二维随机正态分布点的生成——Box-Muller变换算法*

均匀分布的Math.random()

实际上在游戏中不仅仅是战斗命中率需要满足正态分布，就连世界观中，人口的分布、身高分布、角色能力值等等，都需要满足正态分布。

我们知道在JavaScript中有一个`Math.random()`方法，这个玩意我记得在几乎所有语言中都有，实际上我学过的这些语言中，都有这个方法，可以说是万金油。但是这个方法有一个问题，那就是，，，，，他是一个均匀分布！！！！！！

我们不妨来看看：

```html
<!DOCTYPE html>
<html lang="ch-Zh">

<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>

<body>
    <canvas width="1000px" height="1000px">

    </canvas>
    <script>
        var testNum =1000;//样本数
        var testRange = 100;//测试范围
        var array = []

        //初始化
        for (let i =0;i<testRange;i++)
            array[i] =0;
        //获取随机数
        for (let i = 0; i < testNum; i++) {
            var num = Math.floor(Math.random()*testRange)
            array[num]++;
        }
        var canvas = document.getElementsByTagName("canvas")[0]
        var ctx = canvas.getContext('2d')
        for(let i=0;i<testRange;i++){
            ctx.strokeRect(i*20,100,20,array[i]*10)//用数组出现的次数来描述长度
        }

        ctx.fillText("Math.random()方法服从均匀分布",400,50);//实心字
    </script>
</body>

</html>
```

Box-Muller变换（笛卡尔坐标系下）

将一个均匀分布变成正态分布，这就是Box-Muller变换，原理的证明，知乎上写的挺不错的，这里就不再证明了。这里只介绍一下使用方法：

1. 准备两个服从均匀分布的随机变量U1、U2。
2. 那么必然有两个正态分布的变量N1、N2满足。

$$N1 = \sqrt{-2\ln U_1}\cos(2\pi U_2)$$
$$N2 = \sqrt{-2\ln U_1}\sin(2\pi U_2)$$

```html
<!DOCTYPE html>
<html lang="ch-Zh">

<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>

<body>
    <canvas width="2000px" height="1000px"></canvas>
    <script>

        function getNumberInNormalDistribution(mean, delta) {//第一个是均值，第二个是浮动的范围
            var x = mean + (randomNormalDistribution()[0] * delta)
            var y = mean + (randomNormalDistribution()[1] * delta)
            return [x, y];
        }

        function randomNormalDistribution() {
            var U1, U2, N1, N2 ;
            U1 = Math.random();
            U2 = Math.random();

            N1 = Math.sqrt(-2 * Math.log(U1)) * Math.cos(2 * Math.PI * U2);
            N2 = Math.sqrt(-2 * Math.log(U1)) * Math.sin(2 * Math.PI * U2);
            return [N1, N2];
        }



        var testNum = 1000;//样本数
        var testRange = 1000;//测试范围
        var array = []

        //初始化
        for (let i = 0; i < testRange; i++)
            array[i] = 0;
        //获取随机数
        for (let i = 0; i < testNum; i++) {
            var num = Math.floor(getNumberInNormalDistribution(30,10)[0])
            array[num]++;
        }
        var canvas = document.getElementsByTagName("canvas")[0]
        var ctx = canvas.getContext('2d')
        for (let i = 0; i < array.length; i++) {
            ctx.strokeRect(i * 20, 100, 20, array[i] * 10)//用数组出现的次数来描述长度
        }

        ctx.fillText("正态分布", 400, 50);//实心字
    </script>
</body>

</html>
```

Box-Muller变换（极坐标下）

如果在极坐标下，那么其方程会变成：

$$z_0=\sqrt{-2\ln U_1}\cos(2\pi U_2)=\sqrt{-2\ln s}\left(\frac{u}{\sqrt{s}}\right)=u\cdot\sqrt{\frac{-2\ln s}{s}}$$

$$z_1=\sqrt{-2\ln U_1}\sin(2\pi U_2)=\sqrt{-2\ln s}\left(\frac{v}{\sqrt{s}}\right)=v\cdot\sqrt{\frac{-2\ln s}{s}}.$$

其中 $s=\sqrt{u^2+v^2}$，u 和 v 为均匀分布的随机数。

```html
<!DOCTYPE html>
<html lang="ch-zh">
```

```

<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>

<body>
    <canvas width="2000px" height="2000px">

    </canvas>
    <script>
        //随机数生成

        function getNumberInNormalDistribution(mean, delta) {//第一个是均值，第二个是浮动的范围
            var x = mean + (randomNormalDistribution()[0] * delta)
            var y = mean + (randomNormalDistribution()[1] * delta)
            return [x, y];
        }

        function randomNormalDistribution() {
            var u = 0.0, v = 0.0, s = 0.0, c = 0.0;
            do {
                u = Math.random() * 2 - 1.0;
                v = Math.random() * 2 - 1.0;
                s = u * u + v * v;
            } while (s == 0.0 || s >= 1.0)
            c = Math.sqrt((-2 * Math.log(s)) / s);
            // return u * c; //测试正态分布
            return [u * c, v * c];
        }


        class NormalPoint {
            constructor(mean, delta) {
                var [x, y] = getNumberInNormalDistribution(mean, delta)
                this.x = x;
                this.y = y;
            }



            static createPoints(num) {//创造num个点
                var points = [];
                var succeedPoint = 0;
                while (succeedPoint < num) {
                    var tempPoint = new NormalPoint(500, 200);//范围值

                        // console.log(tempPoint.checkValid(points))//永远会输出true
                        points.push(tempPoint);
                        succeedPoint++;

                }
                return points
            }

            static drawPoints(points) {
                var canvas = document.getElementsByTagName("canvas")[0]
                var ctx = canvas.getContext('2d')

                points.forEach(point => {
                    ctx.save();//保存画笔状态
                    ctx.moveTo(point.x + 30, point.y);
                    ctx.arc(point.x, point.y, 30, 0, 2 * Math.PI, false);
                    ctx.stroke();
                    ctx.restore();//恢复画笔状态
                })
            }
        }
         var points = NormalPoint.createPoints(600);//生成600个点
         NormalPoint.drawPoints(points);

```

```
    </script>
</body>

</html>
```

RPG中回避率和暴击率的平衡

数值的完善

首先在设计前，我们得要有一个明确的目标，主角的战斗力大约有多强？atk和def对战斗有多大的影响？

我们不妨先进行设计，然后再建模。

咱们的战斗数值设计如下：

- 无论防御力多高，都会受到至少1的攻击
- 攻击力越高，或者防御力越低，受到的攻击就越大
- 攻击要保持平衡

假如有5个角色，有的擅长暴击，有的擅长高命中率，有的则均衡。

必须保证平均伤害必须一样。

那么我们不妨把战斗公式设为：

$$defender.hp = defender.hp - attacker.atk * \frac{attacker.atk}{defender.def + attacker.atk}$$

这样用比值来确定伤害的方式可以保证每次攻击都有伤害，在双方攻防相等的情况下攻击力可以被减半。当防御者防御力无穷大时，伤害接近0。攻击力无穷大时，攻击力接近其本身。

> 唔，，，但是一般战斗系统都是有暴击率和挥空率的设计吧。 说的也是呢
> 那么我们也把这两个看脸的属性加进去吧！每次都命中的话，实际上就真的有点堆数值的感觉了

说到这个暴击率和挥空率啊， 不管怎么设计，都得要保证一个东西。所受伤害的期望不能发生变化。

不能说我加了两个功能之后，掉血更快了，或者大家战斗存活时间更久了。

那么我们不妨列出等式：

$$\text{普通伤害} * 1 = \text{闪避率} * 0 + （1 - \text{挥空率}） * [\text{暴击率} * \text{暴击伤害} + （1 - \text{暴击率}） * \text{普通伤害}]$$

左边就是原本攻击伤害的期望，普通攻击的命中率是100%，所以乘以1，而右边则是综合伤害的期望。

然后我们不妨推导一下：

$$\text{普通伤害} = [\text{暴击率} * \text{暴击伤害} + （1 - \text{暴击率}） * \text{普通伤害}] - \text{挥空率} * [\text{暴击率} * \text{暴击伤害} + （1 - \text{暴击率}） * \text{普通伤害}]$$

$$\text{挥空率} * [\text{暴击率} * \text{暴击伤害} + （1 - \text{暴击率}） * \text{普通伤害}] = [\text{暴击率} * \text{暴击伤害} + （1 - \text{暴击率}） * \text{普通伤害}] - \text{普通伤害}$$

$$\text{挥空率} = \frac{[\text{暴击率} * \text{暴击伤害} + （1 - \text{暴击率}） * \text{普通伤害}] - \text{普通伤害}}{[\text{暴击率} * \text{暴击伤害} + （1 - \text{暴击率}） * \text{普通伤害}]}$$

我们不妨把中间这个很长的东西设为攻击期望，那么式子可以化简为

$$\text{挥空率} = \frac{\text{攻击期望} - \text{普通伤害}}{\text{攻击期望}}$$

$$\text{攻击期望} = \text{暴击率} * \text{暴击伤害} + （1 - \text{暴击率}） * \text{普通伤害}$$

然后就是暴击率的确定，我们不妨规定其为小概率事件。那么按照正态分布$N(\mu, \sigma^2)$，在$(\mu-2\sigma, \mu+2\sigma)$区间内为普通攻击，之外为暴击，那么暴击率为（1-95.449974），差不多是5%左右。

然后代入闪避率的公式，就可以动态计算挥空率了。