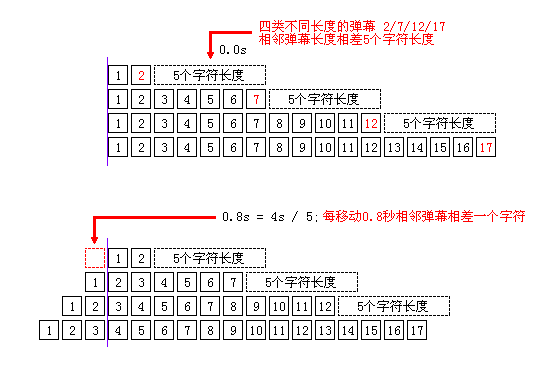
**MIKU滚动变换全解**

拖了好久我自己都不好意思了嗯，于是今日把这个大坑填了。

**一、基础原理**

滚动MIKU的实现是以不同长度的弹幕所造成的速度差实现的。



如图所示，其中所谓不同长度是指，相邻的长度差为5个字符。

而这就造成了以下的结果：

**移动每间隔4s/5即0.8秒，相邻长度差的弹幕有相对1个字符长度的位移。**

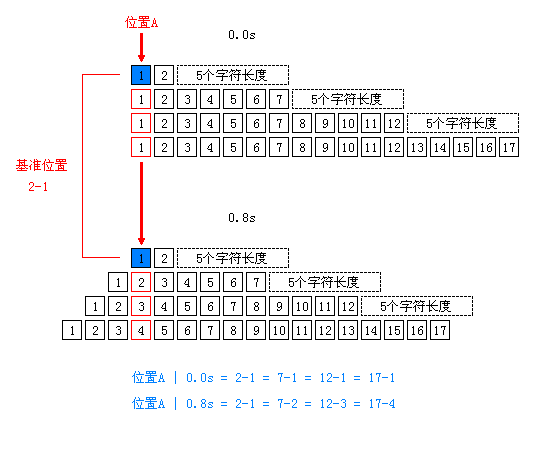
**正因为有了这1个字符长度的位移，我们就可以使用这个位移来进行遮盖，以达到变换的效果。而进行展示的时间点，则是0.8秒/1.6秒/2.4秒/3.2秒这四个时间点。**

这个结论可以有很大的扩展空间。在滚动MIKU的例子中，所有的弹幕分为4种最大横向长度，并且相邻的长度差为5个字符：它们分别为2个字符/7个字符/12个字符/17个字符。而作为显示面板的弹幕，横向长度为12个字符。

那么如果我做这样的设计：所有的弹幕分为5种最大横向长度，并且相邻的长度差为4个字符：他们分别为2个字符/6个字符/10个字符/14个字符/18个字符。而作为显示面板的弹幕，横向长度为14个字符。又会出现何种状态呢？答案如下：

相邻长度差为4个字符，移动每间隔4s/4即1秒，相邻长度差的弹幕有相对1个字符长度的位移。而进行展示的时间点，则是1.0秒/2.0秒/3.0秒这三个时间点。

也就是说，可展示的变换状态的个数，就是长度差的字符数减一。比如我要展示BILI，长度差就需要为5，展示MIO，长度差就需要为4，展示2DLAND，长度差就需要为7了。



接下来说具体的数据。

刚才说的1个字符长度的位移，这是**相对**的。比如说以上图为例，以2-1『注，2-1表示最大弹幕长度为2的弹幕中的第1个位置的字符，下面都遵循这个表达。有时候为了防止歧义我会在外面加上[]来表示。』为基准位置，在0.0s的时候，所有弹幕在Bilibili的播放器中是右对齐的，并且有一个初始距离。但这里我们要注重的只是相对位置：也就是说，**在0.0s的时候，所有弹幕的相对位置是0个字符**。而在0.8s的时候，由于我们将2-1作为了基准位置，则结果就是，2-1的位置所对应的应该是7-2、12-3、17-4这三个位置（在图中由红框表示）。

那么这个结果是如何计算出来的呢？因为运动是相对的，所以我们要设定一个基准长度。而这个基准长度一般来说是刚才所说的展示面板的长度。在滚动MIKU的例子中为12个字符。但是现在我们暂且以图中的2个字符为基准长度来进行计算——

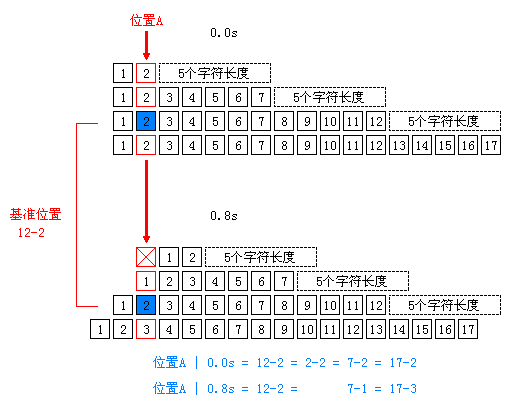
**目标位置 = 基准位置 + ((目标长度-基准长度) / 长度差)\*（当前时间点/时间间隔）**

也就是说，如果我把目标长度定为12，在图中的意思就是：

x = 1 + ((12 - 2) / 5) \* (0.8 / 0.8) = 1 + 2 \* 1 = 3

意思就是2-1在0.8s处所对应的目标位置为12-3

可能有点晕，那就再做一次。这次我们把基准长度设定成滚动MIKU所使用的长度：12个字符。



公式还是这个：

**目标位置 = 基准位置 + ((目标长度 - 基准长度) / 长度差)\*（当前时间点 / 时间间隔）**

这回我们从12-2变换为17-x好了。

即x = 2 + ((17 - 12) / 5) \* (0.8 / 0.8) = 2 + 1 \* 1 = 3

意思就是12-2在0.8s的时候对应的位置为17-3，如图中红框所示。

那么再来一次，这回从12-2变换为2-x看看。

即x = 2 + ((2 - 12) / 5) \* (0.8 / 0.8) = 2 + -2 \* 1 = 0

显然x不可能小于等于0，所以这个变换是失败的，因为目标字符不存在。在图中可以看到，以方格中的红叉表示。

那让我们再次进行扩展：假如当前时间点为2.4秒，基准位置为12-4，需要变换到7-x。

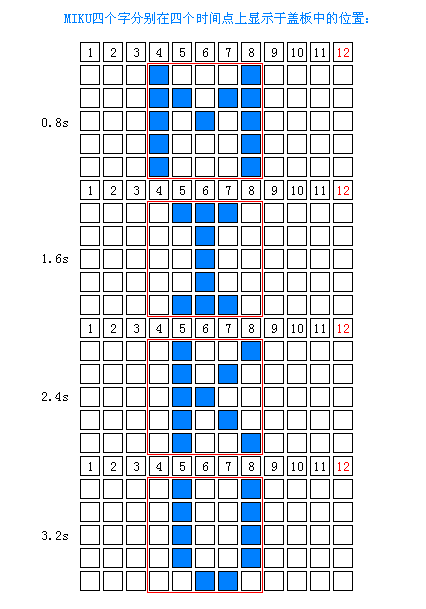
即x = 4 + ((7 - 12) / 5) \* (2.4 / 0.8) = 4 + -1 \* 3 = 1

也就是从12-4在2.4s所对应的位置为7-1。

可能光从这样的计算暂时还没法了解这个式子到底有什么深一层的含义，那就暂且先开始下一个部分好了。

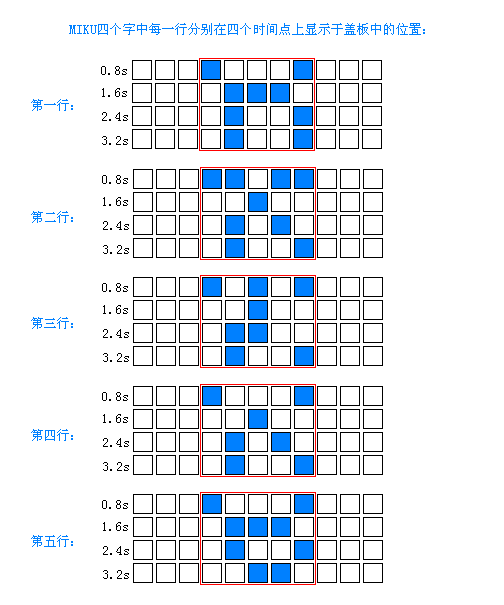
**二、变换目标**

在以前的分析报告里我好像写过盖板的样子，这里我就不再对盖板进行图示了。



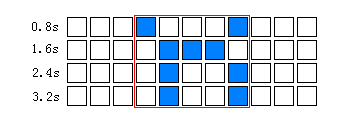
『注意，红色的框表示滚动变换的效果显示区域。也是盖板所留空的区域。』

但是这个图对我们来说是没有什么太大用处的……真正有用的图是下一张——



为什么要分行来处理，因为**行之间是互相独立的**。你可以先把第一行的变换处理了，再继续把之后其他行的变换处理了。而你所得到的每行的几条弹幕，可以**按照排列顺序把相同类型与长度的组合起来**，因为它们互不影响，但是**切记，保持每行所得到的弹幕结果的排列顺序**。

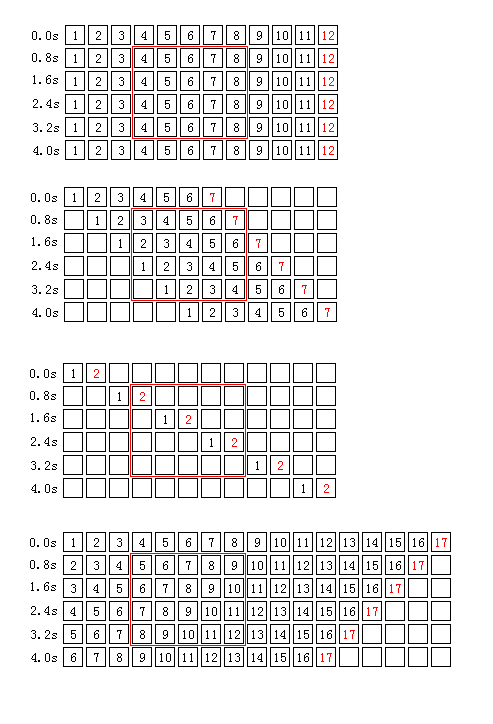
无论如何，让我们先从第一行开始处理吧。



对于滚动MIKU的第一行来说，我们想要得到的是如图所示的变换效果。如何才能做到让各种长度的弹幕交错组合来达成这样的变化效果，接下来的是重头戏。

**三、变换原理**

首先贴一张图。



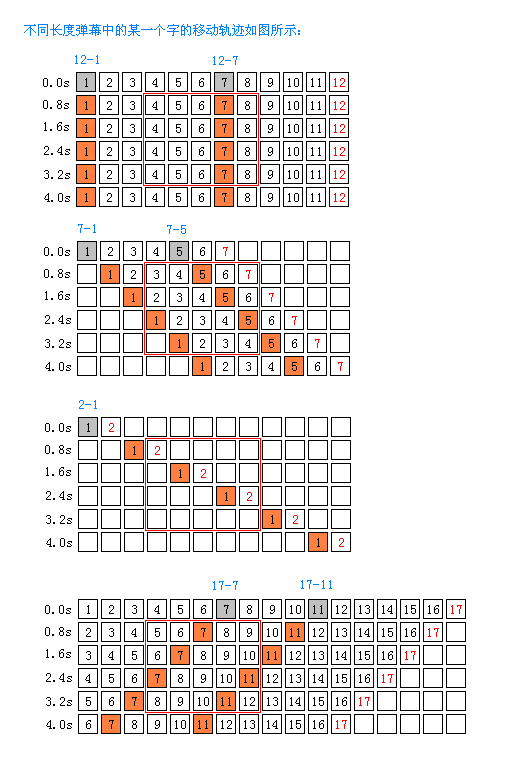
这张图的意思是，**对于12个字符这一基准长度**，其余长度所对应的字符位置的变化方式。

还记不记得第一节最后的公式？

**目标位置 = 基准位置 + ((目标长度 - 基准长度) / 长度差)\*（当前时间点 / 时间间隔）**

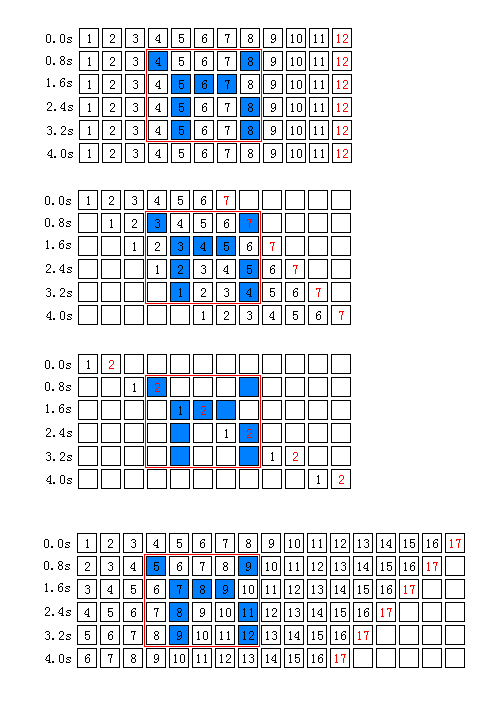
你会发现，**这些格子和其中的数字，就是以上这条公式的最直观表现（基准长度=12）**。

接下来的图，是表明了某个长度中的某个位置的字符，是以怎样的轨迹移动的。



这张图的意思是，**如果你在灰色格子中放入一个字符，这个字符在0.0s-4.0s这六个时间点上，将会以橙色格子的轨迹移动**。

然后让我们把第二节和第三节的图对应起来。



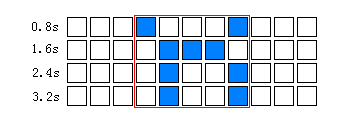
是不是有种恍然大悟的感觉w

还是说“老早知道了混蛋！快点给我接着讲！”w

最后要说的，是变换的核心部分。

**四、变换解析**

**滚动变换，说到底，是遮盖与显示互相交错的过程。**

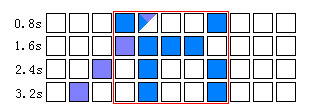


这张图是第二章中我说的，MIKU的第一行所要达到的目标。

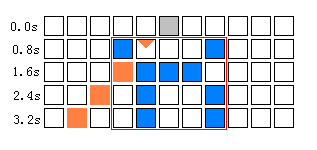
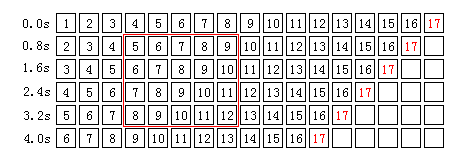
可以看到，图中的空白部分，**既可以没有弹幕，也可以用白色的弹幕遮盖**。

而图中的蓝色部分，**要以蓝色的弹幕来显示**。

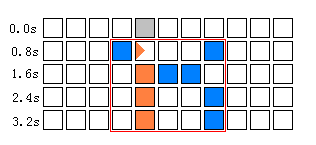
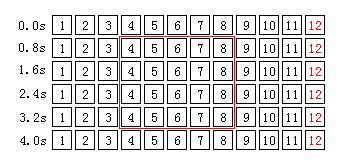
那么为了方便显示，我以**蓝灰色表示遮盖用的弹幕**，而以**蓝色表示显示用的弹幕**。而以**蓝色、蓝灰色、白色所组成的方格**，则表示**需要被遮盖的位置**。这个位置的意义是，**我需要一个显示字符来使变换过程完整，又要在0.8s的时候遮盖掉这个显示字符**。



这张图的意思是，**先以蓝灰色完成目标长度为17的遮盖轨迹**，如下图：



**再以蓝色完成目标长度为12的显示轨迹**，如下图：



为什么我能断言**遮盖用的是长度为17的弹幕，而显示用的是长度为12的弹幕？**因为**字符移动的轨迹**决定了**我应该以哪一种弹幕来实现。这点可以参考一下第三节中的第二张图。**

**综上所述，要达成MIKU第一行的变换结果，请按照上面那两张图的灰色位置，以红色框为标准位置，再以使用的弹幕长度去查找第三节中的那张表。对应的图我截过来了，在上面的右侧。**

你就会发现，**遮盖使用的是17-6**，而**显示使用的是12-5**。

于是第一次处理完成，现在的生成列表里有两条弹幕——

**这是第一次生成，我给予它一个Lv0的编号。**

**！！注意：Lv0的弹幕处于所有“为了实现第一行所生成的”弹幕的最上层！！**

**遮盖生成的是17-6**

**意思是用透明方格生成一条最大长度为17个字符的弹幕**

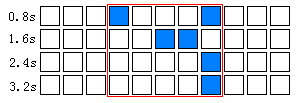
**其中横向第6个位置有一个遮盖用的白色方格**

**显示生成的是12-5**

**意思是用透明方格生成一条最大长度为12个字符的弹幕**

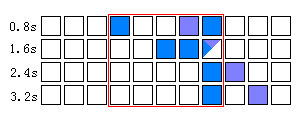
**其中横向第5个位置有一个显示用的蓝色方格**

那么第一次生成完毕后的结果如下图：



**可以看到刚才生成的部分都被挖去了。这是因为接下来的弹幕都在刚才那两条的下面，所以即使再有显示用的弹幕经过刚才的位置也无所谓，都已经被前一条弹幕的轨迹给挡掉了。**

以刚才的方法继续处理——



这回的结果如下——

**Lv1**

**遮盖生成的是7-6**

**显示生成的是12-8**

那么生成列表里有四条弹幕了——

**Lv0**

**遮盖的17-6**

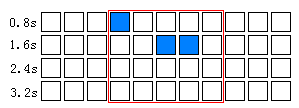
**显示的12-5**

**Lv1**

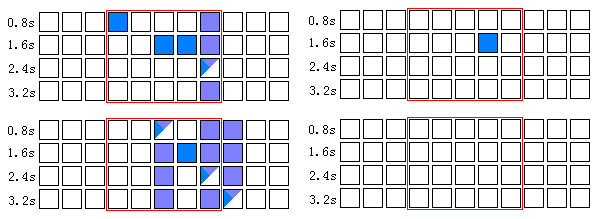
**遮盖的7-6**

**显示的12-8**

第二次生成完毕后的结果如下图：



那么剩余的我是以这样的方法实现的——



第三次：

**Lv2**

**遮盖生成的是12-8**

**显示生成的是2-2**

第四次：

**Lv3**

**遮盖生成的是12-6**

**遮盖生成的是12-8**

**遮盖生成的是12-9**

**显示生成的是7-5**

至此第一行的变换全部实现。现有的弹幕列表如下——

**Lv0**

**遮盖的17-6**

**显示的12-5**

**Lv1**

**遮盖的7-6**

**显示的12-8**

**Lv2**

**遮盖的12-8**

**显示的2-2**

**Lv3**

**遮盖的12-6**

**遮盖的12-8**

**遮盖的12-9**

**显示的7-5**

然后我们从上到下分析一下：

**由于Lv0中已经存在了12-8，所以之下的所有Lv中12-8都可以删除**

**由于Lv3中三条都是长度为12的弹幕，可以将它们合并。**

**！！注意：同Lv的相同长度的弹幕才能合并！！**

所以分析完成之后的弹幕列表如下——

**Lv0**

**遮盖的17-6**

**显示的12-5**

**Lv1**

**遮盖的7-6**

**显示的12-8**

**Lv2**

**显示的2-2**

**Lv3**

**遮盖的12-6、9**

**显示的7-5**

好，那么为了实现第一行的变换，我们需要如下的弹幕：

“　　　　　█　　　　　　　　　　　” 白色

“　　　　█　　　　　　　” 蓝色

“　　　　　█　” 白色

“　　　　　　　　█　　　” 蓝色

“　█” 蓝色

“　　　　　█　　█　　　” 白色

“　　　　█　　” 蓝色

然后说最后的注意事项——

因为，滚动变换中的弹幕都有着相同的出现时间，而XML读取器则是遵循“先读取先显示”的。所以最后创建的弹幕，才能遮盖掉先创建的弹幕。

也就是说，请**自下而上地创建弹幕。**即创建弹幕的顺序为生成顺序的逆转——

**显示的7-5** “　　　　█　　” 蓝色

**遮盖的12-6、9** “　　　　　█　　█　　　” 白色

**显示的2-2** “　█” 蓝色

**显示的12-8** “　　　　　　　　█　　　” 蓝色

**遮盖的7-6** “　　　　　█　” 白色

**显示的12-5** “　　　　█　　　　　　　” 蓝色

**遮盖的17-6** “　　　　　█　　　　　　　　　　　” 白色

哎呀好像超过预计页数了的样子……

第一行的结果已经出来了。你们可以拿设计器v2去模拟，也可以自己写XML来模拟。不过现在只做了一行的滚动变换，虽然结果是正确的但是看上去很不爽对吧……于是你们可以按照以上的步骤，把第二节第二幅图中的全部五行都做出来……然后把弹幕合在一起。

**！！这里还有个要点，行跟行之间的弹幕是没有任何关系的，你可以选择就这么直接叠加，也可以选择按照Lv的顺序将它们合并优化，这会很大程度上减少弹幕的数量，不过这不是必需的。暂时你们只要牢记每行的Lv不能改变就是了，如果愿意研究的可以自己试着去优化一下。！！**

然后，你们就能看到一个自己做出来的MIKU滚动变换啦w

**五、结语**

总结一下要点吧。

首先数学很重要啊。

其次物理也很重要啊。

最后做弹幕的伤不起啊。

好吧扯远了……

如果要做新的变换，比如汉字什么的。

第零：请把理论完全掌握了……

第一：画出第二节的第一幅图样。

第二：画出第二节的第二幅图样。

第三：逐行生成弹幕。

第四：将每行生成出来的弹幕进行优化。

第五：合并所有行的弹幕并优化。

第结束：生成弹幕内容。

还有，刚才我所说的汉字的限制，请自行体会w

**By**

**Azure22**

**静之籁**

**2011/05/29**

**谨以本文纪念我去年的三个通宵。**