# 技能系统

目录

[技能系统 1](#_Toc493498043)

[1. 技能点 3](#_Toc493498044)

[1.1. 激活技能编号 3](#_Toc493498045)

[1.2. 角色需求技能点 3](#_Toc493498046)

[2. 主动技能 4](#_Toc493498047)

[2.1. 编号 4](#_Toc493498048)

[2.2. 名称 4](#_Toc493498049)

[2.3. 描述 4](#_Toc493498050)

[2.4. 目标类型 5](#_Toc493498051)

[2.5. 目标限制 6](#_Toc493498052)

[2.6. 多目标 6](#_Toc493498053)

[2.7. 目标参数 7](#_Toc493498054)

[2.8. 飞行特效 7](#_Toc493498055)

[2.9. 命中特效 8](#_Toc493498056)

[2.10. 射程 8](#_Toc493498057)

[2.11. 伤害 8](#_Toc493498058)

[2.12. 属性改变 9](#_Toc493498059)

[2.13. 特殊效果 9](#_Toc493498060)

[2.14. 消耗时间 9](#_Toc493498061)

[2.15. 能耗 9](#_Toc493498062)

[3. 召唤技能 10](#_Toc493498063)

[3.1. 目标类型 10](#_Toc493498064)

[3.2. 召唤距离 10](#_Toc493498065)

[3.3. 召唤物属性表达式 10](#_Toc493498066)

[3.4. 行为模式 10](#_Toc493498067)

[3.5. 行为参数 10](#_Toc493498068)

[4. 被动技能 11](#_Toc493498069)

[4.1. 作用类型 11](#_Toc493498070)

[4.2. 参数 11](#_Toc493498071)

[5. 状态 12](#_Toc493498072)

[5.1. Buff/Debuff 12](#_Toc493498073)

[5.2. 最大叠加 12](#_Toc493498074)

[5.3. 持续时间 12](#_Toc493498075)

[5.4. 标记 12](#_Toc493498076)

[5.5. 状态优先级 13](#_Toc493498077)

[5.6. 效果 13](#_Toc493498078)

[5.7. 参数 13](#_Toc493498079)

请对应最新配置表阅读以下内容。

## 技能点

这里指玩家获得技能点之后的处理逻辑，而不涉及怎么获得技能点。

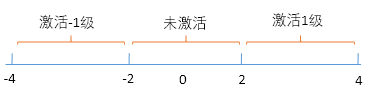
### 激活技能编号

* skillIds，ulong数组（ulong一般用来表示id）。
* 获得技能点后，依次激活的技能。

### 角色需求技能点

* characterActivate ,int二维数组。
* 第一维表示角色数量（玩家可选角色）。
* 第二维是该角色激活对应等级技能所需的技能点数。这是角色差异化的一个重要体现。
* 比如，skillIds是10001;10003;10005;10007, characterActivate 某角色需求技能点是4;5;7;10，那么表示该技能点累计到4后，激活技能10001，累计到5，激活10003…
* 该值可以为负：

某些芯片提供了很多其他高价值属性，代价是会降低某些技能点。如果负数达到某个值，同样激活技能，这种负激活的技能通常是属性向的被动技能，负数激活一般代表该技能的作用是降低某些属性。



## 主动技能

### 编号

* 如果是使用我之前编写的配置表工具与配套代码的话，编号（id）是一定会有的，即便在配置表里没有明确声明该字段。

如果配置表没有明确声明id字段，在代码里会自动生成ulong类型的id，并且从1开始自增。

### 名称

* 技能的名称。

事实上我是考虑过多语言的，但即便如此，在最初的配置表里也应该使用中文来作为标记，否则非常反人类……

为此，我专门准备了一个EasyConfig（简称EC）的类来解决这个问题，EC可以将数据（string）以数组形式存放，也可以按照KEY-VALUE键值对的方式来存放（还可以支持一个key对应多个value）到TXT，这里直接将配置的中文名认为是KEY就行，如果在EC里找到该KEY，那么就取对应的VALUE，否则就是用KEY为显示文本。

### 描述

* 描述技能特性与关键数值。
* 在技能描述里，会用到部分转义字符，也需要对富文本进行支持（其实只会用到颜色控制）。
* 转义字符：使用“[key]”的形式，其中key会根据需求增加，通常是技能宿主的属性字段，也可以是该技能中的某个字段值。

一个通常的用法是：对目标造成[skillDamge]倍相当于你当前生命值（[nhp]）的物理伤害。

其中[skillDamge]是技能中的一个字段（假设存在），[nhp]是宿主当前生命。

召唤物是可以成为宿主的。

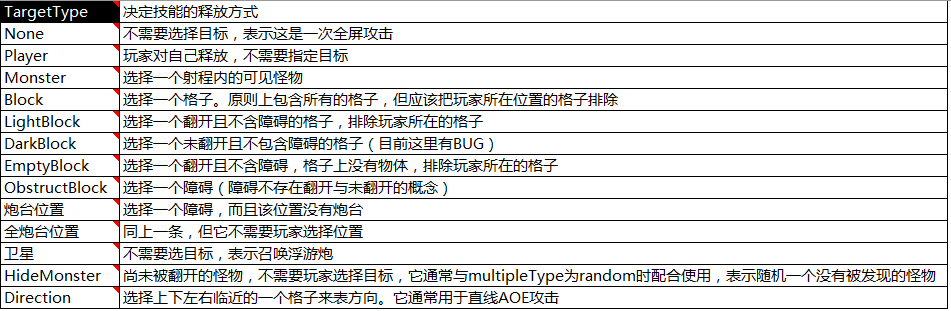
### 目标类型

* targetType，TargetType类型的参数。
* 它的设计初衷，是决定技能的释放方式，在游戏中可能会有以下几种技能释放方式：
* 需要玩家指定1个可见怪物；
* 指定1个格子（无视格子是否翻开，无视格子上是否有物体，不包含障碍）；
* 指定1个空格子（没有物体，其实这样它就必须是翻开的状态，不包含障碍）；
* 指定1个未翻开的格子（Near状态，即邻接玩家的格子同样算未翻开，不包含障碍）；
* 指定1个已经翻开的格子（不包含障碍）；
* 指定1个障碍（不可通过的格子）；

FindTarget类是用来专门解决目标寻找问题的， FindTarget将自己作为消息参数发送出去（原谅我已经不知道该怎么给消息起名了，话说我觉得中文还是蛮好用的…），接收该消息的Block会自行判断自己是否符合筛选条件，并把自己加入FindTarget的目标列表。

* 以及，不需要玩家指定目标（比如全体攻击技能，或者给自己释放BUFF，或者随机目标）。

实际情况中，还需要考虑到技能技能的射程。



* 综上，我们游戏中的技能不会出现让玩家选择1个以上的目标（通常这种会变成随机目标），要嘛选1个目标，要嘛1个都不用选。

### 目标限制

注意，配置表有修改

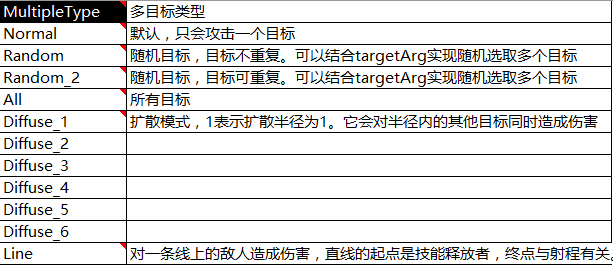
* targetLimit，TargetLimit类型的1维数组（枚举信息在<技能.xlasm>[枚举参数]中）。
* 它其实是一个过滤器，比如某些技能不能对某类技能释放，某些技能不能对某个强度的怪物释放。
* 之后根据使用场景进行增删。



如果这里为空，其实程序里读取到的是None

### 多目标

* multipleType，枚举类型的参数。
* 现在支持内容如下（某些内容需要配合targetArg）



### 目标参数

* targetArg，二维float数组。
* 第一维（第一次分割）表示对N个目标产生攻击，第二维表示分段伤害。
* 第一维目前只会在multipleType为Random时起效，multipleType为其他值时，只会取第一个元素。
* 第二维表示这次攻击伤害被分成了几段。

我并不希望一个很华丽演出的效果打了几分钟，结果过程中目标根本不掉血，只有最后一下暴了一个数值…或者每次攻击都只能发射一个炮弹？所以这个参数的作用就是把一个本来伤害100的攻击，分成了多次进行。

如果飞行特效是非激光类，那么有多少次伤害，就会实际产生多少次飞行道具（炮弹？），如果是激光类，表示本次激光造成N次伤害。但是他们不论有多少次伤害，总量还是100，所以这个参数叫伤害权重，如果是1;3这种形式，表示会产生2次伤害，第一次造成25，第二次造成75.

关于SuperArray：

该字段，读取到的是SuperArray<float>这种类型的对象。以配置内容“1;2;3|4;5”为例。你可以使用superArray.Count()获取第一次分割的长度，结果是2（superArray.Count(0)得到3，它表示第一次分割第0号元素的长度）。要获取某个元素可以使用superArray[1,1]，结果是5。注意，你不能使用superArray[0]直接获取值（除非它是个一维的参数），但是可以用superArray.ToArray(0)或者superArray.ToList(0)来将第一维的0号元素转换成数组。如果直接对superArray进行foreach遍历，它会依次返回1,2,3,4,5.

### 飞行特效

* effect\_fly，一维string数组。
* 字符串就是prefab的名字，原则上会把飞行特效做成一个个prefab，在prefab上绑定脚本以实现飞行特效自己的特性（比如有些飞直线，有得飞曲线），是飞行道具还是激光。
* 如果元素大于1个，每次就随机挑一个。（激光就只取第一个）

### 命中特效

* effect\_hit，一维string数组。
* 和飞行特效一样，它也是一个prefab的名字，它会在对目标造成伤害时，在目标身上播放。
* 同样，如果有多个元素，就每次随机挑一个。

### 射程

* carry，一维int数组。
* 它有且仅有2个元素，第一个元素表示最短距离，第二个表示最大距离，目标必须在这个范围内，射程才能判断成功。

### 伤害

* damage，一维string数组，第一个元素是伤害类型。
* 这里放的是一组四则混合运算（支持加减乘除，括号）的公式，其计算结果就是最终的直接伤害，所以我会将攻防计算全放在这里，在功能层面只需要提供攻防双方的属性，无需关心计算过程。

举例（这是我上一个DIY项目使用的方案）：



其中，冒号前表示这次伤害的类型（M为magic），冒号之后是四则混合运算，其中的各种参数是事前约定好的属性表示方式（包含技能释放者与技能命中者）。

解析四则混合运算最麻烦的地方是需要先将公式转换成逆波兰式，好在这部分功能我已经写好了，使用SuperTool.ToRpn函数，可以将一个合法的四则混合运算字符串转换成逆波兰式，比如上面那个转换后为，之后就好处理了，稍微google一下算法，后面的很容易（每个游戏的属性方案不一样，所以第二步计算不能照搬之前项目的解析方案）。

* 这部分内容需要提前约定属性关键字。

### 特效成功率

* successRate，一维float数组，4个元素，分别对应4种强度的怪物。
* 后面的“属性改变”与“特殊效果”都属于特效，它针对不同强度的怪物会有不同的成功率。

### 属性改变

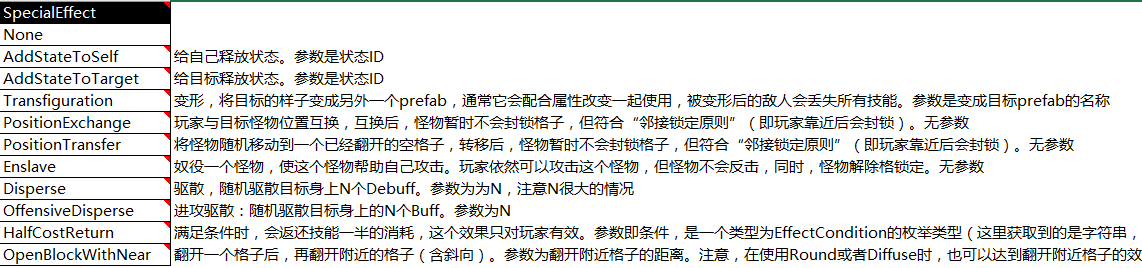
* propertyChange，一维string数组。
* 最通常是用来恢复生命值，但也有一些很特殊的用法，比如以下技能



其中的T是临时存放属性。

### 特殊效果

* specialEffect，枚举类型数组。
* 主动技能并不一定只会造成伤害，它可以是一些其他效果：

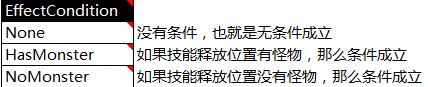


OpenBlockWithNear：翻开一个格子后，再翻开附近的格子（含斜向）。参数为翻开附近格子的距离。注意，在使用Round或者Diffuse时，也可以达到翻开附近格子的效果，他们的区别在于，使用Round或者Diffuse所翻开的格子，都可以被认定是“技能释放的目标位置”，而使用参数达到翻开附近格子的效果，这些格子是不会被判定为“技能释放的目标位置”的，这在使用EffectCondition做为判定条件时，有显著的区别。原则上，如果使用了Round或者Diffuse，而此效果参数又不为0，会先根据Round或者Diffuse翻开周围格子后，再在此基础上，又翻开一圈（后翻开的格子不作为“技能释放的目标位置”）。

* 数组表示可以同时有多个效果，每个效果请依次执行。

### 效果条件

* 一些效果必须要满足条件才能起效，比如HalfCostReturn。



### 消耗时间

* costTime，float类型的参数。
* 使用一个技能需要消耗的时间。如果是怪物的技能，该值表示CD。

### 能耗

* usePower，float类型的参数。
* 使用一个技能需要消耗的能量，注意这只在该技能属于召唤物，而且召唤的行为模式是复制攻击时才会用到。

## 召唤技能

指召唤炮台与浮游炮的召唤技能，怪物不会有此类技能。注意召唤技能通常情况下是不会提供给玩家直接使用的（只有召唤炮台需要玩家主动使用，浮游炮基本上是通过被动间接使用）

### 目标类型

* targetType，TargetType枚举类型。
* 和主动技能的targetType是一回事，只是我把主动\召唤技能目标类型放在一个类型里，其实很多时候他们并不能互用。

### 召唤距离

* carry，int数组，2个元素。
* 召唤距离，这是放置位置的距离，而不是攻击距离，所以只有召唤炮台的时候有效。

### 行为模式

* specialAction，枚举类型。
* 目前只有NormalAttack和CopyAttack两种模式。NormalAttack表示该召唤物拥有自己的技能，CopyAttack是复制宿主使用的技能（宿主用什么技能，它就用什么技能）。

### 行为参数

* speciaArg，string数组。
* 当specialAction为NormalAttack时，speciaArg有且仅有1个元素表示技能编号。
* 当specialAction为CopyAttack时，speciaArg第一元素表示该召唤物能量上限，第二个元素表示能量恢复速度（多长时间可以从0恢复至满）。

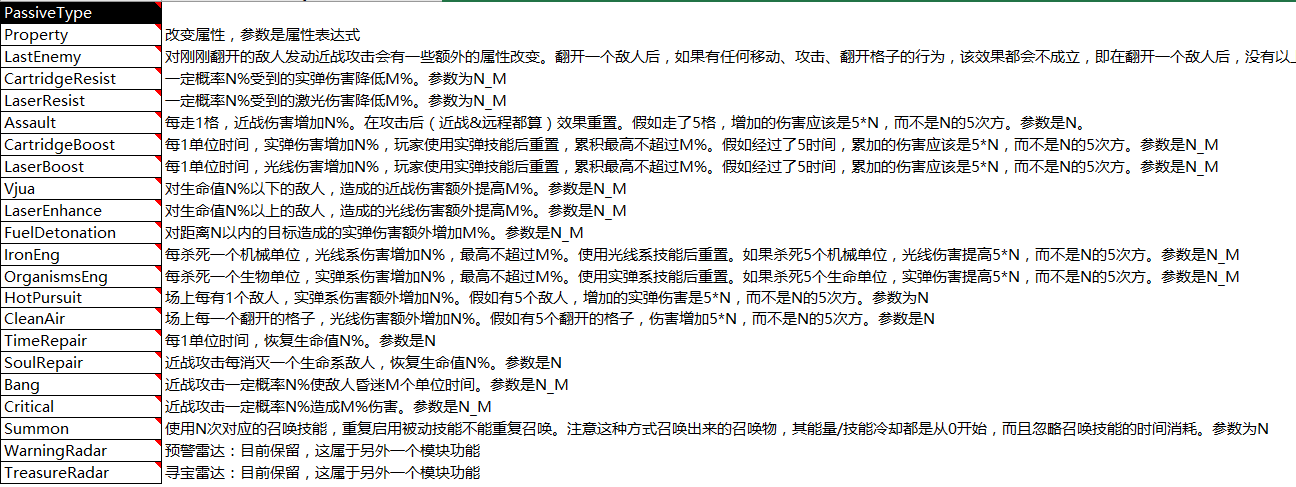
在这种情况下，召唤物是否能够copy宿主的技能，取决于宿主使用技能的能耗（usePower），如果召唤物能量足够，那么则可以copy技能。

**这里主要描述了召唤物的数据构成，其实在游戏中，召唤物有很多表现细节与目标选取规则，特别是后者，非常重要。之后在相应文档中进行描述。**

## 被动技能

### 作用类型

* passiveType，一维枚举数组。
* 被动技能使用后，可以产生一系列效果，把每个效果拆分后，就是PassiveType。



（上图看不全的，直接看注释内容吧…）

### 参数

* args，string一维数组，长度与passiveType相同。
* 作用（passiveType）的参数，根据不同的作用，解释成不同的内容。注意它的长度与要与passiveType一致，即便某些passiveType可能不需要参数。

## 状态

### Buff/Debuff

* isBuff，bool参数，true表示这个是增益。
* 标记该状态是增益还是减益，增益、减益不同，展示会有不同，例外，某些技能的驱散效果，也会判断增益、减益。

### 最大叠加

* max，int类型。
* 状态的最大叠加数量，某些状态是允许重复叠加的（比如增加属性类的），如果是，就等价于不能叠加。
* 关于状态的叠加/替换，在后续文档中详细描述。

### 持续时间

* time，int类型。

### 标记

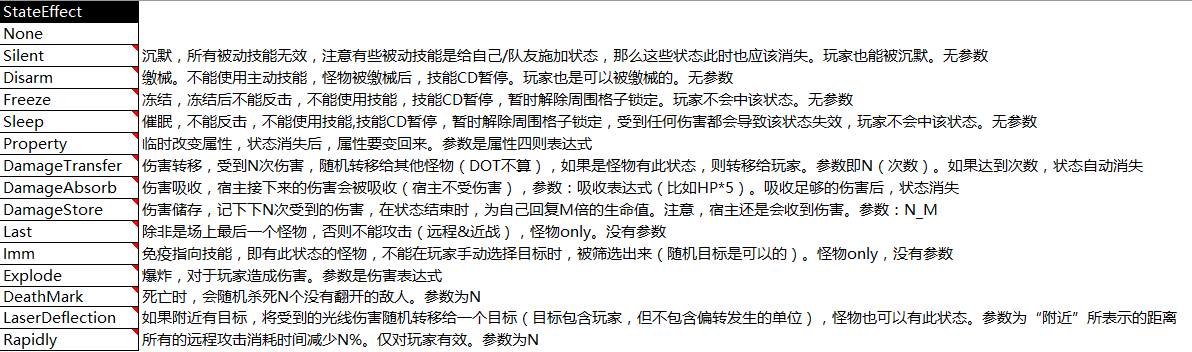
* mark，string类型，prefab名称。
* 状态的图标化表现。
* 和icon不同，mark还会包含一些数字型的表示方式。
* 一个怪物可以有N个状态，但不可能把所有状态图标全显示在怪物头上，所有会有一个交替显示机制。

### 状态优先级

* priority，int，值越大，优先级越高。
* 状态是在一个队列中依次计算，优先级越大，计算越优先。
* 不同优先级的状态，可能导致的计算结果是不同的。比如状态A：增加10生命值，状态B，增加50%生命值。

### 效果

* stateEffects，StateEffect一维数组。
* 状态最终效果由该字段决定。



* 因为一个状态可以有多个效果，所以这里是数组。

### 参数

* args，string一维数组。
* 效果的参数，其长度与效果（stateEffects）长度一致。

状态还有很多细节需要进一步说明（比如关于叠加，优先级），后续我会写说明文档，如果是即时型问题，就语音说明。