**繁花似锦后续开发技术方案**

**一、当前情况**

目前繁花似锦完全只由前端开发完成，并无涉及到后端的相关内容。由于完全由前端开发，所以会出现客户端被破解而出现游戏数据（主要是指货币）被篡改，而造成项目损失这种严重的情况。所以需要游戏的相关重要算法和数据从后端获取，客户只做相应展示，从而避免上面所述被破解情况。因此，需要新增后端的开发工作量和前端的修改工作量。

**二、解决方案**

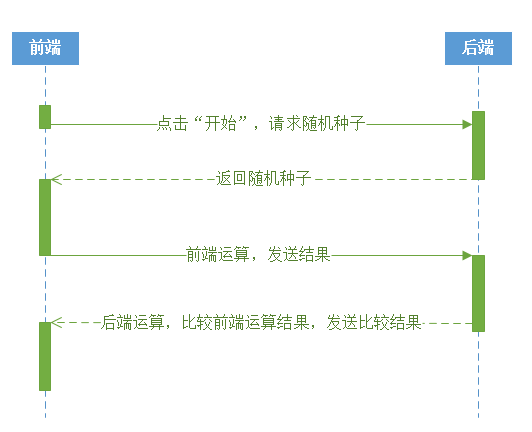
目前有两套解决方案：

方案一：后端根据项目的需求设计相关算法。点击开始游戏时，前端向后端请求结果，此时后端进行计算并得出结果，然后把结果发给前端，最后前端对结果做相应的过程模拟展示，此次操作成功。

方案二：由于前端已经实现了项目的算法（核心算法，随机算法），所以后端依照（或参考）前端的算法进行编码，确保前端和后端的随机种子是一样的（这样才能保证运算过程中的随机数是相同的），随机种子在每次点击开始游戏时都重新同步一次。点击开始游戏时，前端向后端请求随机种子，然后进行自己的运算工作，计算并得出结果，然后把结果发送给后端，后端此时开始计算并得出结果，然后比较后端计算所得结果与前端所发送过来的结果，当两个结果相等（或在允许的偏差范围之内）时，则采用前端的结果，此次操作成功，当结果不相等（或出现不允许的偏差时），则此次操作无效，前端提示用户（例如作弊提示）。

通过比较以上两种方案，决定采用方案二，理由是采用方案一的话，前端基本要重新编码，这样的话，耗时将十分之久，采用方案二的话，前端只需确保随机算法和随机种子与后端的一样即可，从项目的开发周期和复杂度来比较，方案二的可行性更高一些。

**三、对接流程**



以上流程的前提条件：

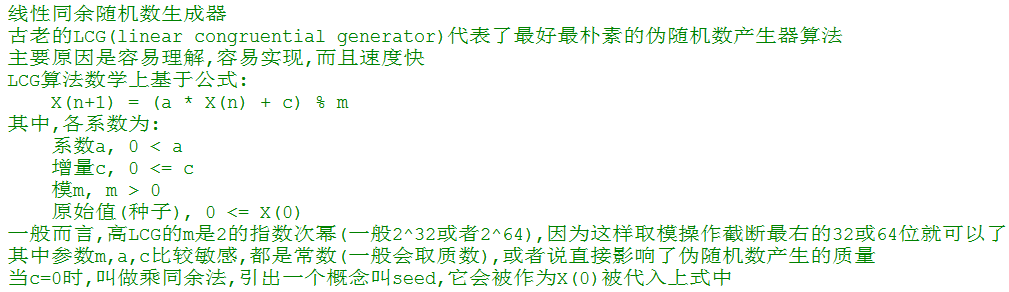
1、前端和后端的随机算法一致，且随机种子一致

2、前端和后端的核心算法一致

3、前端和后端的数据表一致

**四、随机算法**

前端已自行实现了伪随机算法的伪随机函数，前端所有的随机数都是通过此伪随机函数获取，后端可参考前端的伪随机算法编写伪随机函数。



// 随机算法实现

function CreateLCG(a, c, m) {

if ('number' != typeof(a)) {

a = 9301;

}

if ('number' != typeof(c)) {

c = 49297;

}

if ('number' != typeof(m)) {

m = 233280;

}

if (a <= 0) {

throw new Error("a is error");

}

if (c < 0) {

throw new Error("c is error");

}

if (m <= 0) {

throw new Error("m is error");

}

var \_seed = 0;

var lcg = {};

// private methods

function seededRandom() {

\_seed = (\_seed\*a + c)%m;

return \_seed/m;

}

// public methods

lcg.seed = function(seed) {

if ('number' != typeof(seed) || seed < 0) {

throw new Error("seed is error");

}

\_seed = seed;

};

lcg.random = function(min, max) {

if ('number' != typeof(min)) {

min = 0;

}

if ('number' != typeof(max)) {

max = 1;

}

return min + seededRandom()\*(max - min);

};

return lcg;

}

// 获取随机数

var mLCG = null;

function getRandom () {

if (!mLCG) {

mLCG = CreateLCG();

mLCG.seed(Math.random()); // 设置随机种子

}

return mLCG.random();

};

**五、根据权重计算概率算法**

// create probability object,weightList:[[1,20],[2,60]],contains two random value: 1.value,20.weight;2.value,60.weight

function CreateProbability(weightList, randomFunc) {

if (!(weightList instanceof Array)) {

throw new Error("weightList is not table, it's type is " + typeof(weightList));

}

var weightListInit = [];

for (var i = 0, len = weightList.length; i < len; ++i) {

var factor = weightList[i];

if (!(factor instanceof Array)) {

throw new Error("weight list at index [" + i + "] is not table, it't type is " + typeof(factor));

}

if (undefined == factor[0] || null == factor[0]) {

throw new Error("weight list at index [" + i + "] format is error, not support value type " + typeof(factor[0]));

}

if (isNaN(factor[1])) {

throw new Error("weight list at index [" + i + "] format is error, not support weight type " + typeof(factor[1]));

}

if (factor[1] < 0) {

throw new Error("weight list at index [" + i + "] is error, not support weight value " + factor[1] + " < 0");

}

weightListInit[i] = [factor[0], factor[1]];

}

// private member variables

var mThreshold = 1;

var mWeightRange = [];

var probability = {};

// private methods

function random(min, max) {

randomFunc = 'function' == typeof(randomFunc) ? randomFunc : function() {

return Math.random();

};

var randomValue = randomFunc();

if (isNaN(randomValue)) {

throw new Error("random value is not number");

}

return Math.floor(randomValue \* (max - min) + min + 0.5);

}

function parseWeightList() {

for (var index = 0, count = weightList.length; index < count; ++index) {

var randIndex = random(0, index);

if (randIndex != index) {

var temp = weightList[randIndex];

weightList[randIndex] = weightList[index];

weightList[index] = temp;

}

}

mThreshold = 1;

mWeightRange = [];

for (var i = 0, len = weightList.length; i < len; ++i) {

var factor = weightList[i];

if (!(factor instanceof Array) || isNaN(factor[1])) {

throw new Error("weightList format is error");

}

var value = factor[0];

var weight = factor[1];

if (weight > 0) {

var range = {value: value, begin: mThreshold, end: mThreshold + weight - 1};

mWeightRange.push(range);

mThreshold += weight;

}

}

}

// public methods

probability.getValue = function() {

if (1 == mThreshold) {

return;

}

var index = random(1, mThreshold - 1);

for (var i = 0, len = mWeightRange.length; i < len; ++i) {

var range = mWeightRange[i];

if (index >= range.begin && index <= range.end) {

return range.value;

}

}

};

probability.getWeight = function(value) {

for (var i = 0, len = weightList.length; i < len; ++i) {

var factor = weightList[i];

if (value == factor[0]) {

return factor[1];

}

}

return 0;

};

probability.getTotalWeight = function() {

var totalWeight = 0;

for (var i = 0, len = weightList.length; i < len; ++i) {

var factor = weightList[i];

totalWeight += factor[1];

}

return totalWeight;

};

probability.setWeight = function(value, weight) {

if (undefined == value || null == value) {

throw new Error("not support value type " + typeof(value));

}

if (isNaN(weight)) {

throw new Error("not support weight type " + typeof(value));

}

if (weight < 0) {

throw new Error("not support weight value " + weight + " < 0");

}

var isFind = false;

for (var i = 0, len = weightList.length; i < len; ++i) {

if (value == weightList[i][0]) {

weightList[i][1] = weight;

isFind = true;

break;

}

}

if (!isFind) {

weightList.push([value, weight]);

}

parseWeightList();

};

probability.reset = function() {

weightList = [];

for (var i = 0, len = weightListInit.length; i < len; ++i) {

var factor = weightListInit[i];

weightList[i] = [factor[0], factor[1]];

}

parseWeightList();

};

// initialize weight list

parseWeightList();

return probability;

}

**六、核心数据结构**

地图：地图大小为6x6，总共36个格子，横坐标从左->右，纵坐标从下->上

第1列，第2列，第3列，第4列，第5列，第6列

第6行： (6,1), (6,2), (6,3), (6,4), (6,5), (6,6)

第5行： (5,1), (5,2), (5,3), (5,4), (5,5), (5,6)

第4行： (4,1), (4,2), (4,3), (4,4), (4,5), (4,6)

第3行： (3,1), (3,2), (3,3), (3,4), (3,5), (3,6)

第2行： (2,1), (2,2), (2,3), (2,4), (2,5), (2,6)

第1行： (1,1), (1,2), (1,3), (1,4), (1,5), (1,6)

格子：格子是一种数组结构，可包含4个元素，例如，[地表元素, 物件元素, 附加元素, 覆盖元素]。

元素：地图中的最小数据结构，有自己的属性。

地表元素：在格子中的位置为第0个，共有2种，表现为，

物件元素：在格子的位置为第1个，共有8种，表现为、、、、、、、

附加元素：在格子的位置为第2个，共有2种，表示为：十字消技能、X消技能

覆盖元素：在格子的位置为第3个，共有1种，表示为：固定元素

**七、数据表说明**

掉落表：

关卡表：

特效表：，bonus1和bonus3的效果

地图表：，主要是设置每个关卡的地表元素位置

**八、bonus1，bonus2触发算法**

// bonus1效果获取函数

function getBonus1Effect (levelIdOrData) {

var levelData = null;

if ('number' == typeof(levelIdOrData) && levelIdOrData > 0) {

levelData = DataTB.get("level\_tplt", levelIdOrData, true);

} else if (levelIdOrData && 'object' == typeof(levelIdOrData) && !(levelIdOrData instanceof Array)) {

levelData = levelIdOrData;

} else {

return;

}

function randomFunc() {

return getRandom();

};

var effectProbability = CreateProbability(levelData.bonus1, randomFunc); // 把百分比转为权重计算方法

var effectTotalWeight = effectProbability.getTotalWeight();

if (1000 != effectTotalWeight) { // 由于百分比(10表示1%),所以1000表示百分百,这里必须确保所有效果id的概率相加为百分百(必须有一个效果命中)

throw new Error("关卡[" + levelData.name +"]中的bonus1配置出错,概率相加不等于百分百");

}

var effectId = effectProbability.getValue(); // 效果id

if (0 == effectId) {

return;

}

var effectData = DataTB.get("effect\_tplt", effectId, true);

var valueProbability = CreateProbability(effectData.value, randomFunc);

var valueTotalWeight = valueProbability.getTotalWeight();

if (valueTotalWeight > 0 && 1000 != valueTotalWeight) {

throw new Error("特效表[" + effectId +"]中的概率相加不等于百分百");

}

var data = {

type: effectData.type, // 效果类型

value: valueProbability.getValue() // 效果值

}

return data;

};

-------------------------------------------------------------------------------------------------------

// bonus3效果获取函数

function getBonus3Effect(levelIdOrData) {

var levelData = null;

if ('number' == typeof(levelIdOrData) && levelIdOrData > 0) {

levelData = DataTB.get("level\_tplt", levelIdOrData, true);

} else if (levelIdOrData && 'object' == typeof(levelIdOrData) && !(levelIdOrData instanceof Array)) {

levelData = levelIdOrData;

} else {

return;

}

function randomFunc() {

return DataModel.getRandom();

};

var effectProbability = CreateProbability(levelData.bonus3, randomFunc); // 把百分比转为权重计算方法

var effectTotalWeight = effectProbability.getTotalWeight();

if (effectTotalWeight < 0 || effectTotalWeight > 1000) { // 由于百分比(10表示1%),所以1000表示百分百,这里必须确保所有效果id的概率相加为<百分百(有概率命中)

throw new Error("关卡[" + levelData.name +"]中的bonus3配置出错,概率相加出错");

}

effectProbability.setWeight(0, 1000 - effectTotalWeight);

var effectId = effectProbability.getValue(); // 效果id

if (0 == effectId) {

return;

}

var effectData = DataTB.get("effect\_tplt", effectId, true);

var valueProbability = CreateProbability(effectData.value, randomFunc);

var valueTotalWeight = valueProbability.getTotalWeight();

if (valueTotalWeight > 0 && 1000 != valueTotalWeight) {

throw new Error("特效表[" + effectId +"]中的概率相加不等于百分百");

}

var data = {

type: effectData.type, // 效果类型

value: valueProbability.getValue() // 效果值

}

return data;

};

-------------------------------------------------------------------------------------------------------

// 游戏阶段

game\_section = {

"normal" : 1, // 常规阶段

"bonus1" : 2, // bonus1阶段

"bonus2" : 3, // bonus2阶段

"bonus3" : 4 // bonus3阶段

};

// 元素单价

ELEMENT\_PRICE: {

20101: 12, // 红色

20102: 10, // 黄色

20103: 8, // 绿色

20104: 6, // 蓝色

20105: 5, // 紫色

20106: 3, // 粉红

20107: 2, // 李子

20108: 0 // 百搭

};

**九、奖励计算算法**

// 计算奖励数据,screenTimes:刷屏次数,objectId:元素id,count:消除个数

function calcAwardData(levelId, gameSection, screenTimes, objectId, count) {

var levelData = DataTB.get("level\_tplt", levelId, true);

var rateList = [];

if (game\_section["normal"] == gameSection) { // 常规刷屏

rateList = levelData.rate\_normal;

} else if (game\_section["bonus1"] == gameSection) { // bonus1刷屏

rateList = levelData.rate\_bonus1;

} else if (game\_section["bonus2"] == gameSection) { // bonus2刷屏

rateList = levelData.rate\_bonus2;

} else if (game\_section["bonus3"] == gameSection) { // bonus3刷屏

rateList = levelData.rate\_bonus3;

} else {

rateList = levelData.rate\_normal;

}

screenTimes = (screenTimes >= rateList.length) ? rateList.length - 1 : screenTimes;

var rate = rateList[screenTimes]; // 刷屏倍率

var elementData = DataTB.get("map\_element\_tplt", objectId, true);

var price = ELEMENT\_PRICE[objectId]; // 元素单价

var value = price \* count \* rate; // 元素价值=元素单价\*数量\*刷屏倍率

var data = {

count: count, // 元素数量

image: elementData.file, // 元素图片

value: value // 元素价值

};

return data;

};

**十、掉落算法**

地图搜索顺序为：先左->右，然后下->上

空格子：格子数组里无“物件元素”

先搜索(1,1)，判断其下方是否为有效坐标并且是空的格子，如果有效其为空，则把(1,1)的数据移动到其下方的格子位置，然后(1,1)置空

然后搜索(1,2)，类同(1,1)的判断方式

以此类推，当(1,6)搜索结束后，则从上一行的起始位置开始搜索，即从(2,1)开始搜索，以此类推直至(2,6)搜索结束，再往上一行搜索

最后当(6,6)搜索结束时，本地图的数据已经全部发生了掉落，然后判断第6行的每个格子，即从(6,1)->(6,6)判断格子是否为空，如果非空，则在该空格生成新的元素

**十一、元素生成算法**

掉落数据表目前设计为100x6的二维数组，即100行，每行6列，所以每一列对应有100条数据。当在游戏中每次点击“开始”时，对6列数据根据0-99的下标各随机出一个索引值作为起始点，然后从二维数组中取出指定位置的数据，并填充到对应的格子里面。注意：索引值的顺序为递减取值，当索引值为0时，重新指向99索引值，然后递减取值。