep= iHStep;

self.visibleSize= visibleSize;

1. 每一行、每一列的数量:

self.iRow= iRow;

self.iCol= iCol;

1. 获取某一个每一个子节点的原始尺寸:

self.ItemSize= self.getItemSize();//获取子节点原始尺寸

//获取菜单项的原始尺寸

getItemSize:function()

{

var self= this;

//起始一个

var path= self.prePath+"1\_1.png";

var tempItem= cc.Sprite.create(path);

return tempItem.getContentSize();

}

1. 计算子节点在本关卡中，应该缩放的次数

/\*

func:

计算:缩放子节点的尺寸

\*/

calculateScale:function(){

var self= this;

var size= self.ItemSize;

self.fScaleX= (self.visibleSize.width- (self.iCol- 1)\* self.iHStep)/ (self.iCol\* size.width);

self.fScaleY= (self.visibleSize.height-(self.iRow- 1)\* self.iVStep)/ (self.iRow\* size.height);

}

1. 计算最下角的子节点的位置

/\*

func:

计算:最左下角图块的位置

\*/

calculateLeftBottomPos:function(){

var self= this;

//可视区域的最左下角的位置

self.leftBottomPos= cc.pSub(cc.p(cc.winSize.width\* 0.5, cc.winSize.height\* 0.375), cc.p(self.visibleSize.width\* 0.5, self.visibleSize.height\* 0.5));

//最下角的子节点位置

self.leftBottomPos= cc.pAdd(self.leftBottomPos, cc.p(self.fScaleX\* self.ItemSize.width\* 0.5, self.fScaleY\* self.ItemSize.height\* 0.5));

}

1. 根据行号和列号，设置子节点的位置:

//设置精灵位置

setItemPos:function(row, col, target)

{

var self= this;

var stepWidth= col\* self.iHStep;//夹缝的宽度

var stepHeight= row\* self.iVStep;//夹缝的高度

var itemsWidth= col\* self.ItemSize.width\* self.fScaleX;//精灵的高度【紧挨着】

var itemsHeight= row\* self.ItemSize.height\* self.fScaleY;//精灵的高度

var offsetPos= cc.p(stepWidth+ itemsWidth, stepHeight+ itemsHeight);//偏移量

var pos= cc.pAdd(self.leftBottomPos, offsetPos);//节点对应位置

target.setPosition(pos);

}

1. 创建二维数组:

for(var i=0;i< self.iRow; ++i)

{

for(var j=0;j< self.iCol; ++j)

{

var item= null;

var tag= 0;

//创建子节点菜单项

……

self.setItemPos(i, j, item);//设置子节点位置

item.setScale(self.fScaleX, self.fScaleY);//设置子节点的缩放比例

menu.addChild(item, 1, tag);

}

}

* //虚函数virtual
* //静态函数static作用