《音乐拼块》程序设计说明

学习和使用 Music Blocks 编程的完整指南

《音乐拼块》是一个培养青少年对音乐和图像的兴趣的程序环境。《音乐拼块》扩展另一个程序，《乌龟拼块》，加上了一集关于音调和韵律的功能。

《音乐拼块》程序设计说明是一个开始学程序基础的好地方。 在这个说明书里，我们会让读者看看了解几个程序例子，来展示《音乐拼块》的音乐功能。

目录

许多说明里给的例子可以链接到可以执行的程序，只需注意 `RUN LIVE` 链接.

1. 开始

回去目录 | 下一章 (2. 发出声音)

《音乐拼块》是设计在游览器上使用。大多数的程序发展是在 Google Chrome 上做的，可是程序也应该在 Mozilla Firefox 上使用。 你可以从 github.io (http://sugarlabs.github.io/musicblocks) 执行程序， 或下载一个程序的复制，直接在自己的电脑上，使用文件系统执行下载的复制程序。

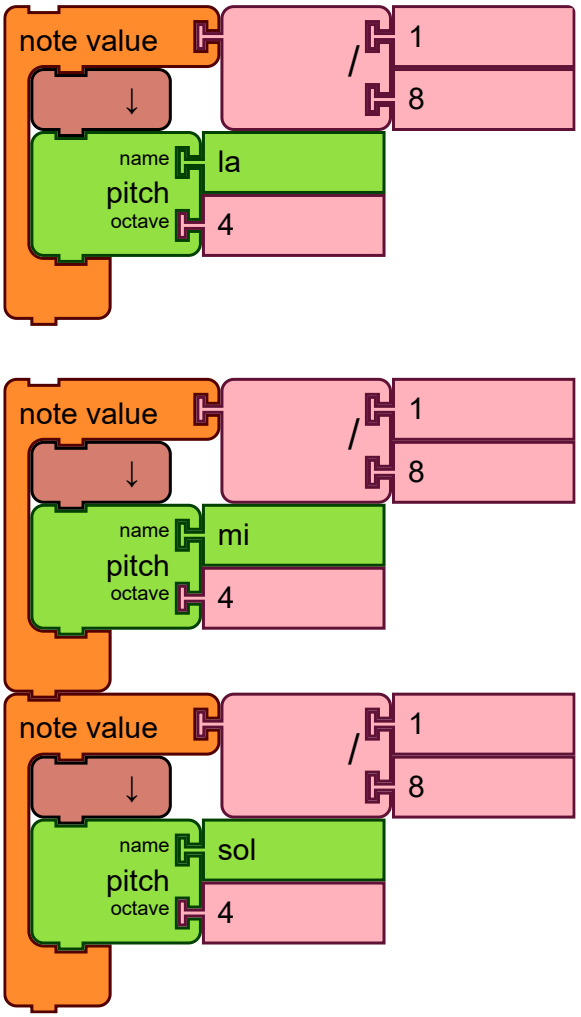
想知道更多关于《音乐拼块》的详情，你可以看 Using MusicBlocks (http://github.com/sugarlabs/musicblocks/tree/master/documentation). 想知道更多关于《乌龟拼块》的详情，你可以看 Using TurtleBlocksJS (http://github.com/sugarlabs/turtleblocksjs/tree/master/documentation).

2. 发出声音

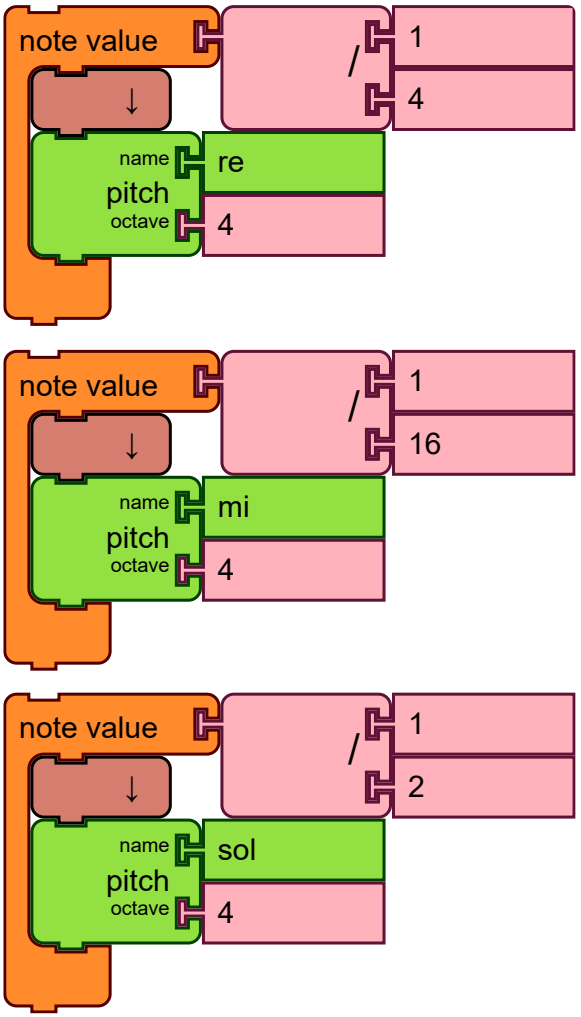
上一章 (1. 开始) | 回去目录 |下一章 (3. 使用音乐设计程序)

《音乐拼块》使用很多常见的音乐元素，相似音调, 音律,声音,和一点 乐器的选择。

2.1 《音乐拼块》最有用的拼块是《音符长度》。《音符长度》里面有一个音调拼块和音符的长度。

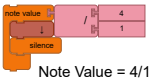
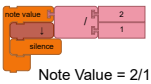
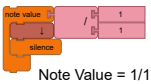
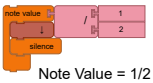
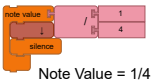
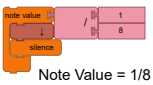
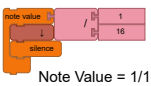
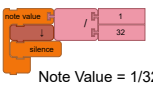
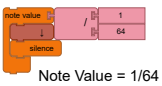
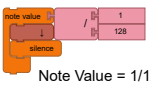


上面的例子有一个“音符拼块”。“1/8”是价值或长度，在这种情况下是一个八分音符下面有两个连接的音符。这些音符是八分音符，一起做一个四分序列。



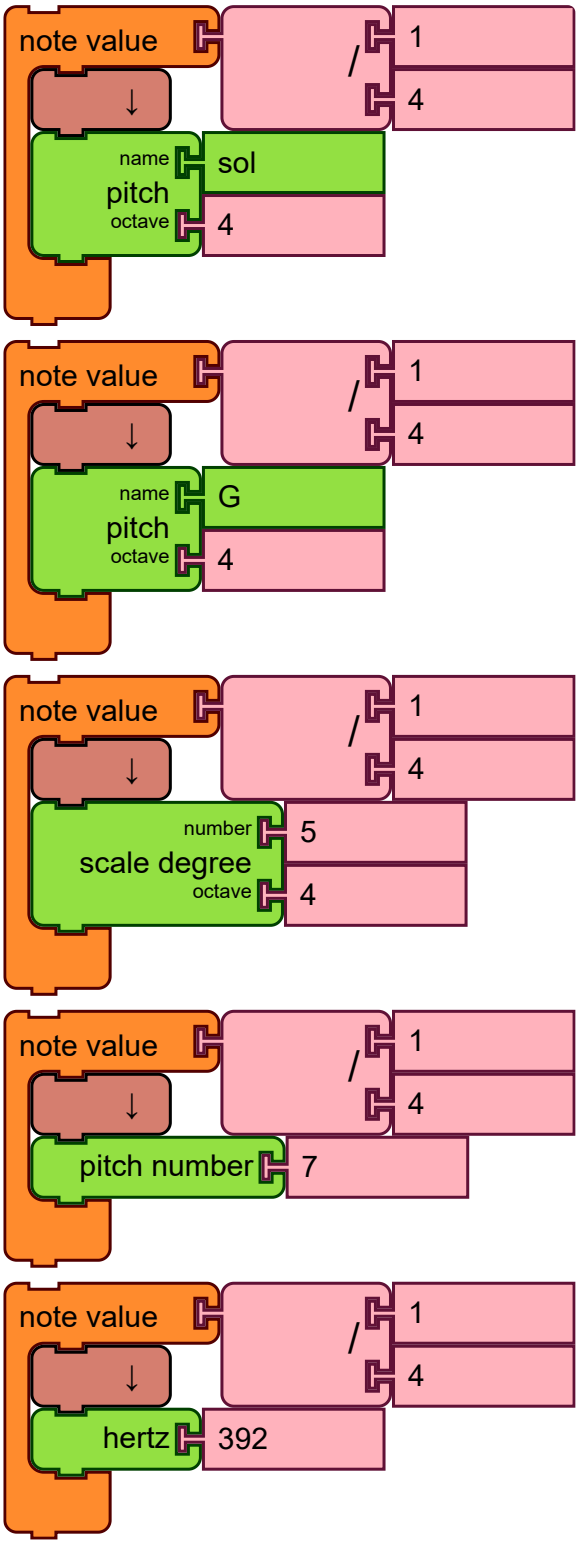
这个例子有不同的音符。从上到下有: 1/4 为一个四分音符， 1/16 为一个十六分音符，和 1/2 为一个半音音符。

《音符长度拼块》可以使用任何数学运算。

Note Value Blocks	Western Notation (Notes)	Silence Blocks	Western Notation (Rests)
 Note Value = 4/1	 Longa Note	 Note Value = 4/1	 Longa Rest
 Note Value = 2/1	 Breve Note	 Note Value = 2/1	 Breve Rest
 Note Value = 1/1	 Whole Note	 Note Value = 1/1	 Whole Rest
 Note Value = 1/2	 Half Note	 Note Value = 1/2	 Half Rest
 Note Value = 1/4	 Quarter Note	 Note Value = 1/4	 Quarter Rest
 Note Value = 1/8	 Eighth Note	 Note Value = 1/8	 Eighth Rest
 Note Value = 1/16	 Sixteenth Note	 Note Value = 1/16	 Sixteenth Rest
 Note Value = 1/32	 Thirty-second Note	 Note Value = 1/32	 Thirty-second Rest
 Note Value = 1/64	 Sixty-fourth Note	 Note Value = 1/64	 Sixty-fourth Rest
 Note Value = 1/128	 Hundred twenty-eighth Note	 Note Value = 1/128	 Hundred twenty-eighth Rest

请使用上面的图片作为表示值。

2.2 音调。我们已经看到了音调拼块在音符长度拼块里面。 音调拼块指定音调的名字和音高八度。



有很多方法来指定一个音调拼块音高八度和音名。上面有例子 -

上面的音调拼块是使用唱名指定(`Sol` 在 八度-四)，里面有音符 `Do Re Me Fa Sol La Ti` 。

下一个拼块有用音高名称指定的音高。(`G` 在 八度-四)，里面有音符 `C D E F G A B` 。

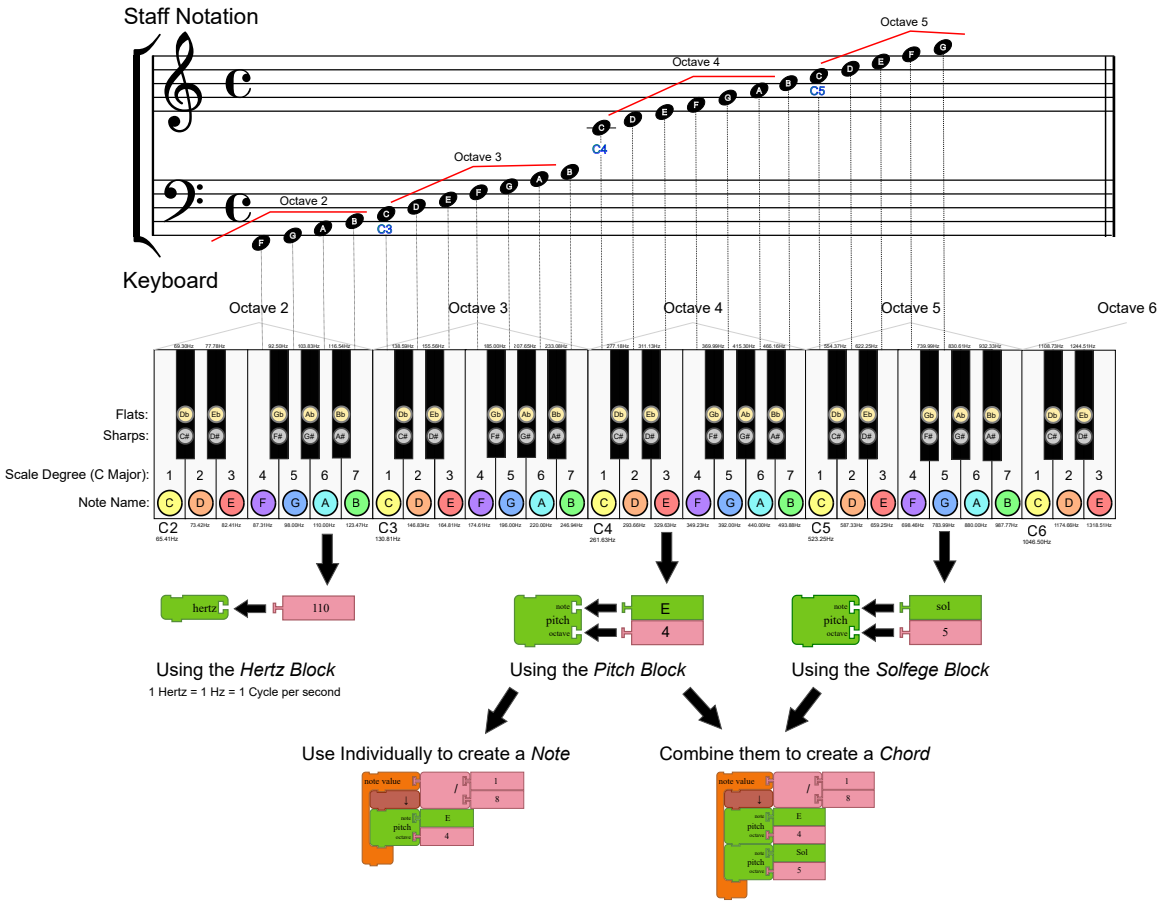
下一个拼块有用音阶程度拼块(音阶中的第五个音符，'G'，也在 八度-四)，`C == 1, D == 2, ...`

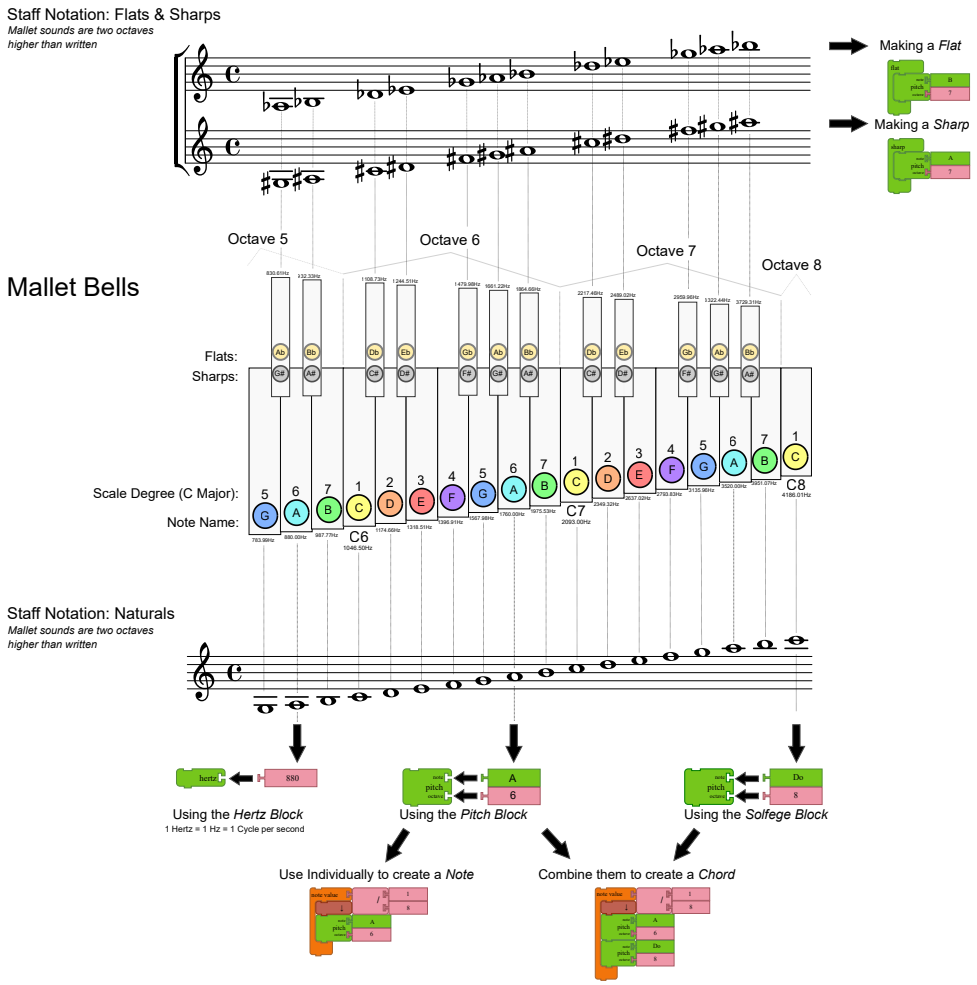
下一个拼块有用音音调编号拼块。(c 上面的第七个半音，也在 八度-四)。音高数字偏移可以用《设置音高数字偏移拼块》固定。

最后一个拼块的音调用赫兹和一个《数字拼块》指定。赫兹是频率的度量。

八度是用《数字拼块》指定的。这个数字不能有小数。 如果音高由赫兹指定，八度将不会被使用。

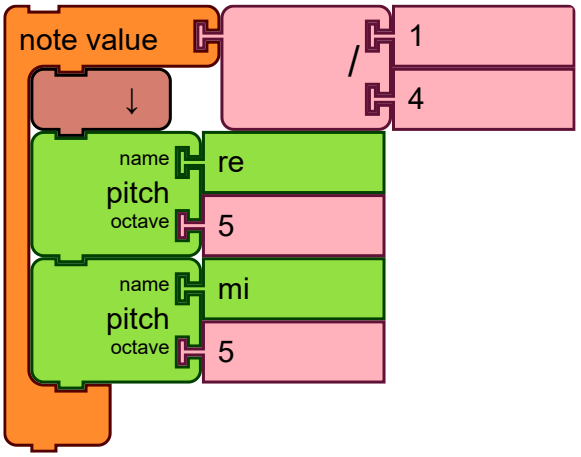
最后，音高名称可以用一个文字拼块指定。





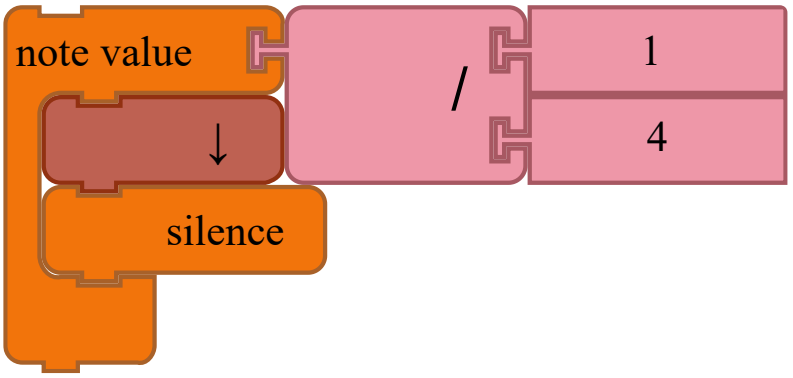
请参考上面的图表，了解音符在键盘或五线谱上的位置。

2.3 和弦



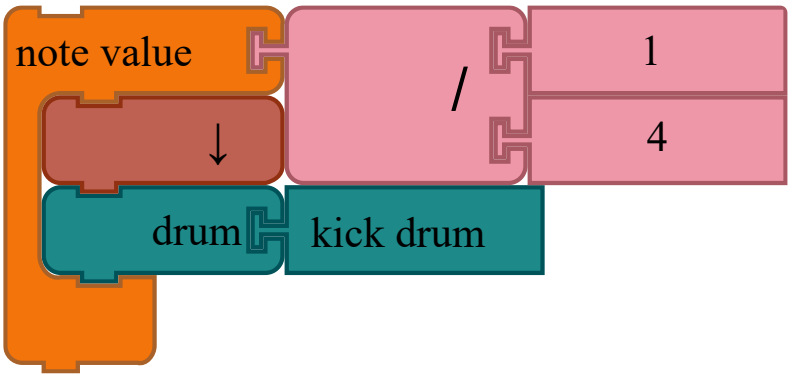
一个和弦可以通过将多个音调拼块放在一个音长度拼块里面。

2.4 休止

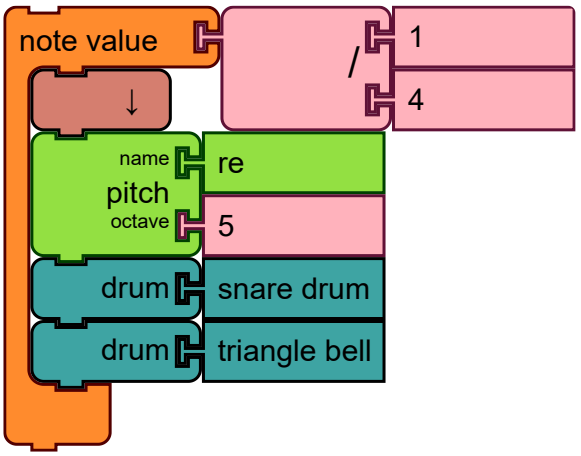


休息可以使用一个音符长度拼块。在这个拼块应该有一个《安静拼块》而不是一个音调拼块。

2.5 鼓声



一个音调拼块可以在任何地方使用—例如，在音符长度里面—一个鼓样品可以用来代替。 现在大概有二十多个不同的样本可选择。默认鼓是踢鼓。



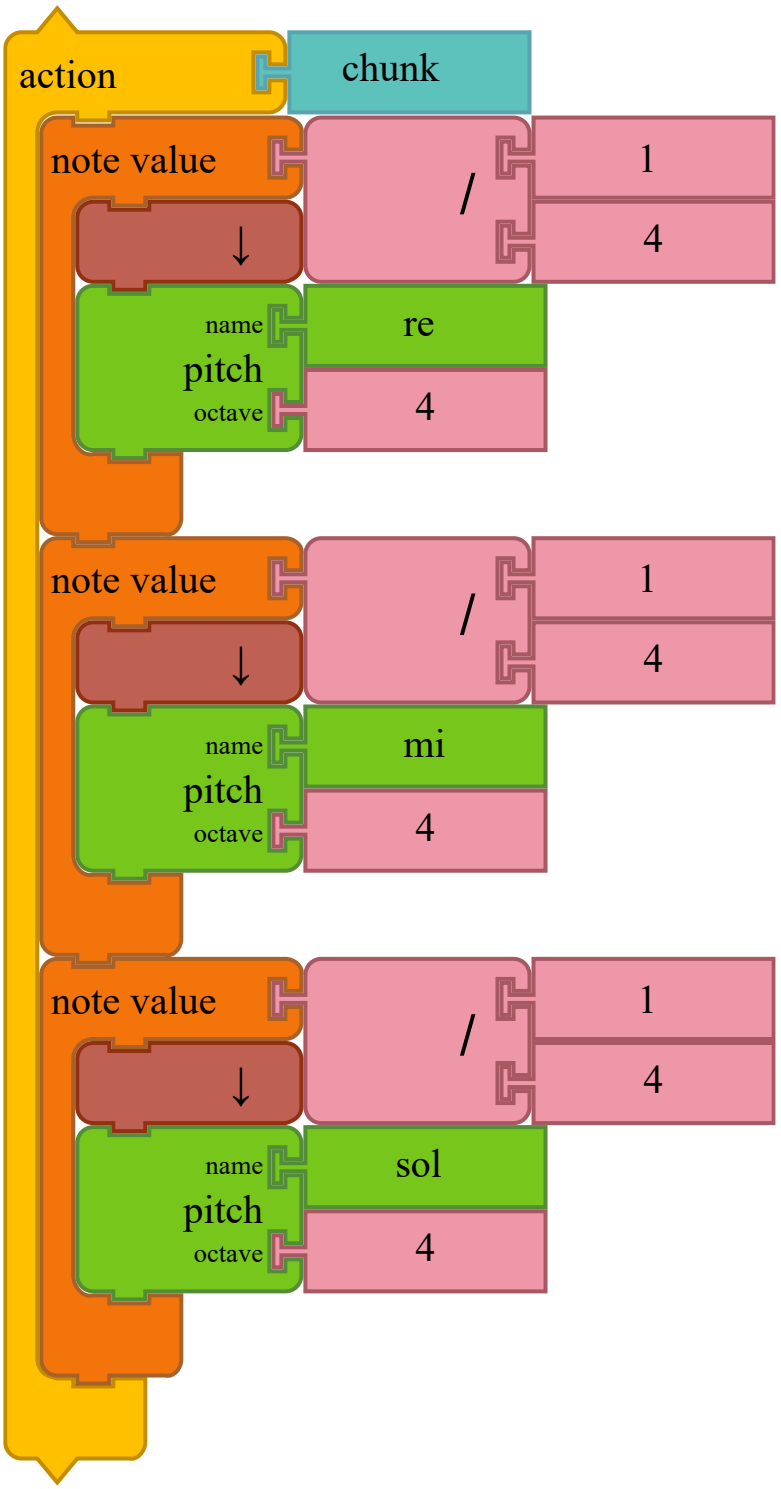
上面有一个和弦的例子。这个例子的和弦使用鼓拼块。

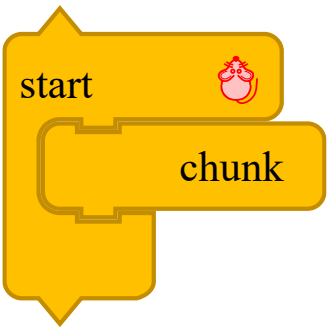
3. 使用音乐设计程序

上一章 (2. 发出声音) | 回去目录 | 下一章 (4. 部件)

这一章将会说明怎么使用音乐砖块来产生音乐。 一件必须注意的事是你可以使用自己做出来的音乐砖块来做出你的程序， 或使用 *音调-时间矩阵* 开始做出你的程序.

3.1 砖块


action
chunk
note value
/
1
4
name
pitch
octave
re
4
note value
/
1
4
name
pitch
octave
mi
4
note value
/
1
4
name
pitch
octave
sol
4

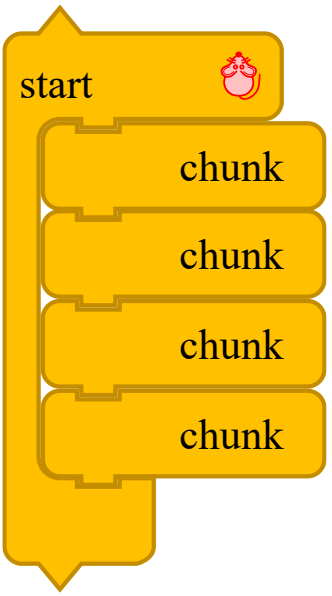


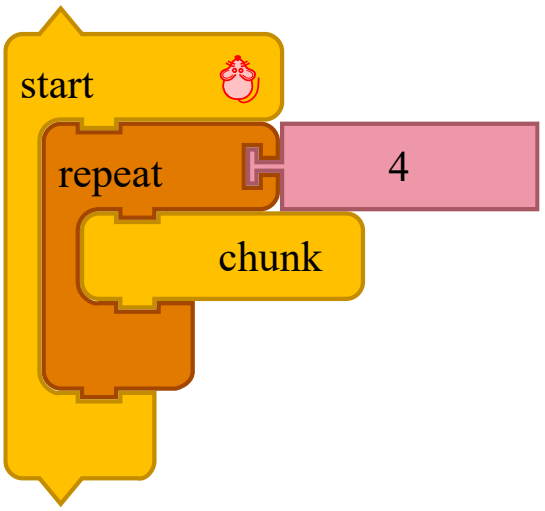
每当你做出一个新的 *Action* 程序堆，《音乐拼块》会产生一个新的拼块。 这个拼块会是独特的，也会链接程序堆。 (这个新的拼块可以在屏幕的左角， *Block* 的部分里找的) 按上和执行这个拼块和执行以上的程序堆是一样的。 这些新的拼块创造时的名字是 `chunk` , `chunk1` , `chunk2` ... 可是你可以编改 *Action* 拼块的标签来换拼块的名字。

一个 *Action* 拼块包含了一系列的行动，这些行动只有当拼块被其他拼块提名才会执行。 例如一个*Start*拼块。 这个特性有用于产生更复杂的音乐程序。

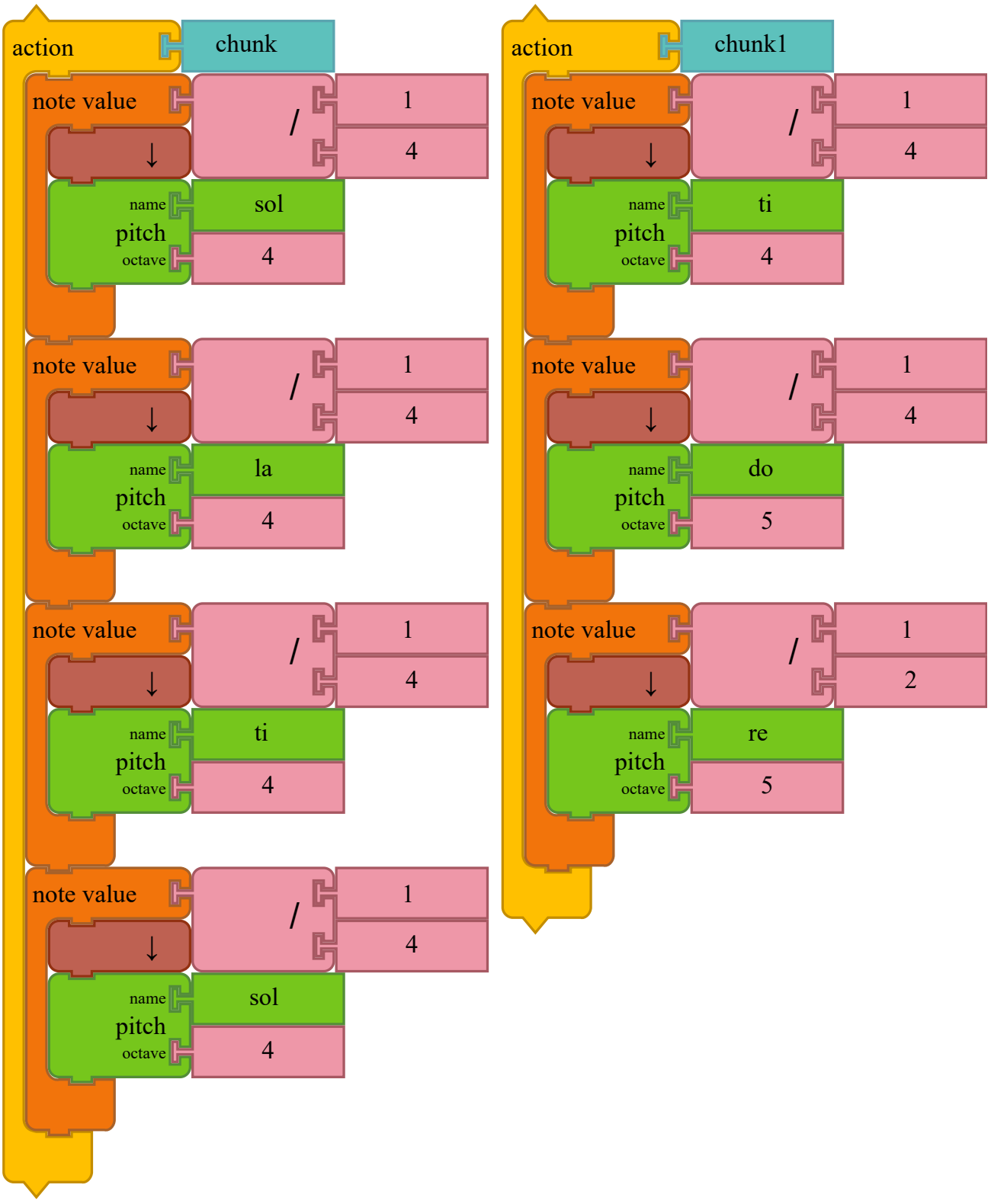
一个 *Start* 拼块 是一个在按下开始按钮后，直接执行的 *chunk*。 这个拼块是大多数程序开始的地方。 《音乐拼块》包含着很多可以 *Run* 一个程序的方式: 你可以按下在屏幕左上角的 *Run* 按钮("兔子" 图标) 快速播放音乐; 按下 *Run Slow* 按钮 ("乌龟" 图标) 慢速播放音乐; 和按下 *Step* 按钮 ("蜗牛" 图标) 每按下按钮播放程序的一个拼块。

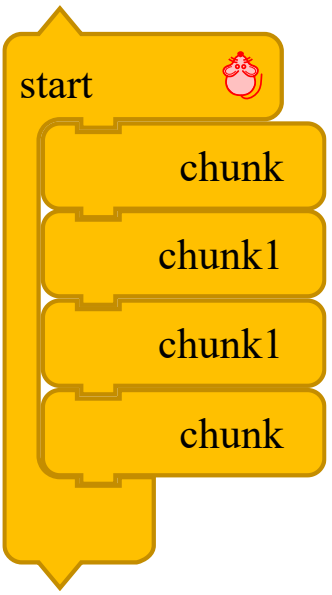
在上面的程序例子， *Chunk* 拼块在 *Start* 拼块里面。 这代表每当其中一个开始按钮按下之后，*Start* 拼块里面的程序 (*Chunk* 拼块) 会被执行。 之后，你可以在 *Start* 加更多拼块, 顺序的执行拼块里的程序。



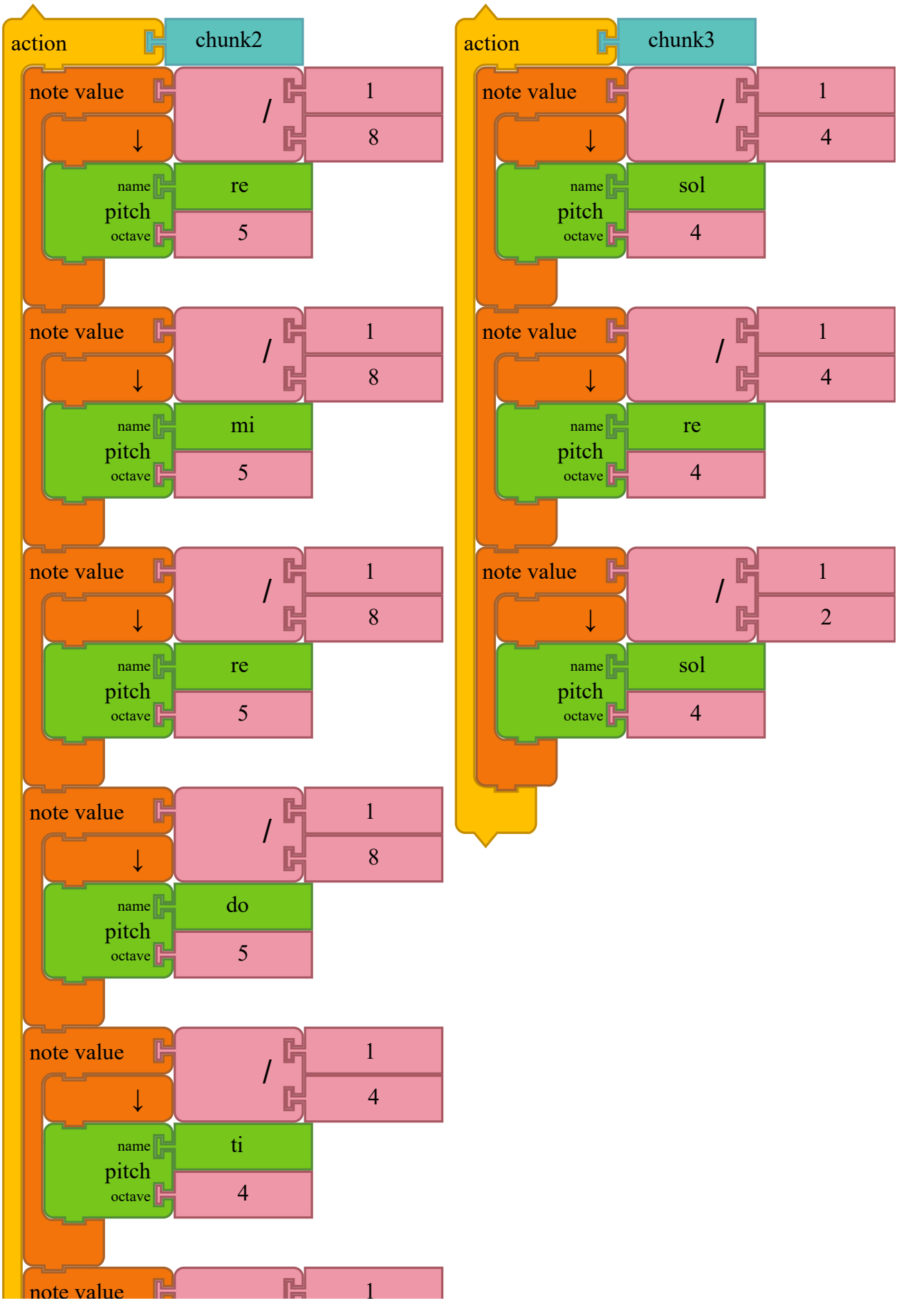


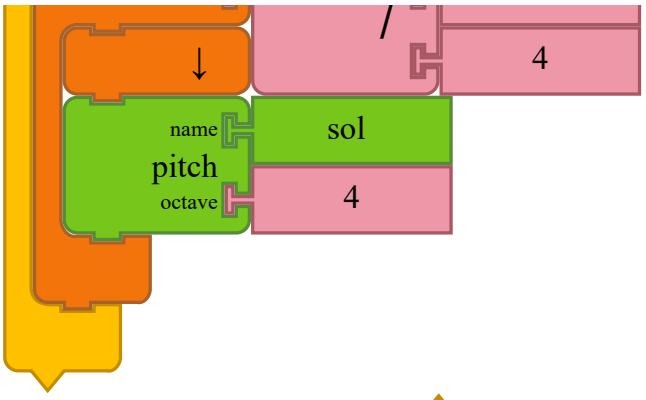
你可以使用更多 *Chunk* 拼块或使用 *Repeat* 拼块 重复 程序。

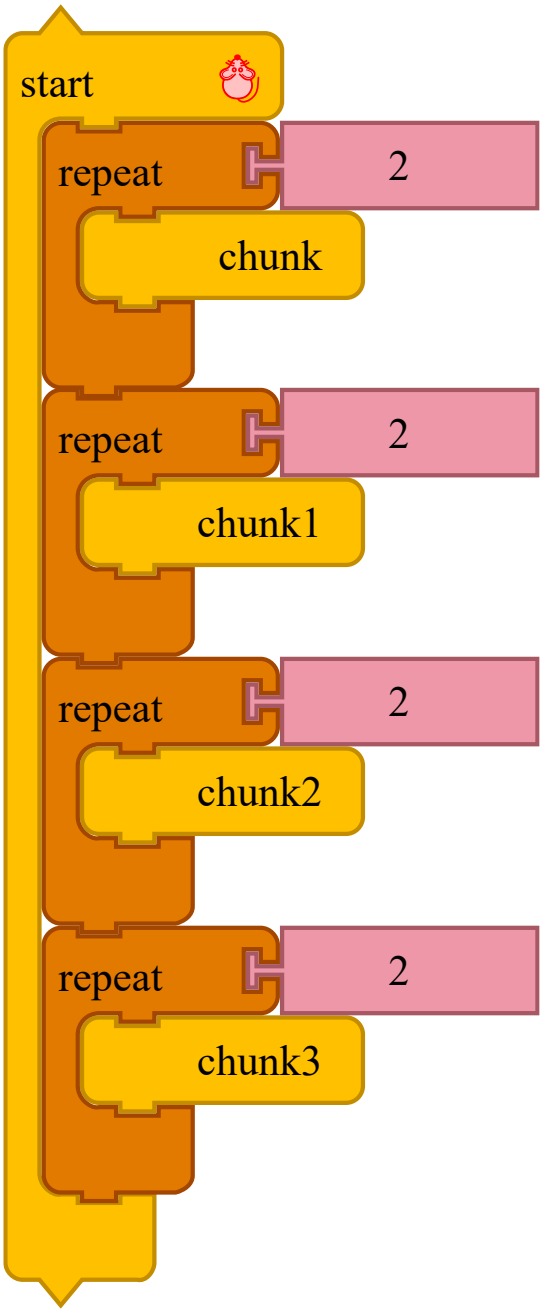

action
chunk
note value
/
1
4
name
pitch
octave
sol
4
note value
/
1
4
name
pitch
octave
la
4
note value
/
1
4
name
pitch
octave
ti
4
note value
/
1
4
name
pitch
octave
sol
4
action
chunk1
note value
/
1
4
name
pitch
octave
ti
4
note value
/
1
4
name
pitch
octave
do
5
note value
/
1
2
name
pitch
octave
re
5



你也可以混合不同的拼块。 在上面的程序例子， 我们执行 "chunk" 的行动拼块, 接着我们两次执行 "chunk1" , 然后又执行 "chunk".


action
chunk2
note value
/
1
8
name
pitch
octave
re
5
note value
/
1
8
name
pitch
octave
mi
5
note value
/
1
8
name
pitch
octave
re
5
note value
/
1
8
name
pitch
octave
do
5
note value
/
1
4
name
pitch
octave
ti
4
note value
1
action
chunk3
note value
/
1
4
name
pitch
octave
sol
4
note value
/
1
4
name
pitch
octave
re
4
note value
/
1
2
name
pitch
octave
sol
4



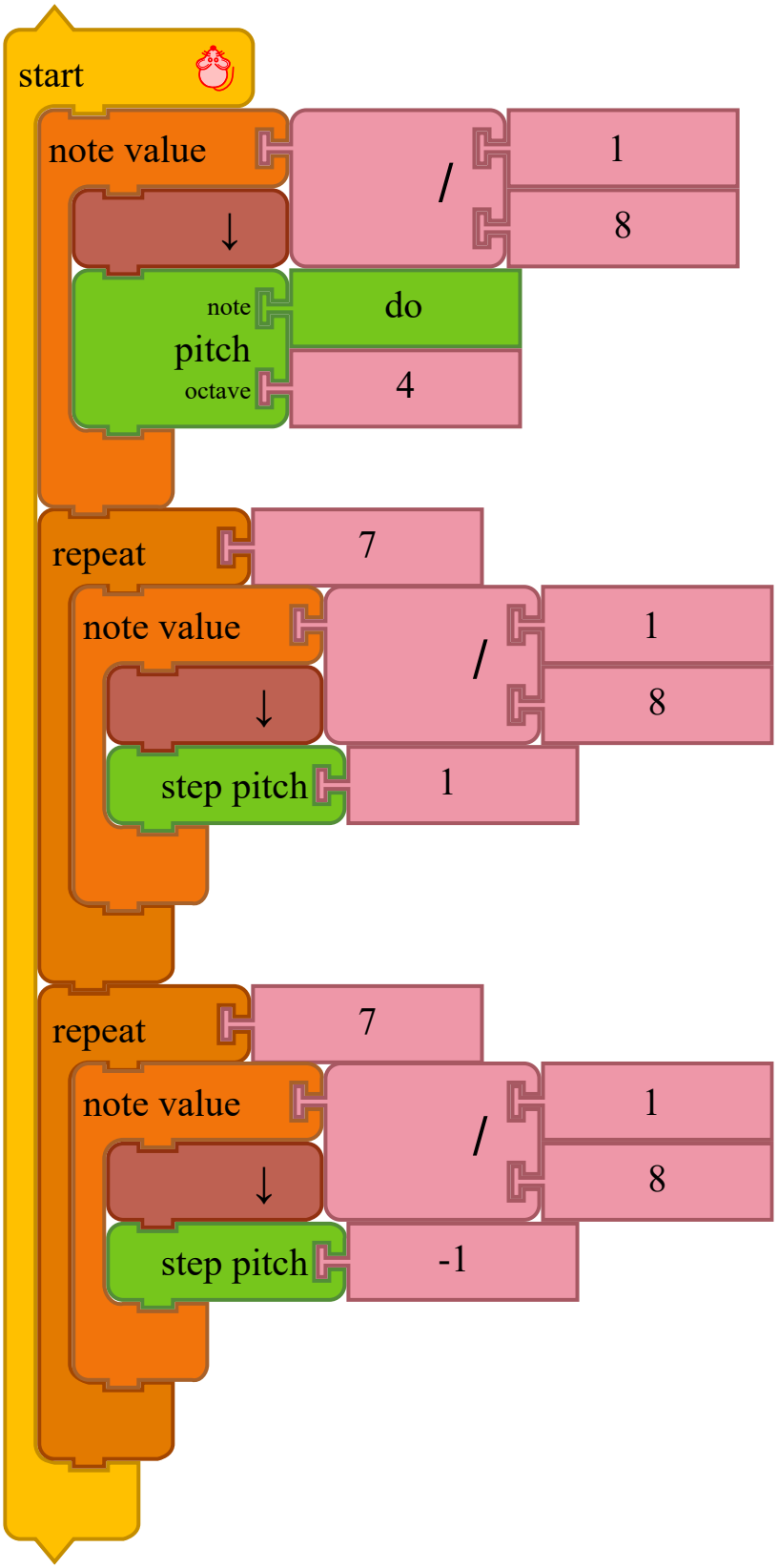


假如我们有更多拼块，我们就可以创造一个歌曲. (你可以猜猜程序的后果吗? 你对我们制造的歌曲熟悉吗?)

3.2 音乐转变

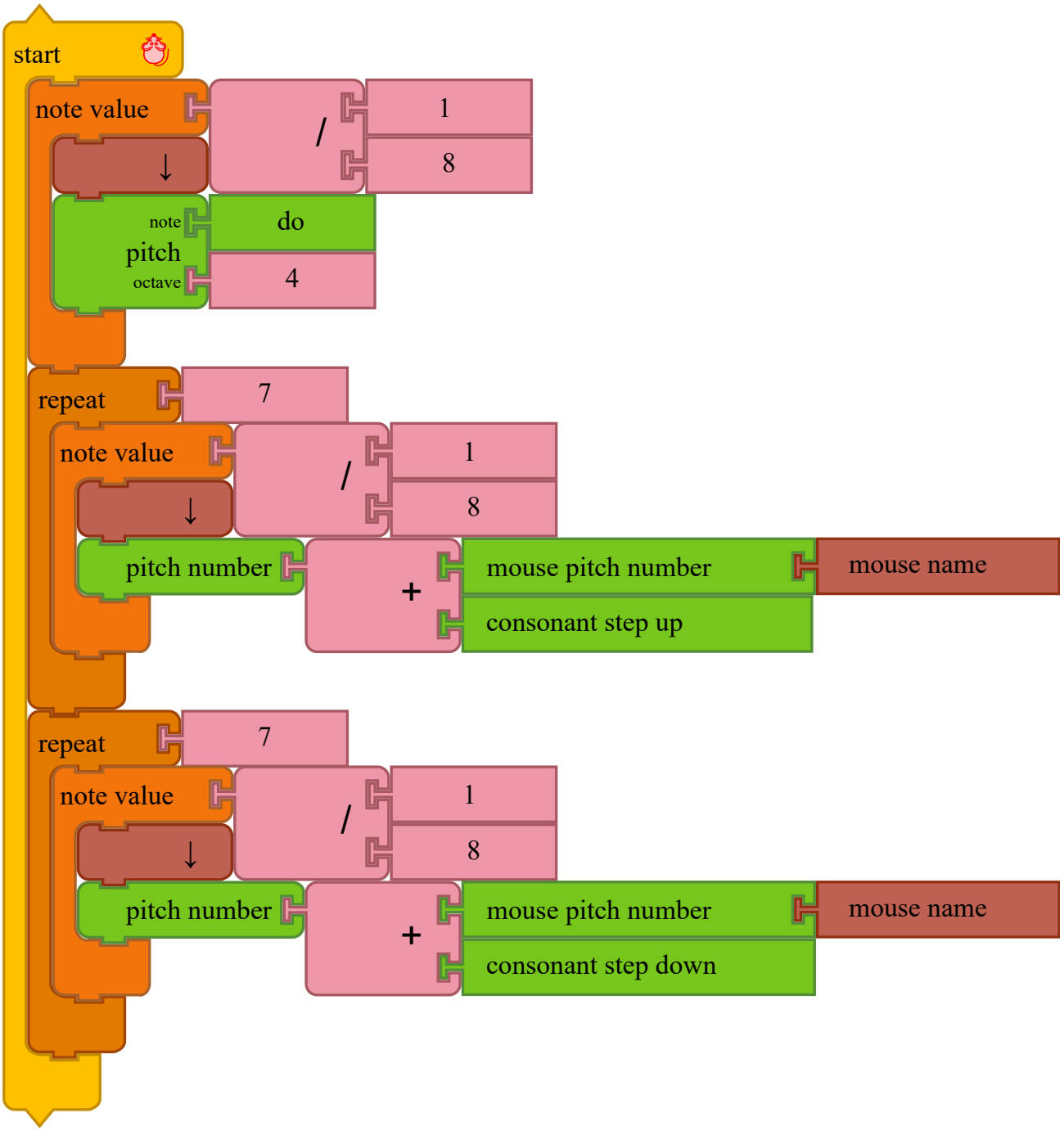
《音乐拼块》拥有不同的方式改变音调，节奏和其他声音的特征.

3.2.1 转变音调



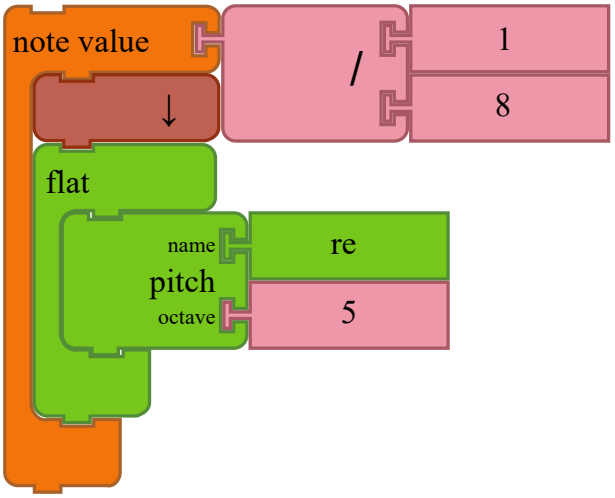
Step Pitch 拼块从上一个音符提高或降低下一个音符。 在上面的例子里， *Step Pitch* 拼块在 *Repeat* 拼块里使用，七次重复里面的程序, 发出一个音阶的不同声音.

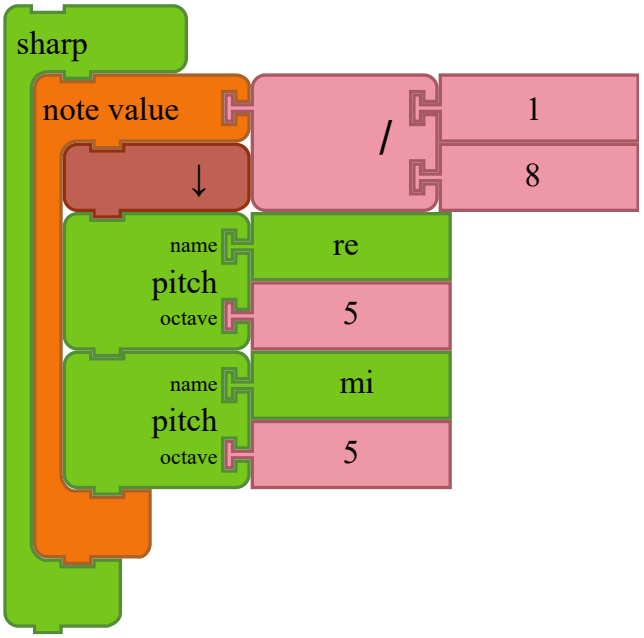
RUN LIVE (https://musicblocks.sugarlabs.org/index.html?id=1523032034365533&run=True)



另外一个提高或降低音阶里的音符就是使用*Consonant Step Up* 和 *Consonant Step Down* 拼块。 这两个拼块会在现在的模式计算到下一个音符有几个半步。 (想知道更多关于 音乐模式 可以看下面。) 注意 *Mouse Pitch Number* 拼块会给回上一个音符的音调号码。

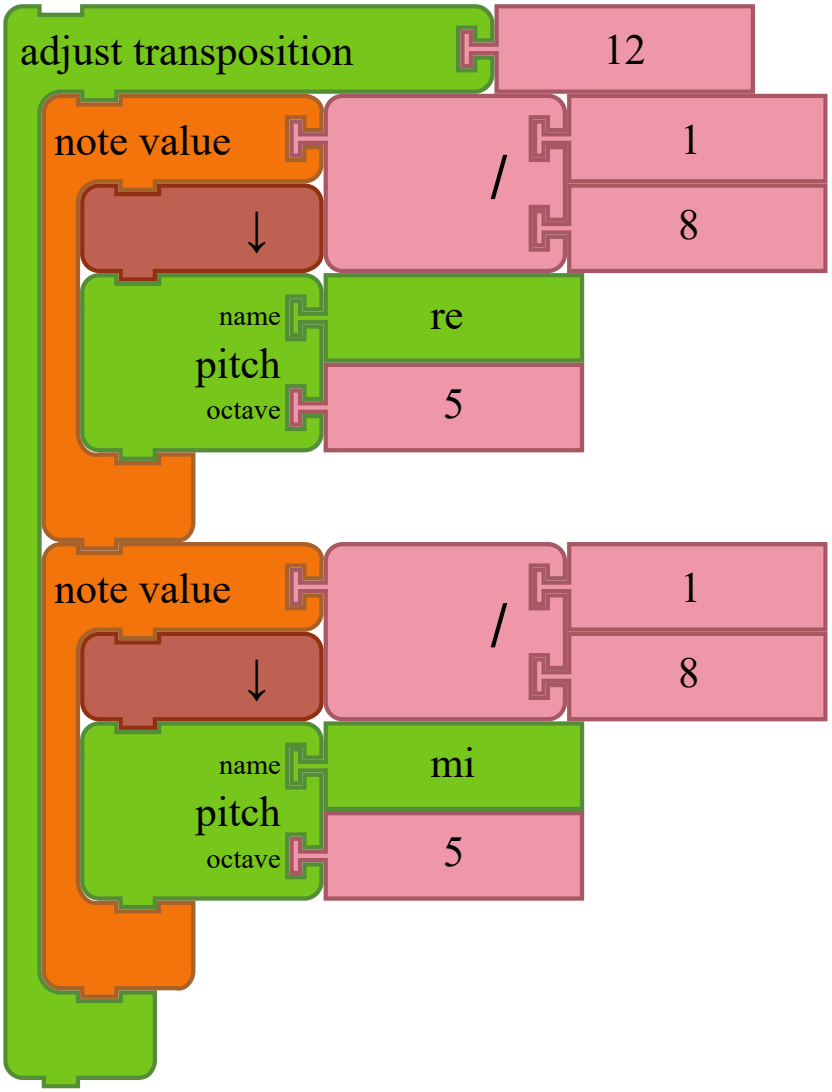
3.2.2 升号和降号



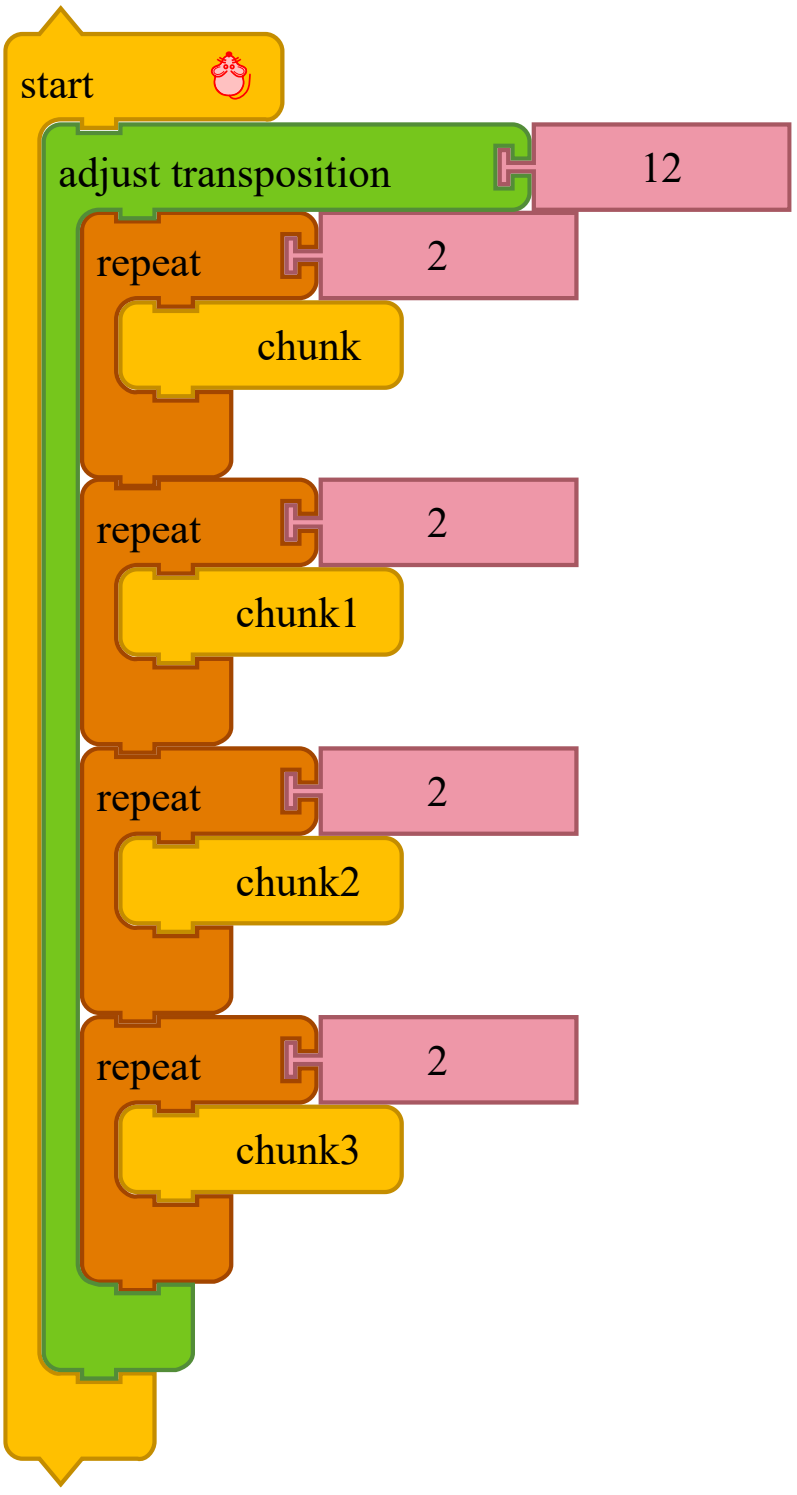


Sharp 和 *Flat* 拼块可以包住 *Pitch* 拼块, *Note value* 拼块, 或 行动拼块. 一个 “sharp” 会把音调提高半步， 而一个 ”flat“ 会把音调降低半步。 在上面左边的例子里， 只有 ”Mi“ 的 *Pitch* 拼块音调降低半步; 在上面右边的例子里, 两个 *Pitch* 拼块音调提高半步。

3.2.3 移调法

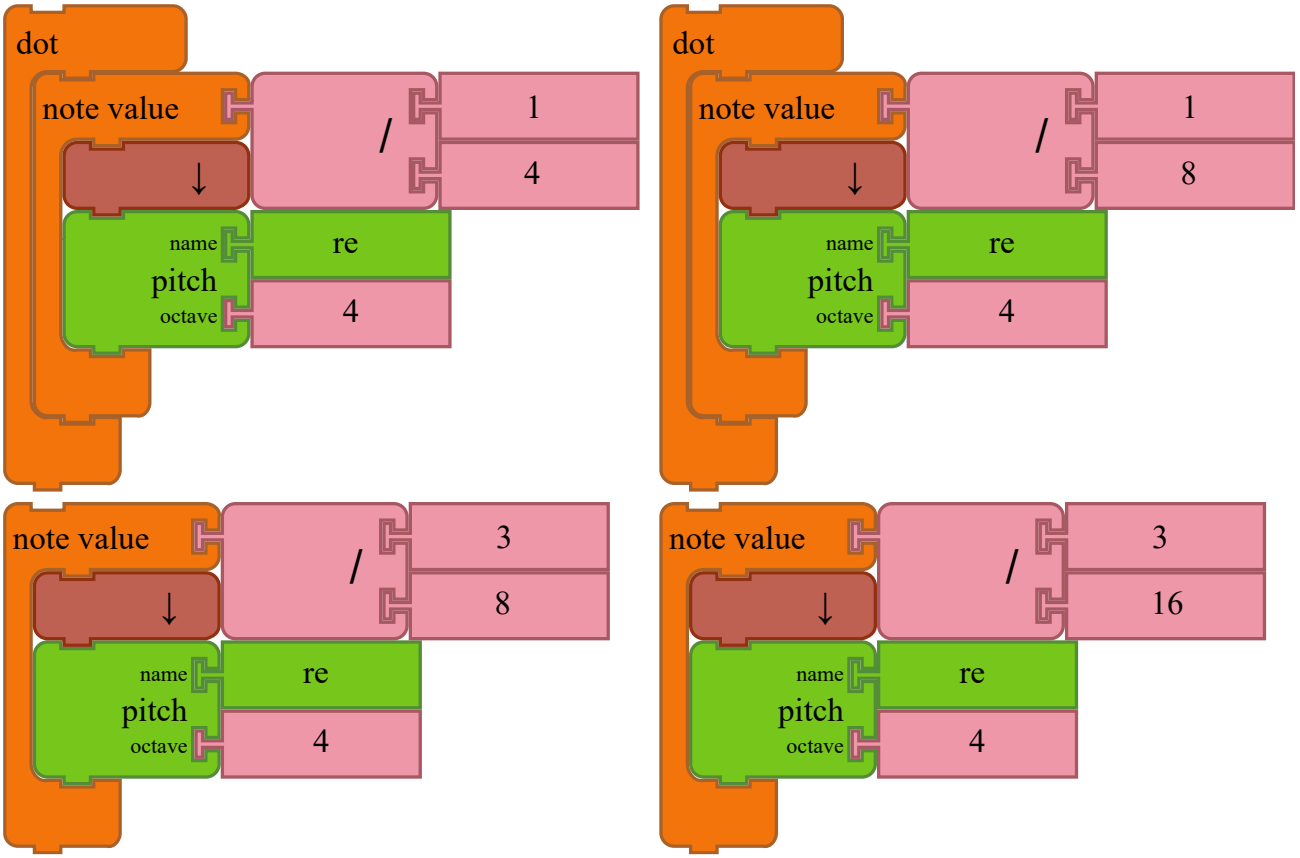


Adjust-transposition 拼块可以用来做出更大音调的改变。 一个正整数会把音调提高，而一个反整数会把音调降低。 如果要改变一个八度， 提高 12 个半步. -12 会降低一个八度。



在上面的例子，我们把上次设计的歌曲程序提高一个八度。

3.2.4 附点音符



你可以使用 *Dot* 拼块来 "dot" 音符. 一个被点到的音符把音符的时间加长50%。 例如，一个被点到的四分音符会播放一个拍的 3/8(1/4 + 1/8)， 而一个被点到的音符会播放一个拍的 3/16 (1/8 + 1/16).

你也可以换音符播放的时间来代替一个被点到的音符，例如把音符播放时间定为 3/8 (一个被点到的四分音符).

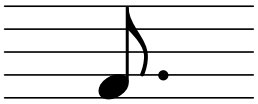
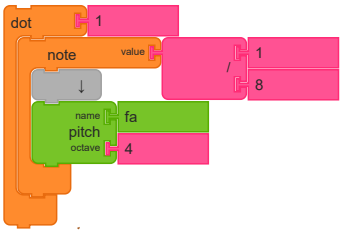
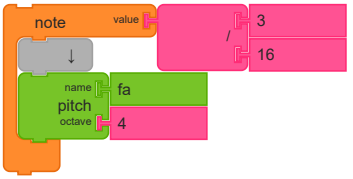
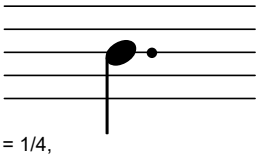
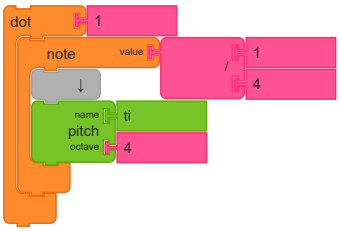
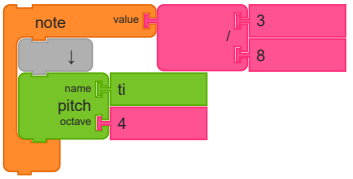
Using Dotted Notes

The dot increases the value
of a note by half of its value.

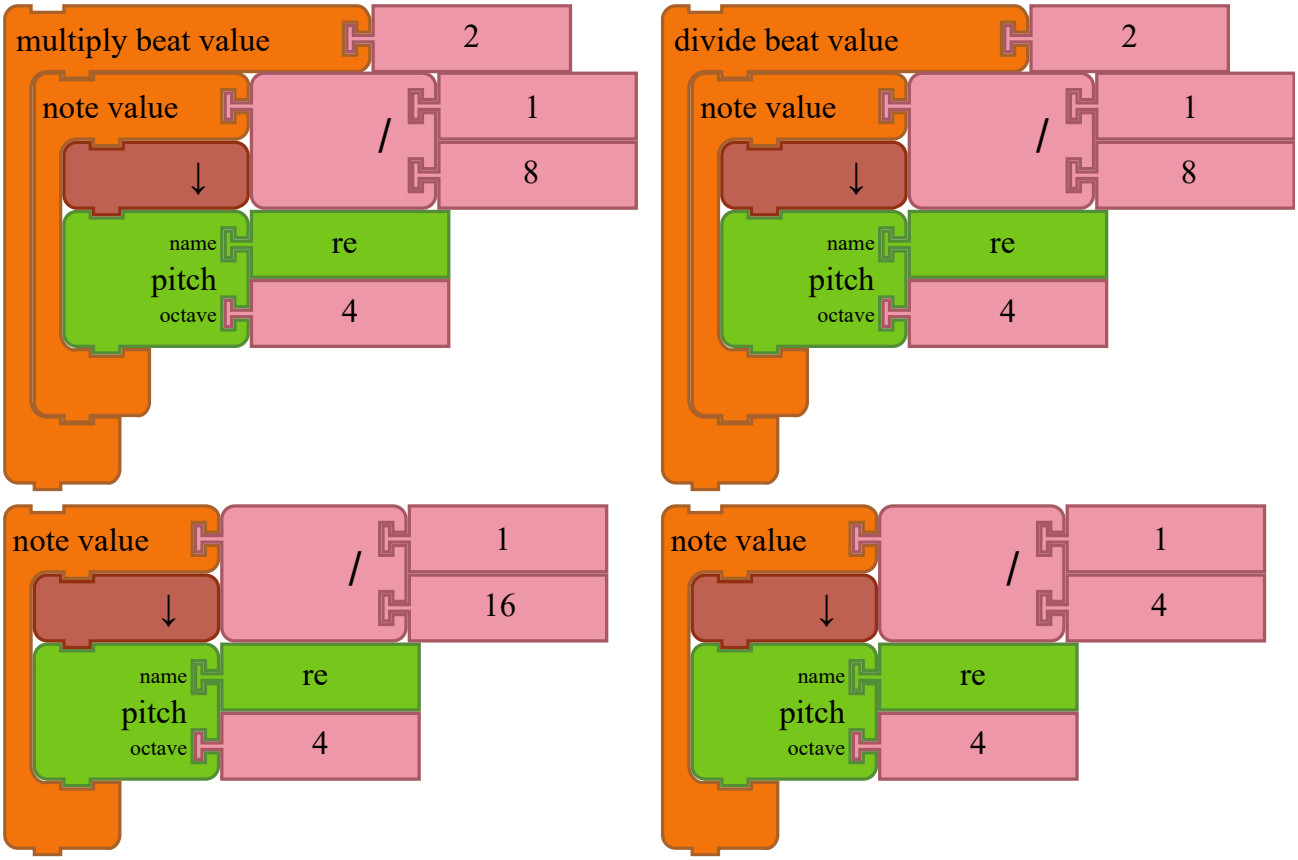
x= value of note

Formula: $x + \frac{x}{2} =$ value of dotted note

Examples:

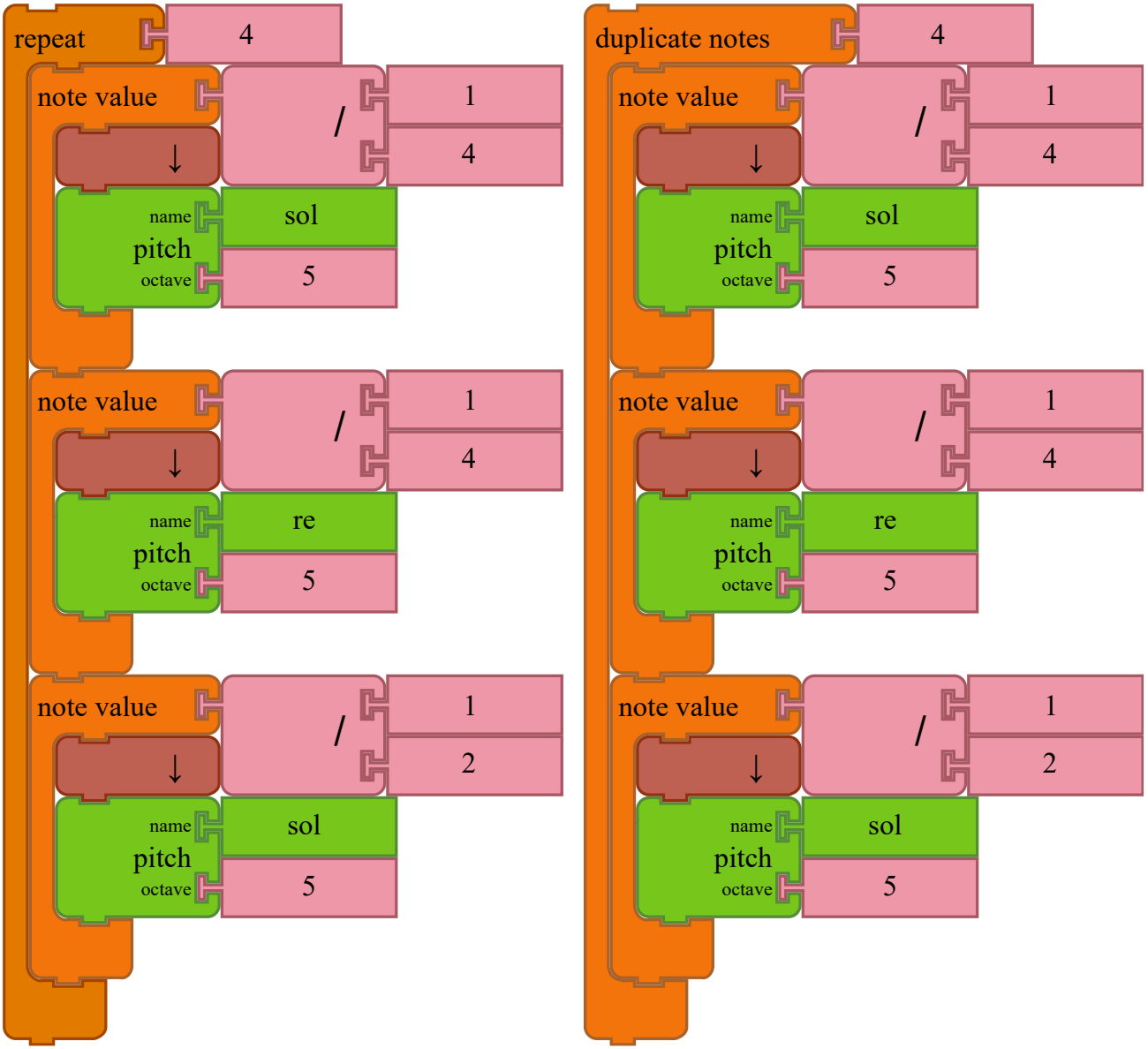
Western Notation	Music Blocks Notation with dot	Music Block Notation without dot
 For x = 1/8, $\frac{1}{8} + \frac{1}{(8*2)} = \frac{1}{8} + \frac{1}{16} = \frac{2}{16} + \frac{1}{16} = \boxed{\frac{3}{16}}$		
 For x = 1/4, $\frac{1}{4} + \frac{1}{(4*2)} = \frac{1}{4} + \frac{1}{8} = \frac{2}{8} + \frac{1}{8} = \boxed{\frac{3}{8}}$		

3.2.5 使用数学加快放慢音符



你也可以乘除音符播放的时间，这就会加快或放慢音符。 把一个 `1/8` 音符的播放时间乘 `2` 等于播放一个 `1/16` 音符。 把一个 `1/8` 音符的播放时间除 `2` 等于播放一个 `1/4` 音符。

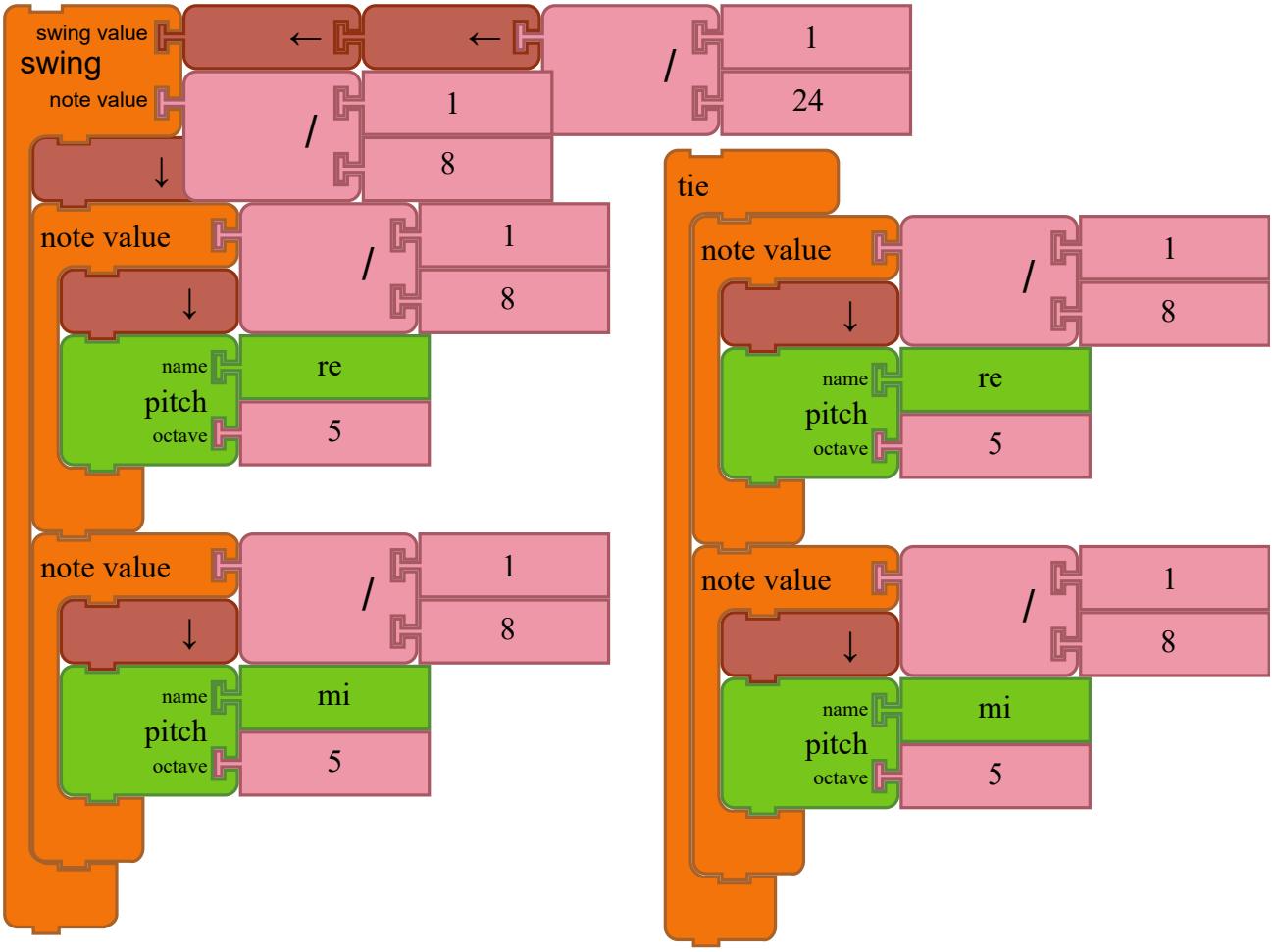
3.2.6 重复音符



《音乐拼块》有几个重复拼块的方法。 *Repeat* 拼块会重复播放一整个系列的音符， 而 *Duplicate* 拼块会重复那个系列中的每一个音符。

在左边的例子，后果就会变成 `Sol, Re, Sol, Sol, Re, Sol, Sol, Re, Sol, Sol, Re, Sol` 。 在右边的例子，后果就会变成 `Sol, Sol, Sol, Sol, Re, Re, Re, Re, Sol, Sol, Sol, Sol` .

3.2.7 摆动音符和合并音符



Swing 拼块改变两个音符 (定位音值), 加长第一个音符播放的时段 (定位 “swing” 号码) 和减少第二个音符播放的时间。 不搭配的音符不会被影响。

在例子里， `re5` 就会变成一个 `1/6` 音符，而 `mi5` 就会变成一个 `1/12` 音符 (`1/8 + 1/24 === 1/6` and `1/8 - 1/24 === 1 / 12`)。 注意这两个音符播放的总共时间还是一样的。

“Tie” 拼块改变两个音符,把音符组合起来. (这两个音符音调可以一样，但是节奏可以不同。)

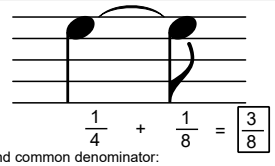
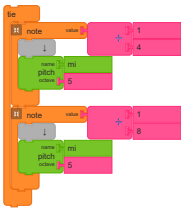
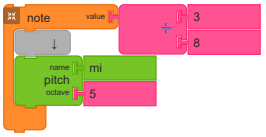
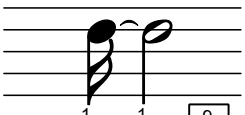
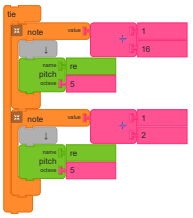
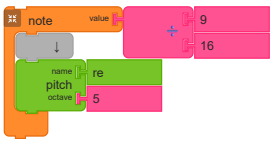
Using Notes with Ties

A tie connects two notes of the same pitch* and indicates that they are to be played as the sum of the two notes.

x= value of note 1 y= value of note 2

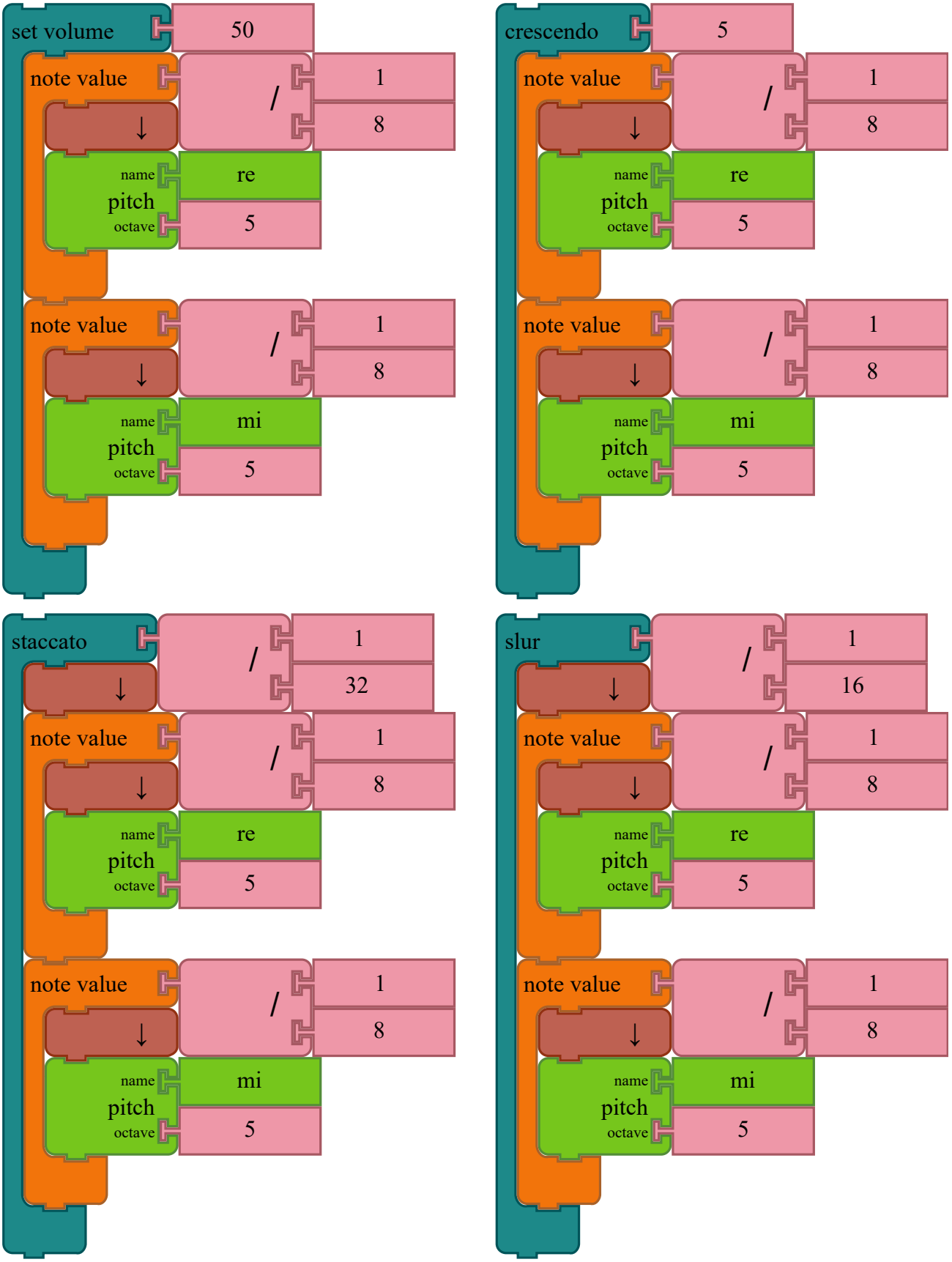
Formula: x + y = total value of notes contained within tie

Examples:

Western Notation	Music Blocks Notation with tie	Music Block Notation without tie
 $\frac{1}{4} + \frac{1}{8} = \boxed{\frac{3}{8}}$ Find common denominator: $x = \frac{1}{4}$, $y = \frac{1}{8}$ $2 * \frac{1}{4} = \frac{2}{8} + \frac{1}{8} = \frac{3}{8}$		
 $\frac{1}{16} + \frac{1}{2} = \boxed{\frac{9}{16}}$ Find common denominator: $x = \frac{1}{16}$, $y = \frac{1}{2}$ $8 * \frac{1}{2} = \frac{8}{16} + \frac{1}{16} = \frac{9}{16}$		

* Ties affect rhythm, not pitch. For tie to work, both pitches must be exactly the same. If not, it will be considered a slur.

3.2.8 定住，转变，断奏，模糊声音



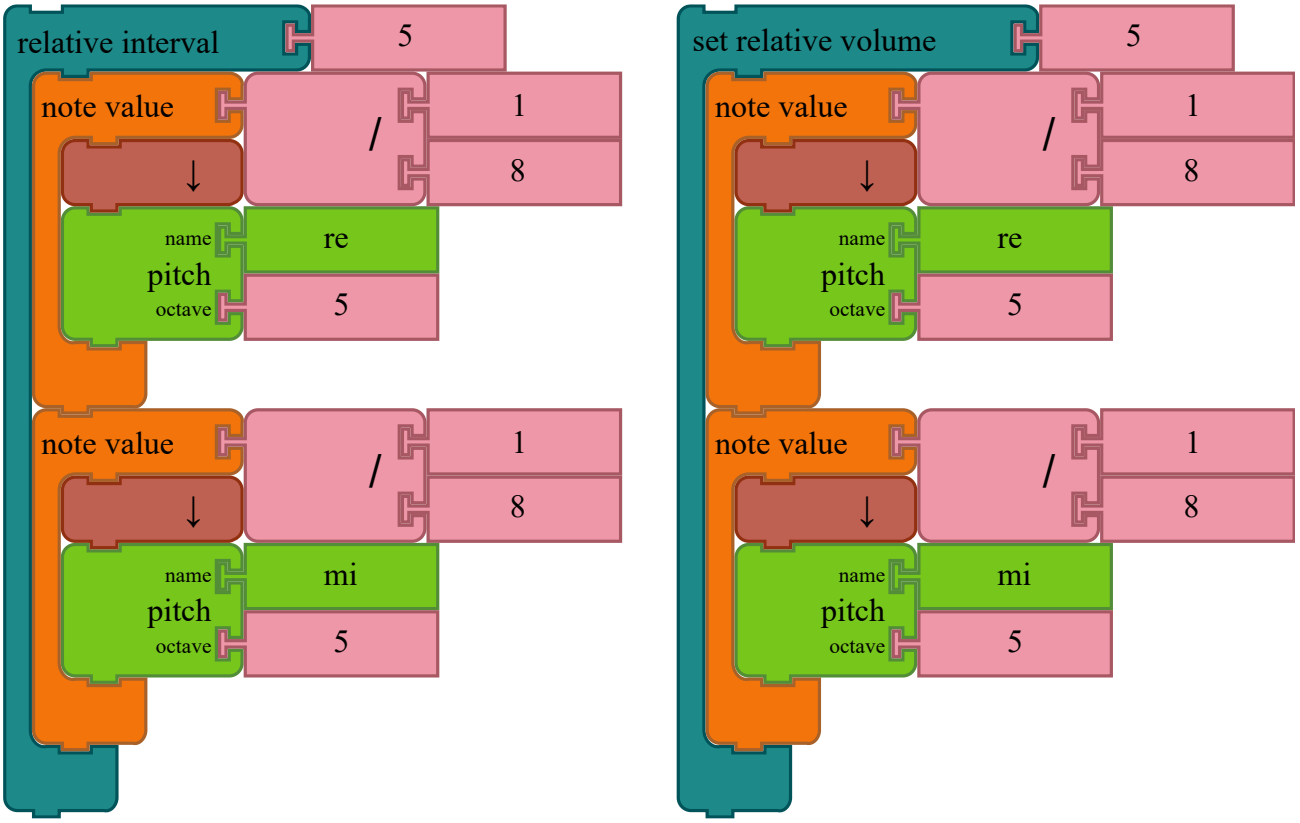
Set volume 拼块可以改变音符的声量. 声量开始定位 50; 范围是 0 (无声) 至 100 (最大声量).

Crescendo 拼块可以增加 (或减少) 里面每一个音符的声量. 例如， 一个有 3 个音符的系列在一个定位 “5” 价值的 *Crescendo* 拼块，最后的音符的声量就会比开始的声量多 15% 。

Staccato 拼块会减少音符的播放时间;在保持音符的节奏价值，使它们更尖锐.

Slur 拼块增加音符的播放时间，让它播放比定位更长的时间，保持音符的节奏价值，把它和下一个音符稍微混合在一起。

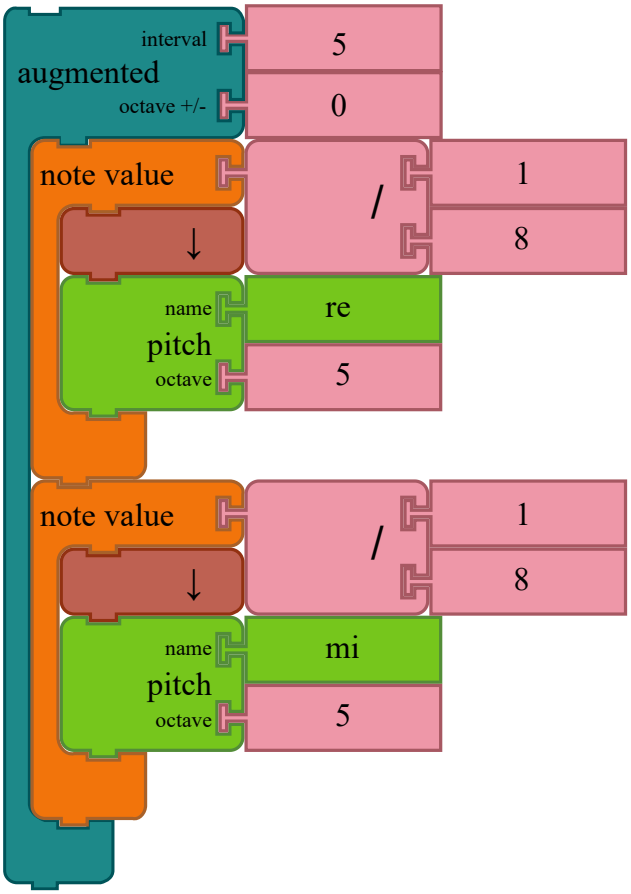
3.2.9 间隔和定住相对声音

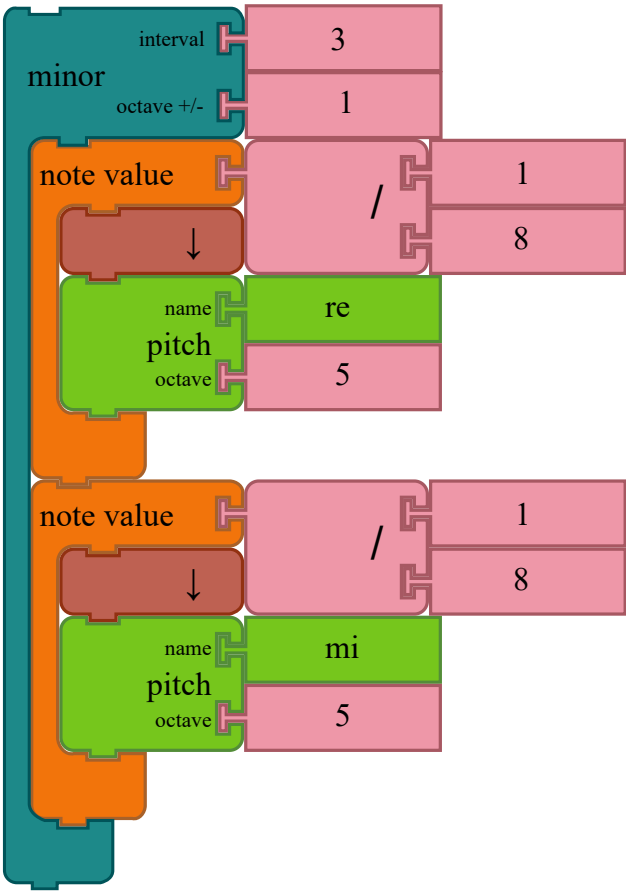


Interval 拼块计算一个相对的间隔，例如一个 "fifth"， 然后在音符播放时加上间隔的音调。在图示里，我们在 `La` 上加 `Re` 和在 `Ti` 上加 `Mi` 。

Set Relative Volume 拼块根据输入的号码，改变包住的音符的声量，根据原本的声量进行加减。 例如， 输入 1 0 0 等于把原本的声量乘二。

绝对间隔

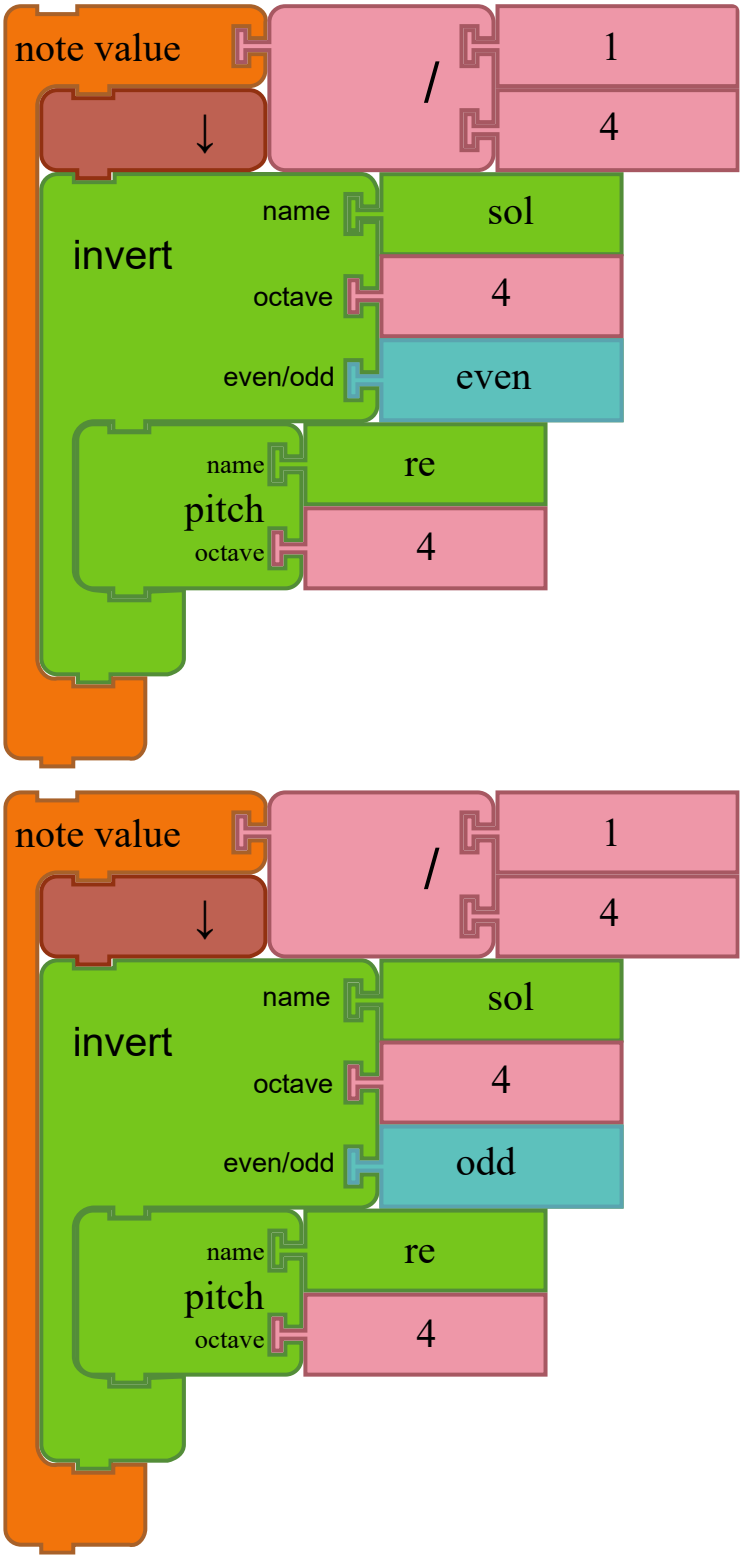




Augmented 拼块计算一个绝对的间隔， 如一个 “augmented fifth”， 然后在音符播放时加上间隔的音调。 同样的， Minor *拼块计算一个绝对的间隔, 如一个 "minor third". 其他绝对的间隔包括* Perfect, Diminished *和* Major*.

在上面 “augmented fifth” 的例子, 一个 D5 和 A5 的和音被播放, 接着播放一个 E5 和 C5 的和音。 在上面 “minor third” 的例子 （这包括一个八度的提升）, 首先一个 D5 和 F5 的和音被播放, 然后一个 E5 和 G6 的和音被播放。

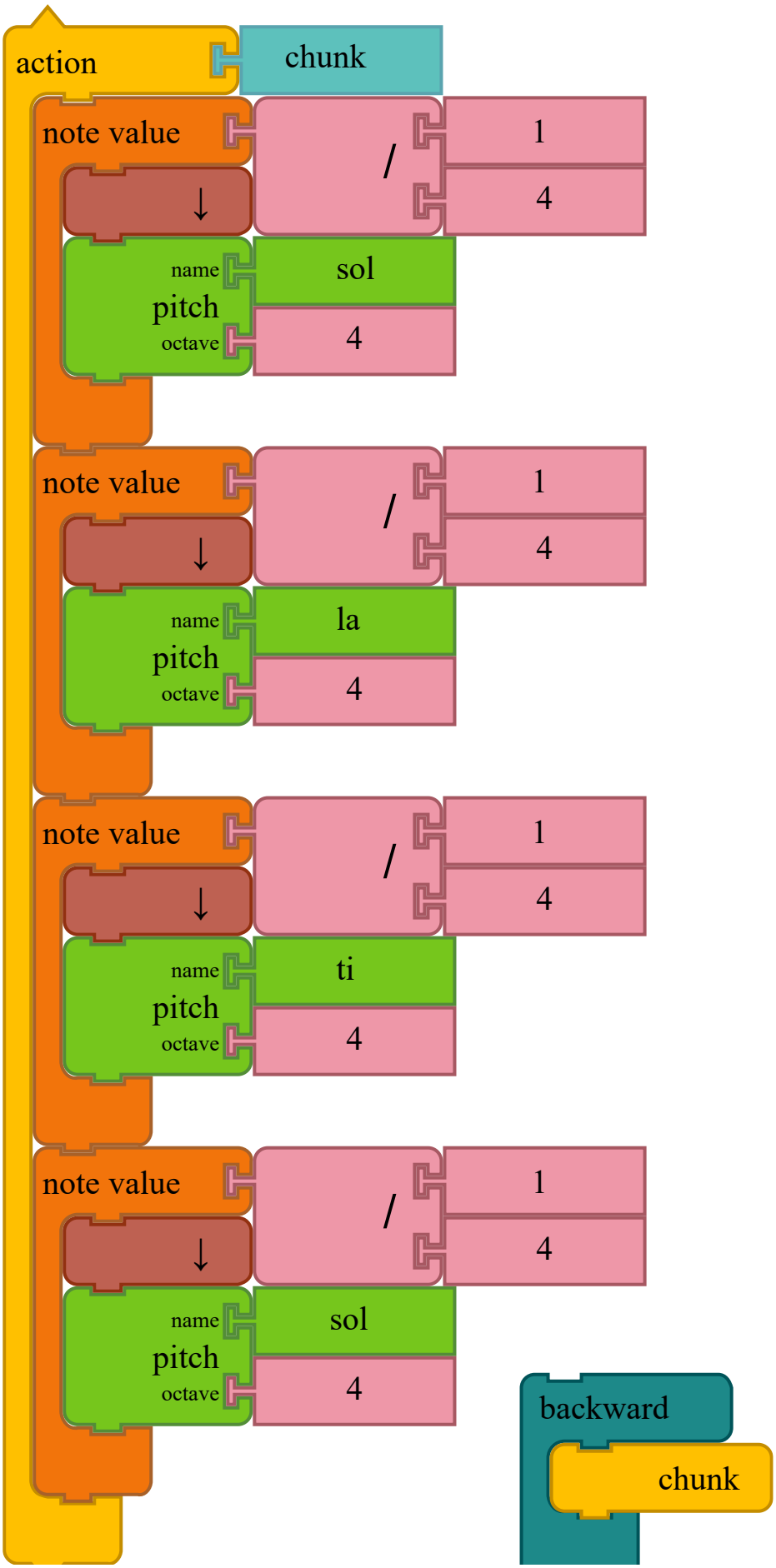
3.2.11 倒位



Invert 拼块会绕着一个被定住的音符，倒立另一系列的音符。 *Invert* 拼块包含两个方式 - *odd* 和 *even*. *Even* 把回转点的定位加 `1/4` 半步，允许两个音符之间有一个可以回转的点。

在 *invert (even)* 的例子，`D4` 绕着 `G4` 倒位，结果产生一个 `C5`。在 *invert (odd)* 的例子，`D4` 绕着 `G4` 和 `G♯4` 中间的一个点，结果产生一个 `C♯5`。

3.2.12 反向播放音乐

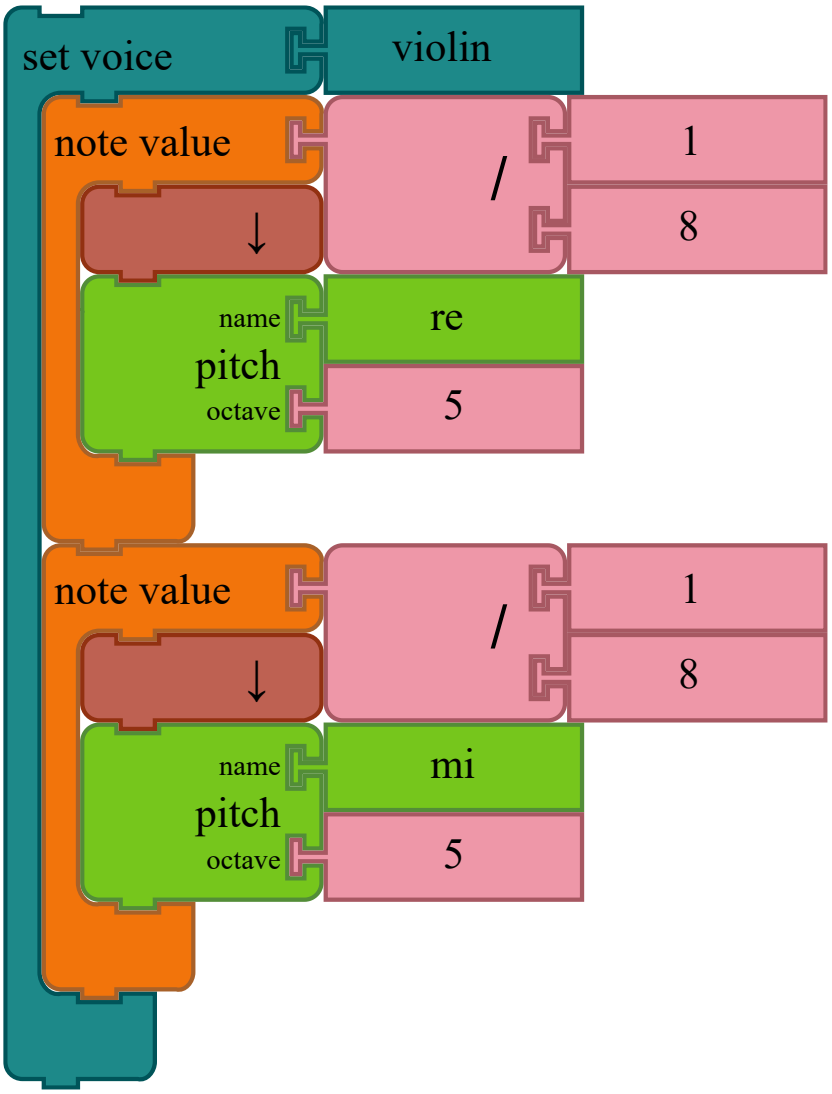

action
chunk
note value
/
1
4
name
pitch
octave
sol
4
note value
/
1
4
name
pitch
octave
la
4
note value
/
1
4
name
pitch
octave
ti
4
note value
/
1
4
name
pitch
octave
sol
4
backward
chunk

Backward 拼块会倒叙的播放里面包住的音符。 在上面的例子，*Chunk* 里面的音符播放的顺序是 `Sol`，`Ti`，`La`，`Sol`，明确的来说从程序堆的下面到上面。

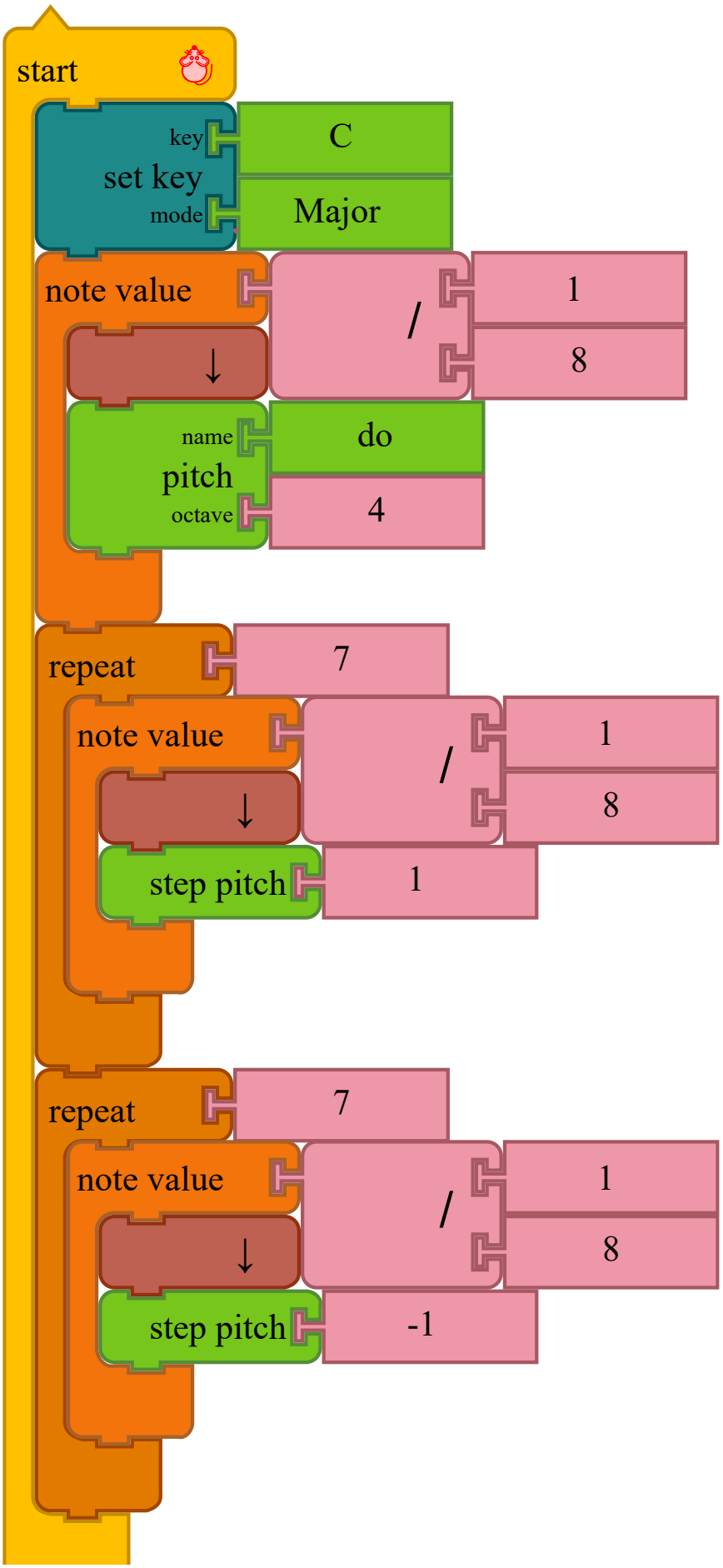
RUN LIVE (https://musicblocks.sugarlabs.org/index.html?id=1522885752309944&run=True)

注意所有在 *Backward* 拼块里面的音符是倒叙的， 所以如果你的程序有逻辑和音符混合在一起，记得注意这个拼块。

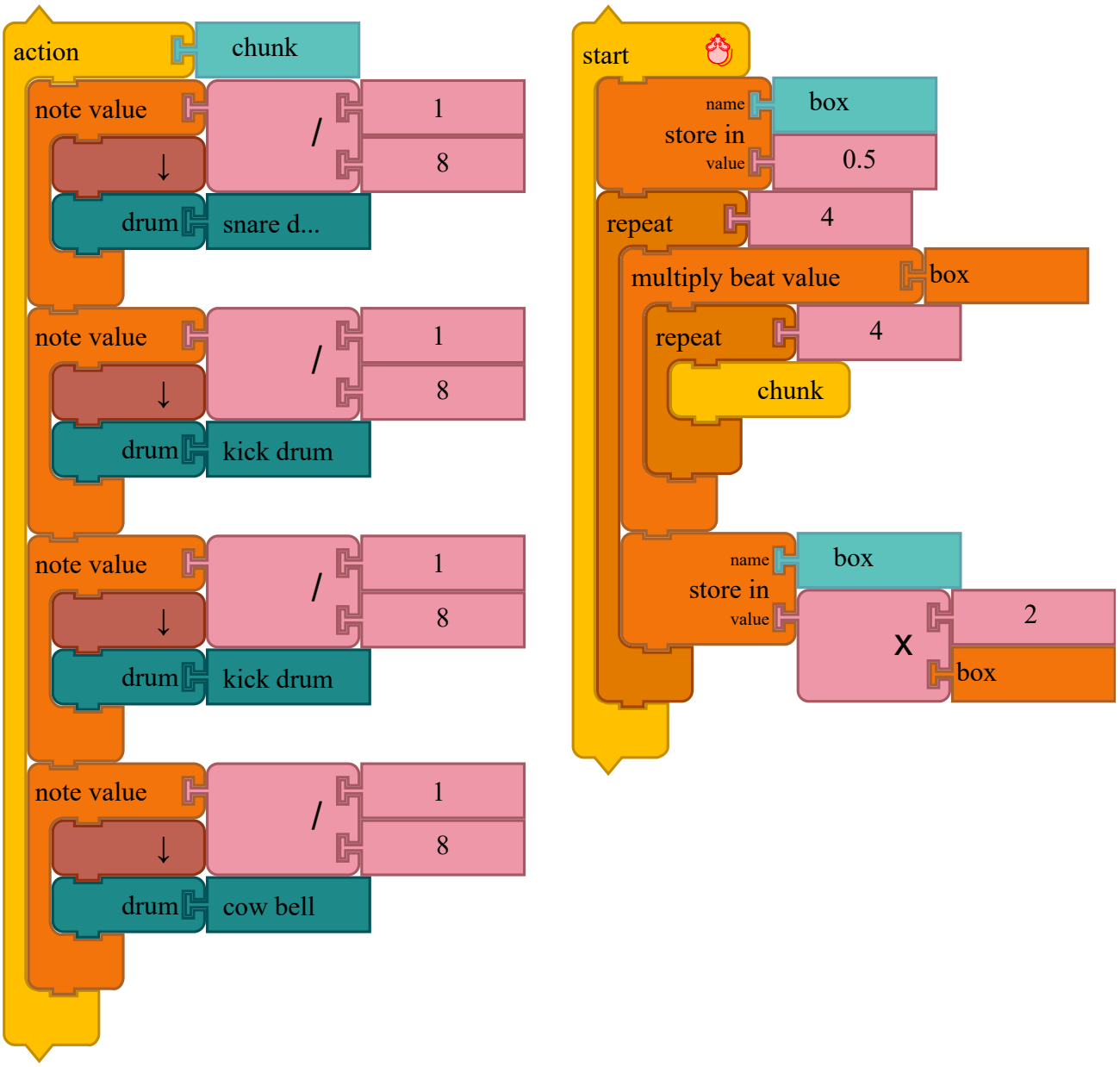
3.2.13 定住音色和音调



Set Voice 拼块为里面的拼块的发声器选择一个 音色，如小提琴或大提琴。


start
key
C
set key
mode
Major
note value
/
1
8
name
do
pitch
octave
4
repeat
7
note value
/
1
8
step pitch
1
repeat
7
note value
/
1
8
step pitch
-1

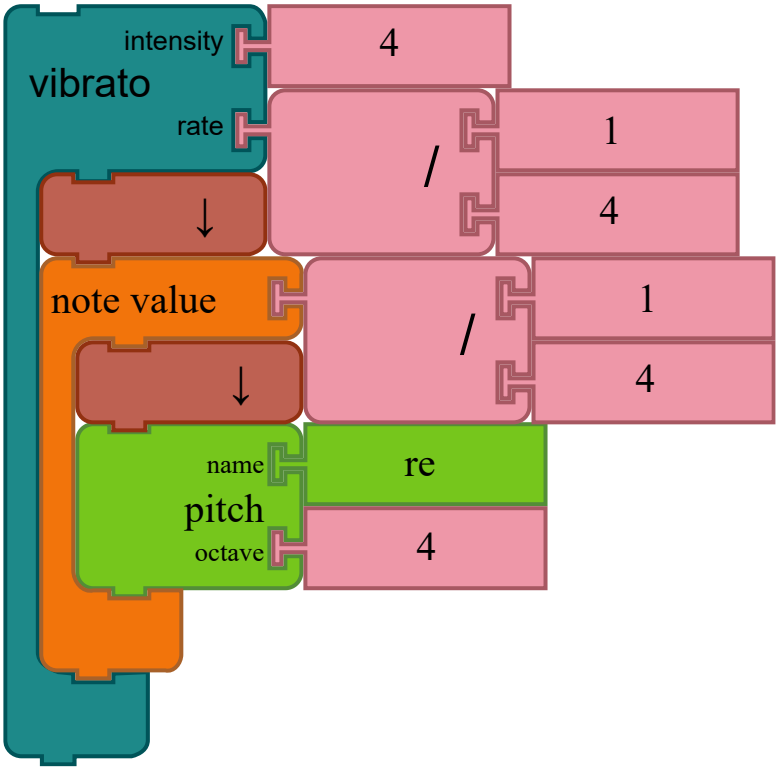
Set Key 拼块可以改换音乐音符的模式, 如把 `Do`， `Re`， `Mi`，变成音符名字如在 C Major 里的， `C`，`D`，`E`. 模式包括 Major,Minor,Chromatic 和一群比较罕见的模式，如 Bebop, Geez, Maqam, 等. 这个拼块能根据使用者选择的音乐模式，让使用者得到 《音乐拼块》 的 "movable Do"。



在上面的例子, 鼓声 拍的次数跟着时间增加。

RUN LIVE (https://musicblocks.sugarlabs.org/index.html?id=1523106271018484&run=True)

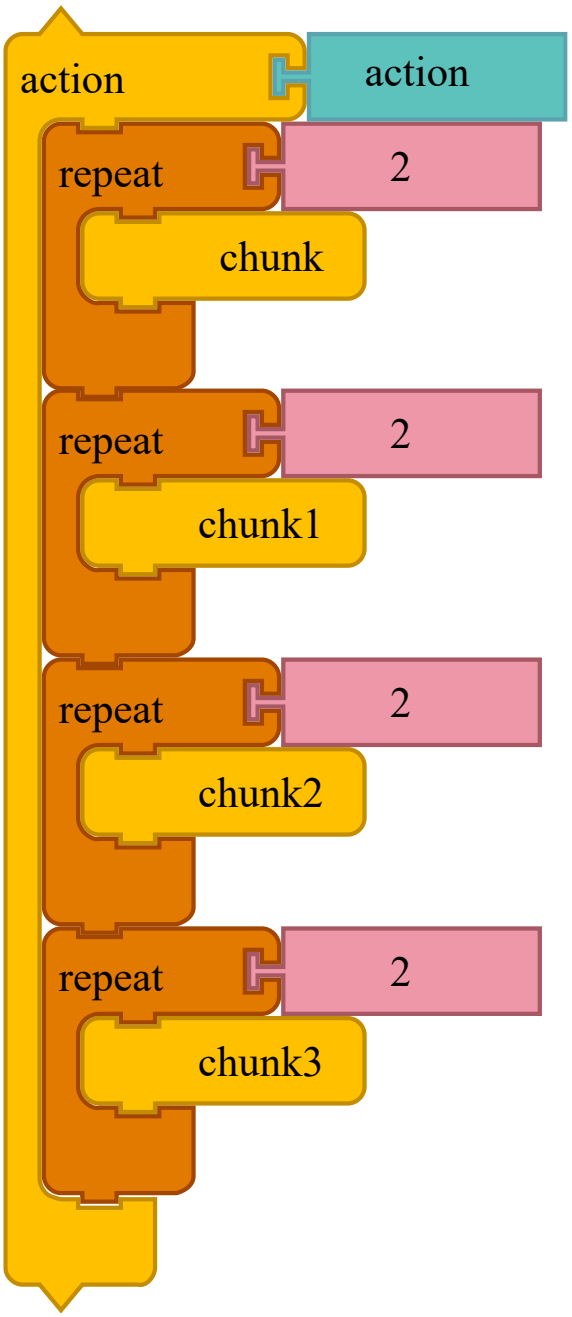
3.2.14 颤音



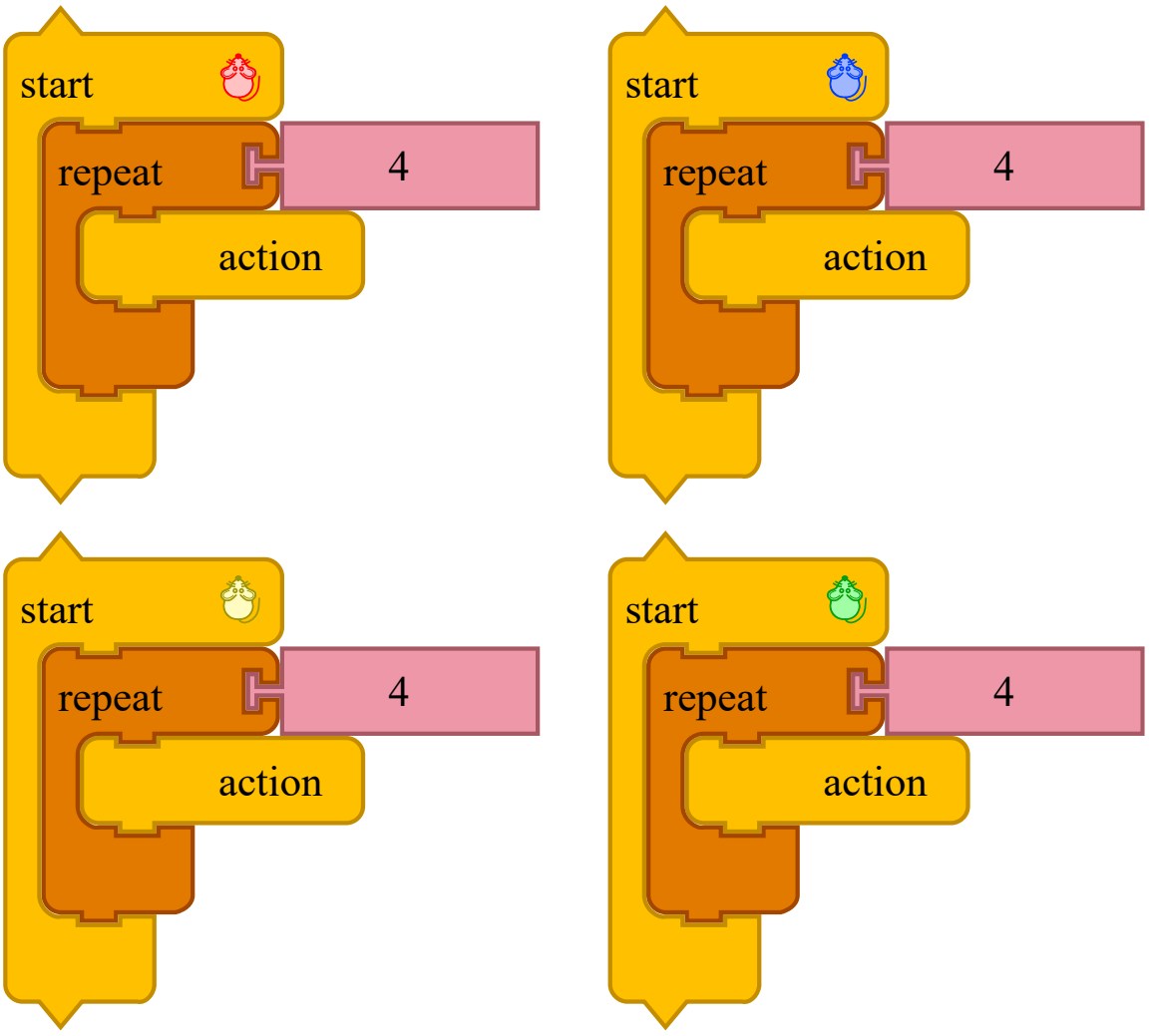
Vibrato 拼块在里面包住的拼块上加上一个音调的快转变。 转变的强度的范围从 1 至 100（加或减一个半步）。 频率的定值会影响转变的速度。

3.3 音色

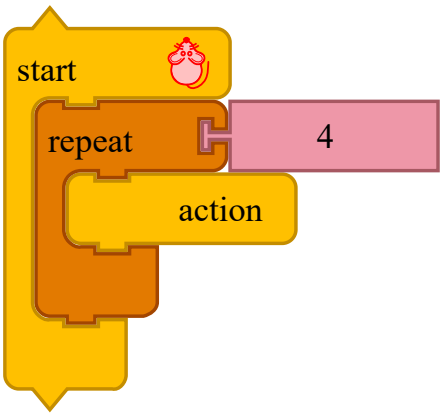
每一个 *Start* 拼块在《音乐拼块》执行一个不同的音色。 (每当你按下 "Run" 按钮，全部的 *Start* 拼块在一样时间执行.)

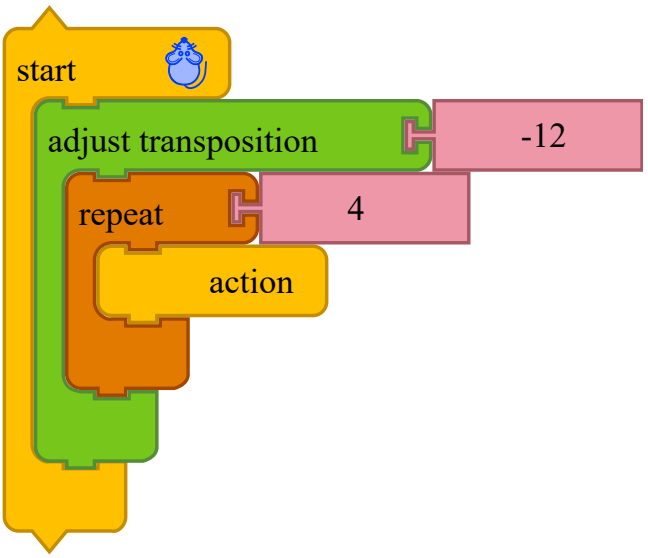


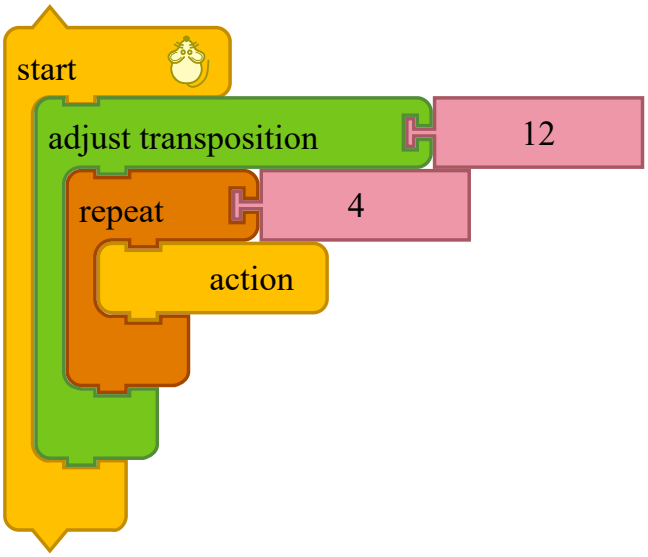
如果我们执行我们的程序和歌曲。。。

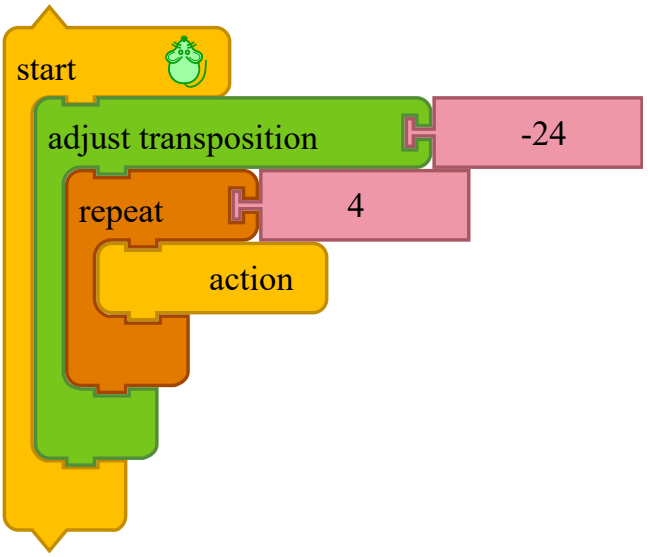


...我们可以从不同的 *Start* 拼块开始.





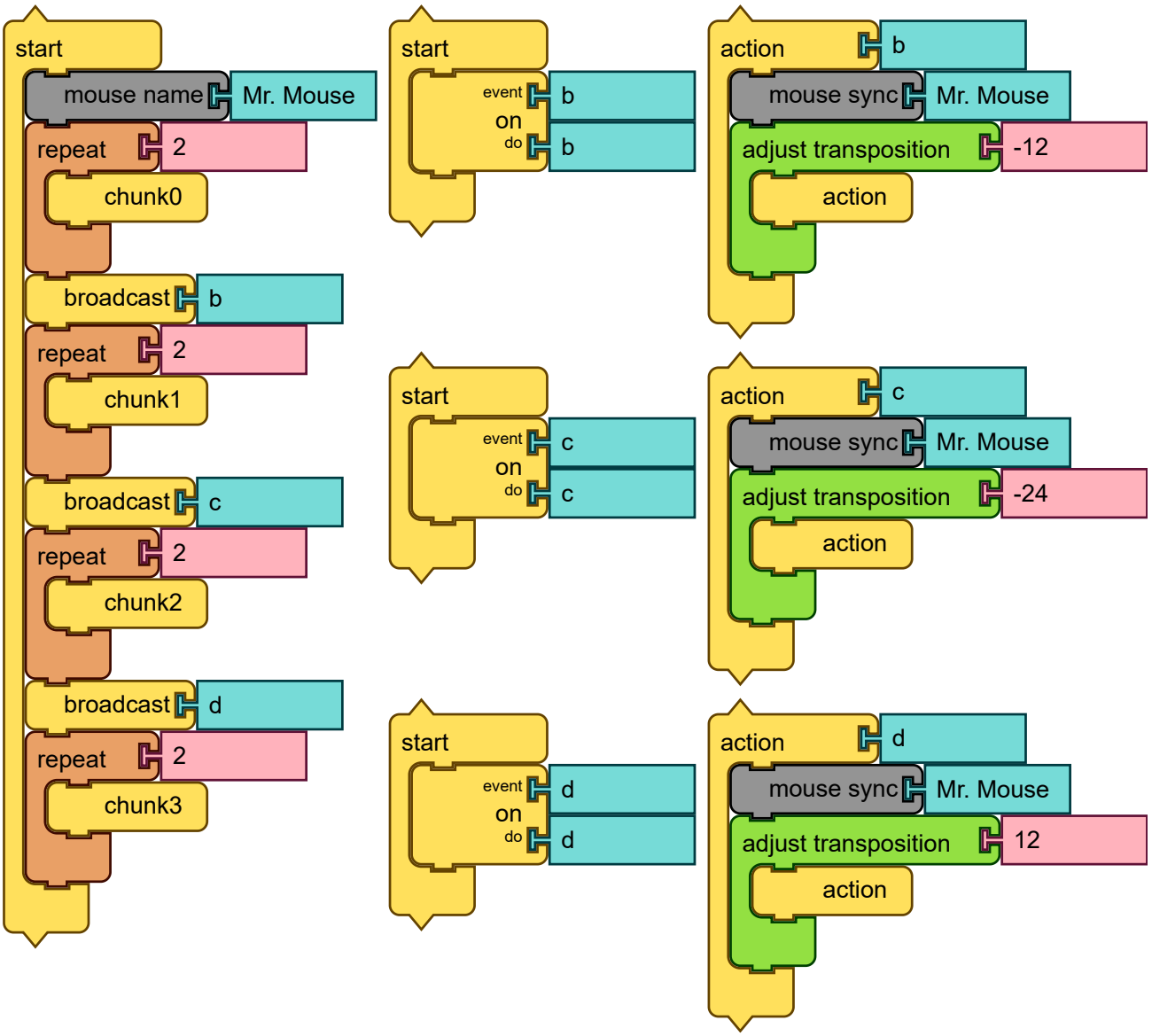




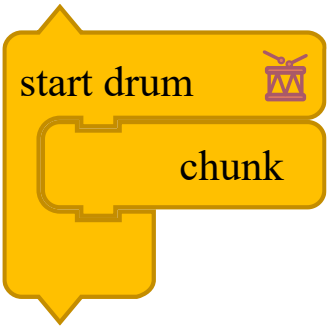
如果我们增加或降低音乐的八度，这会使我们的歌曲更有趣。

alt tag

如果我们再把不同的音色在不同的时间播放，我们也可以使歌曲变有趣。 RUN LIVE (https://musicblocks.sugarlabs.org/index.html?id=1523026536194324&run=True)

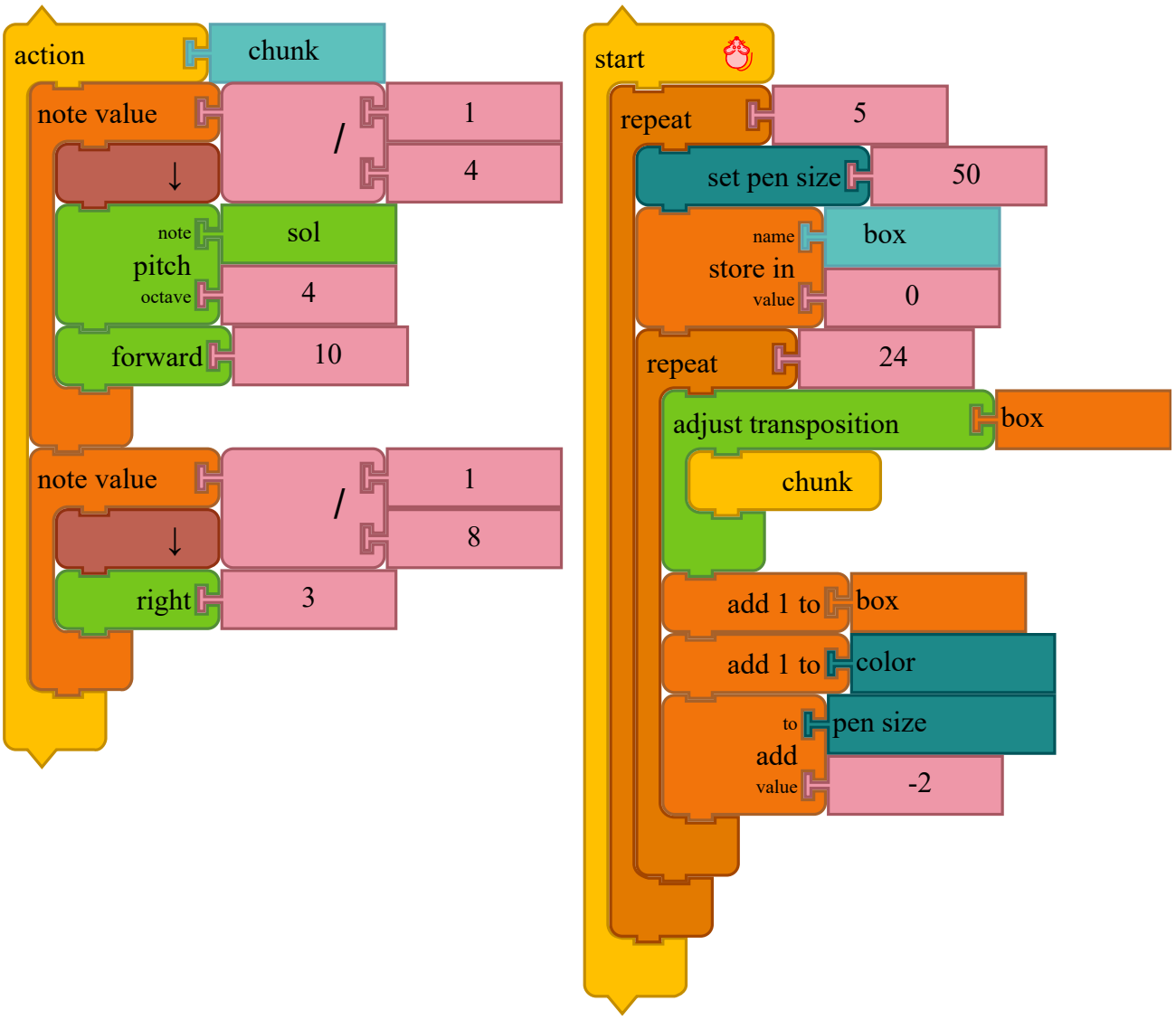


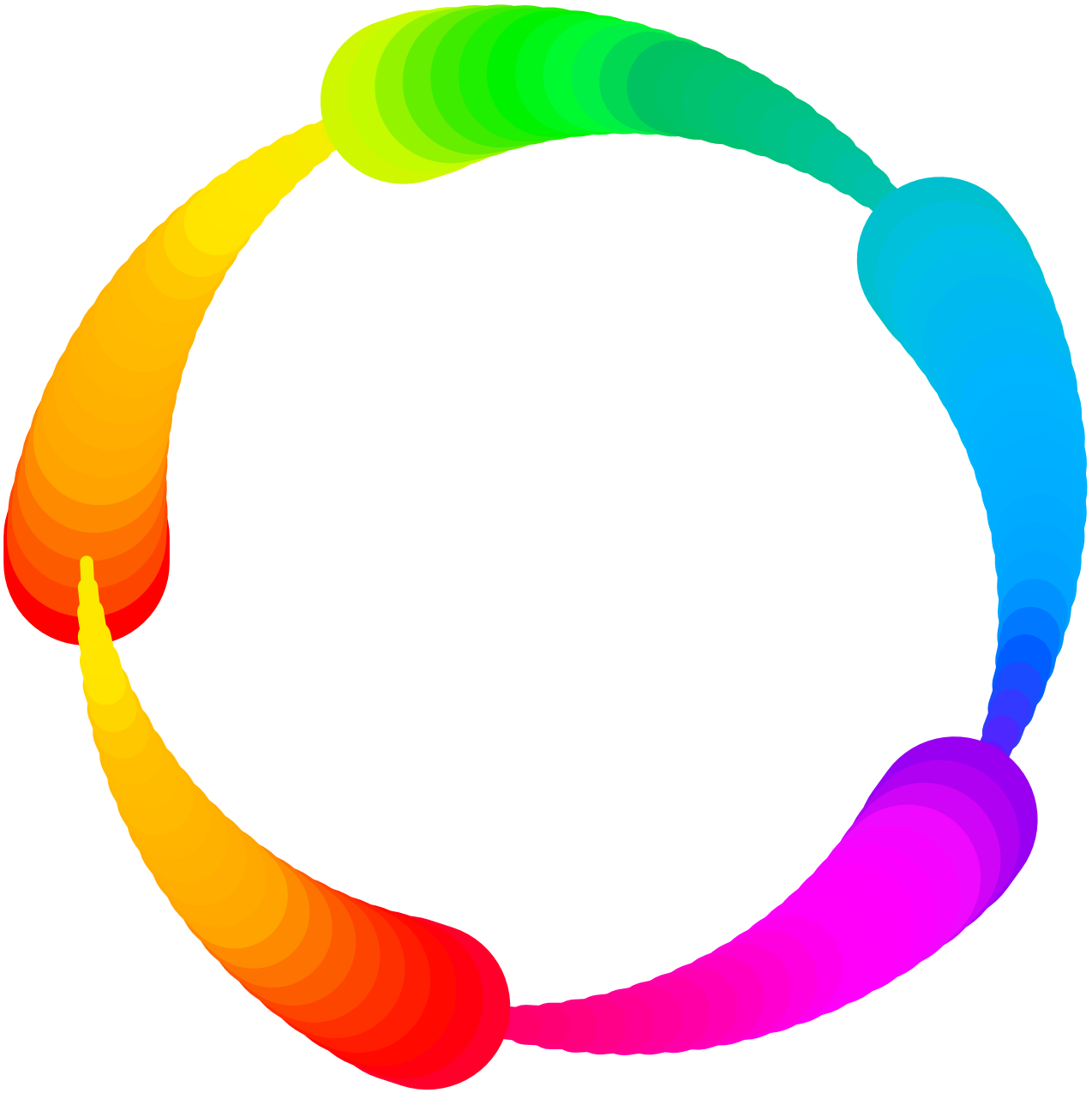
另外一个使用以计算好的延迟的方式是使用 *Broadcast* 拼块，带出不同的音色。 在上面的例子每当歌曲的一部分播放完之后，一个新的事件会被执行，带出一个新的音色。 注意我们在例子里使用 *Mouse Sync* 拼块，这能确保不同的音色和主时钟（master clock）同步。



《音乐拼块》拥有一个特别 "鼓声" 模式的 *Start* 拼块，这个拼块适合设计鼓声歌曲。 在使用这个拼块时，里面的 *Pitch* 拼块就会使用原本的鼓声，播放成 `C2` 。 在上面的例子里，全部 `chunk` 里面的拼块会播放为踢鼓（kick drum）。

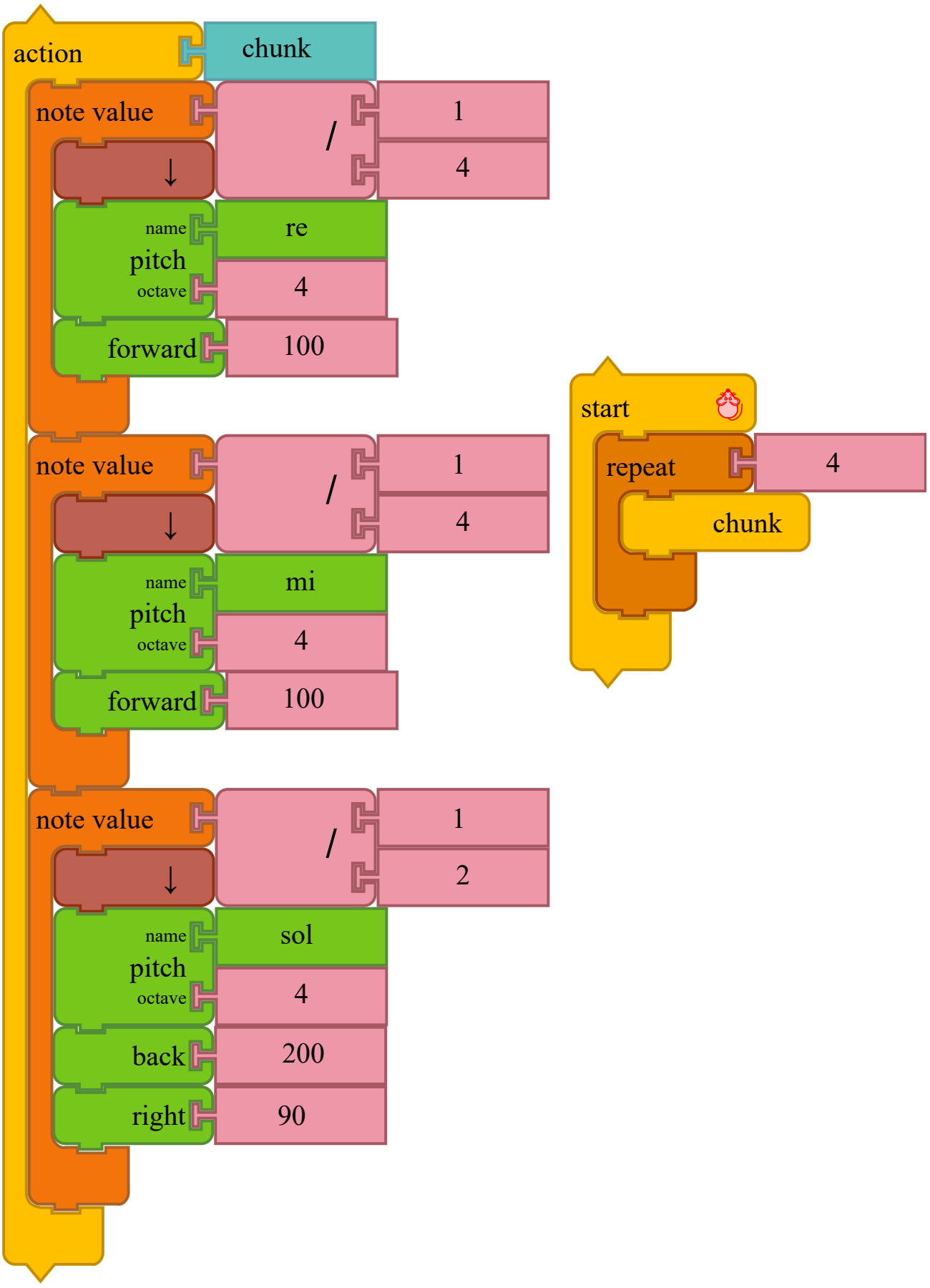
3.4 图像


action
chunk
note value
/
1
4
↓
note
sol
pitch
octave
4
forward
10
note value
/
1
8
↓
right
3
start
repeat
5
set pen size
50
name
box
store in
value
0
repeat
24
adjust transposition
box
chunk
add 1 to
box
add 1 to
color
to
pen size
add
value
-2



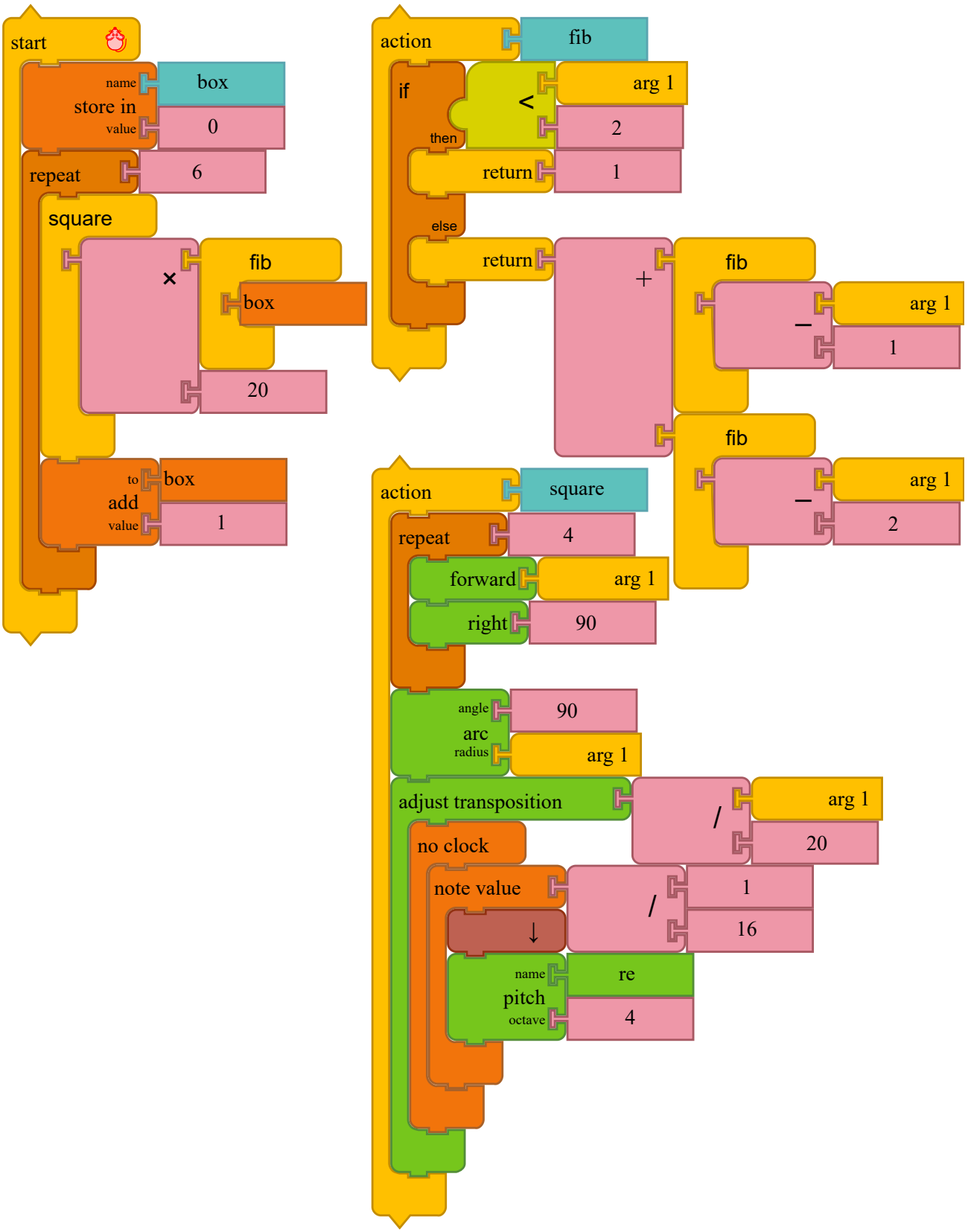
《乌龟》图像可以和音乐拼块融合在一起。 当我们使用图像拼块，如在 *Note Value* 拼块里面，使用 *Forward* 和 *Right* 拼块,图像可以和音乐同步。 在例子里，每当一个四分之一音符被播放后，乌龟就会往前方向走一步。 在第八个音符，乌龟就会转向右边，音符上升半步，笔的尺度被减少，笔的颜色也在里面 “重复” 的程序每一步增加。

RUN LIVE (https://musicblocks.sugarlabs.org/index.html?id=1518563680307291&run=True)

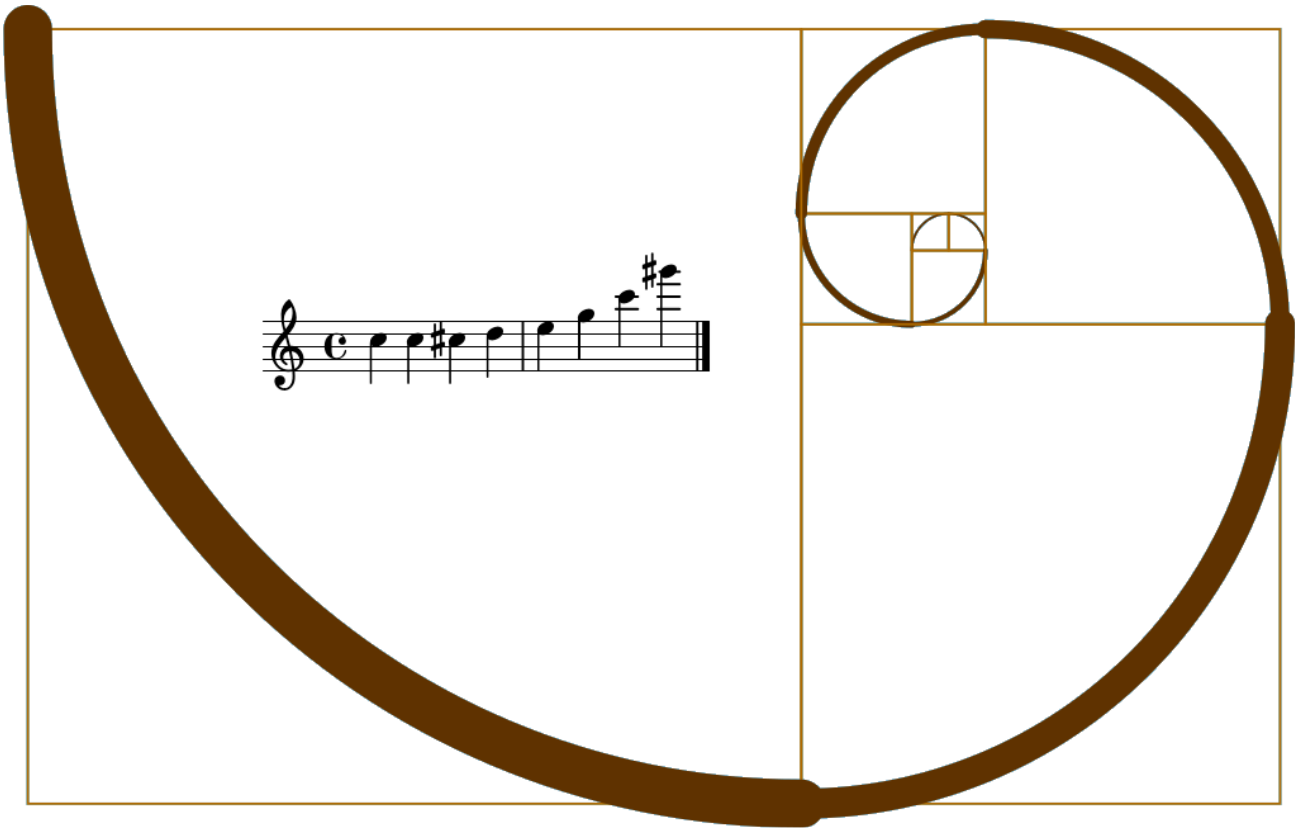


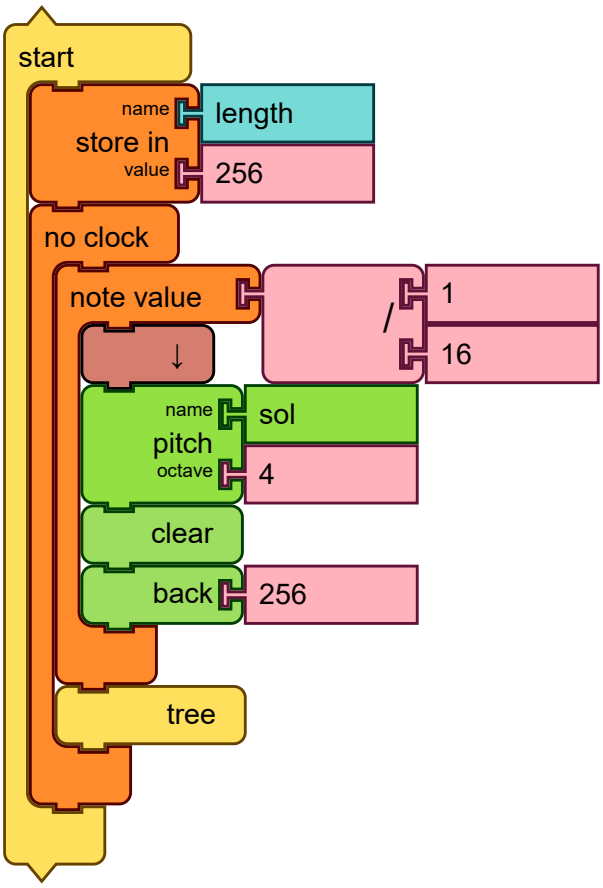
另外一个图像和音乐合步的程序例子就是把图像程序放进 *Note Value* 拼块。

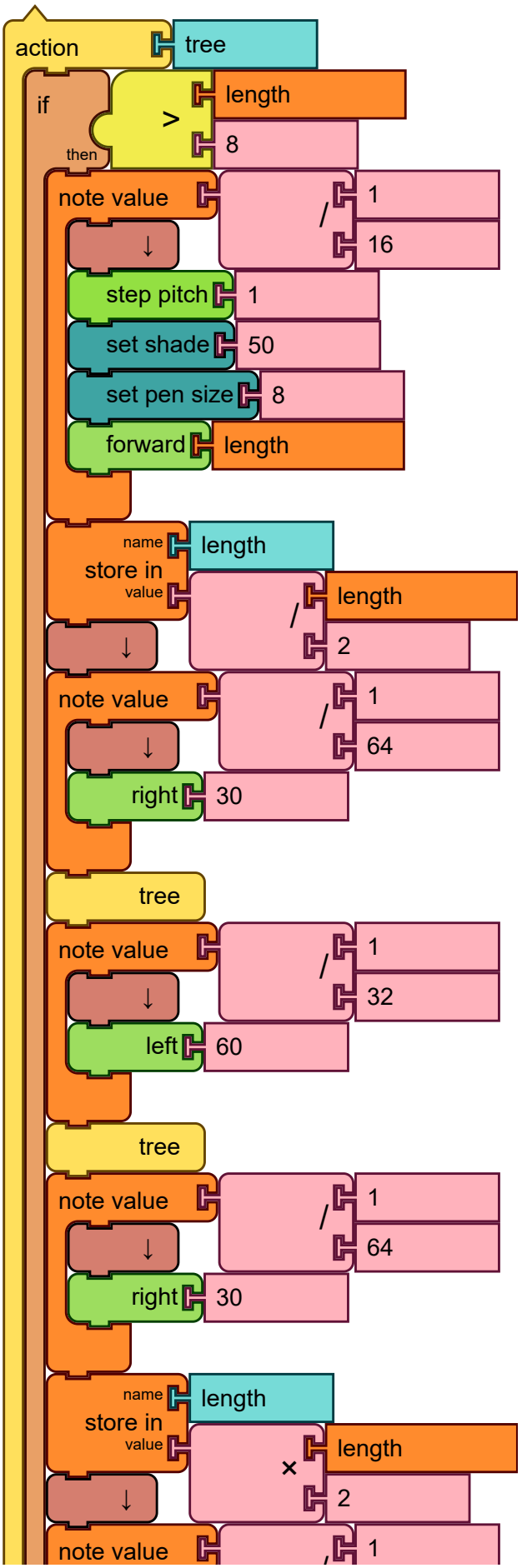
RUN LIVE (https://musicblocks.sugarlabs.org/index.html?id=1523106271018484&run=True)

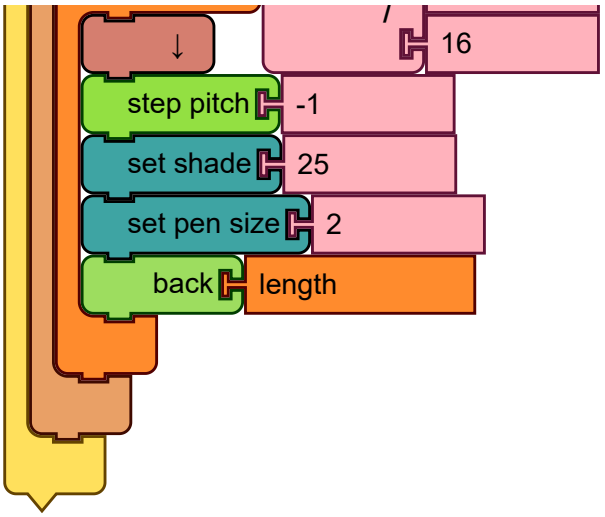


在上面的程序例子，因为图像和音乐比较复杂，我们使用一个 *No-clock* 拼块，使图像和音乐分开执行。 "No-clock* 拼块会优先执行图像，然后执行音乐的节奏。

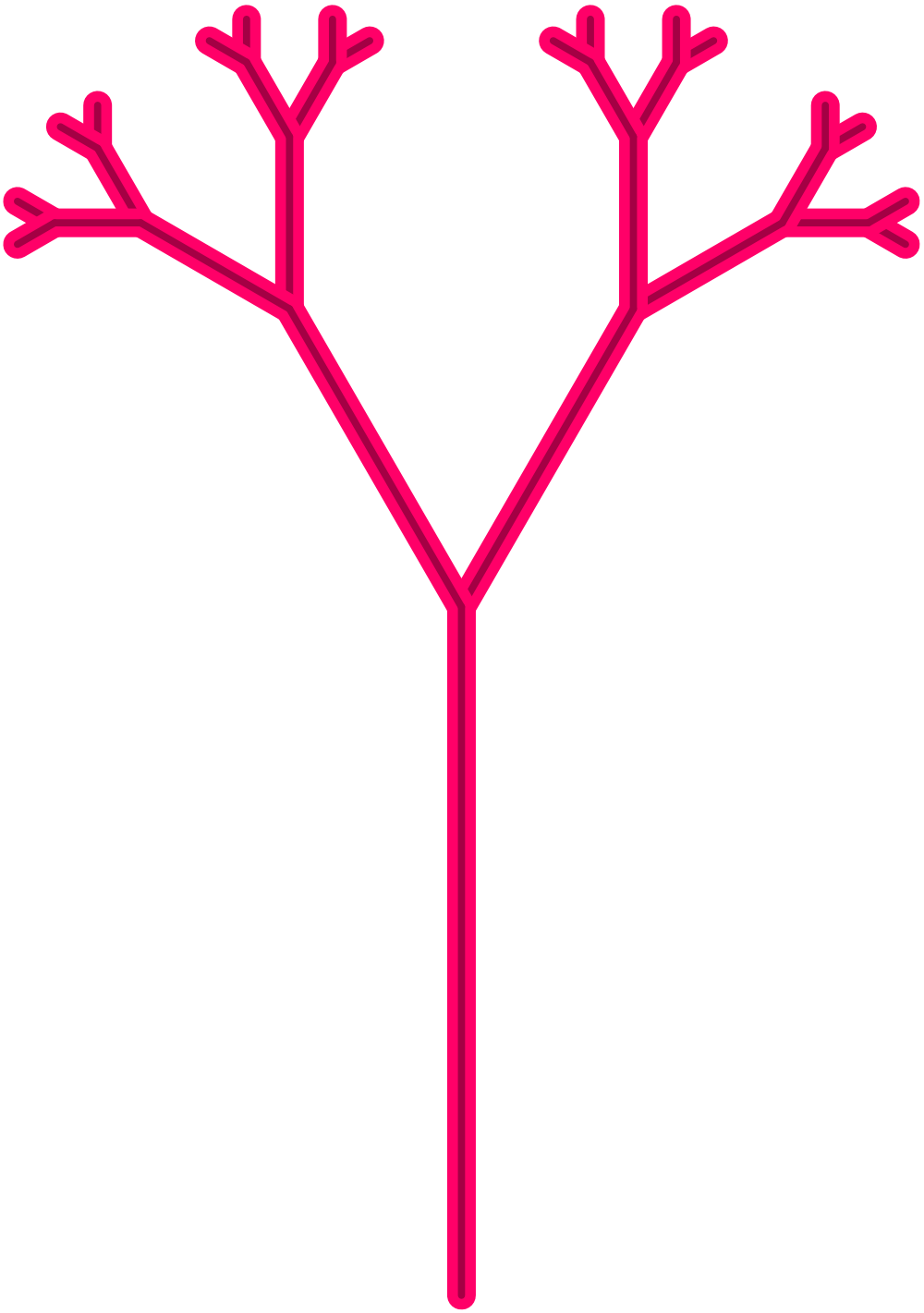



start
name
store in
value
length
256
no clock
note value
/
1
16
name
pitch
octave
sol
4
clear
back
256
tree


action
tree
if
then
>
length
8
note value
/
1
16
step pitch
1
set shade
50
set pen size
8
forward
length
name
store in
value
length
/
length
2
note value
/
1
64
right
30
tree
note value
/
1
32
left
60
tree
note value
/
1
64
right
30
name
store in
value
length
×
length
2
note value
/
1



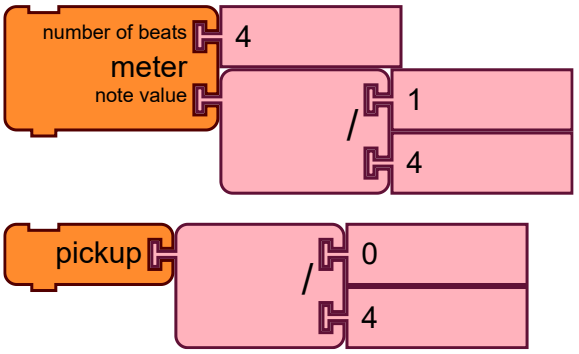
另外一个把图像和音乐混合在一起的例子是一个递归树；在树枝往上移的时候，音调也往上调高。



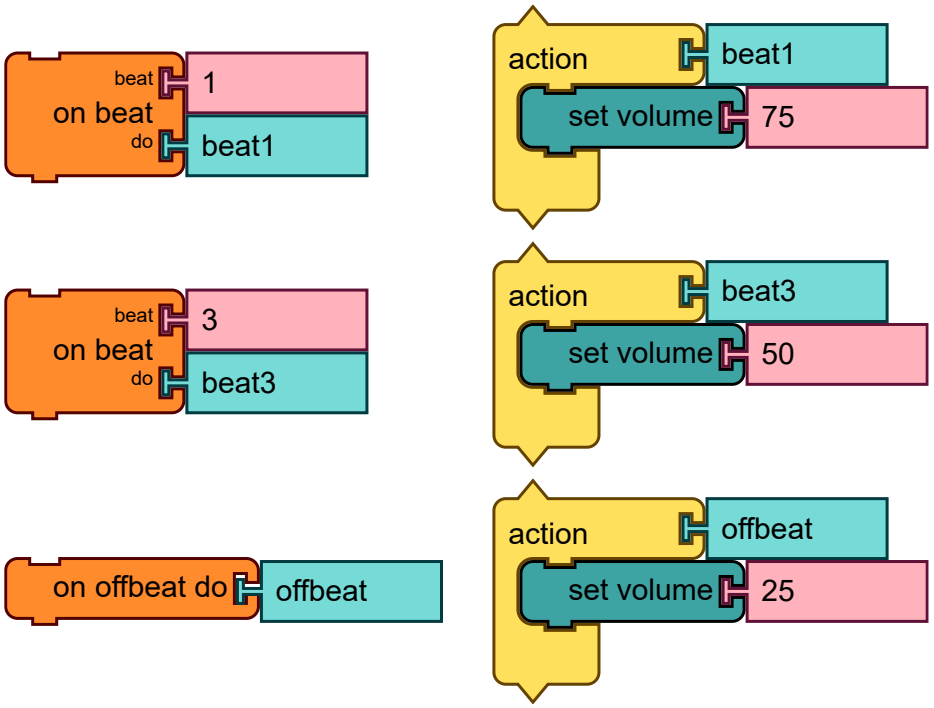
RUN LIVE (https://musicblocks.sugarlabs.org/index.html?id=1523029986215035&run=True)

3.5 节奏

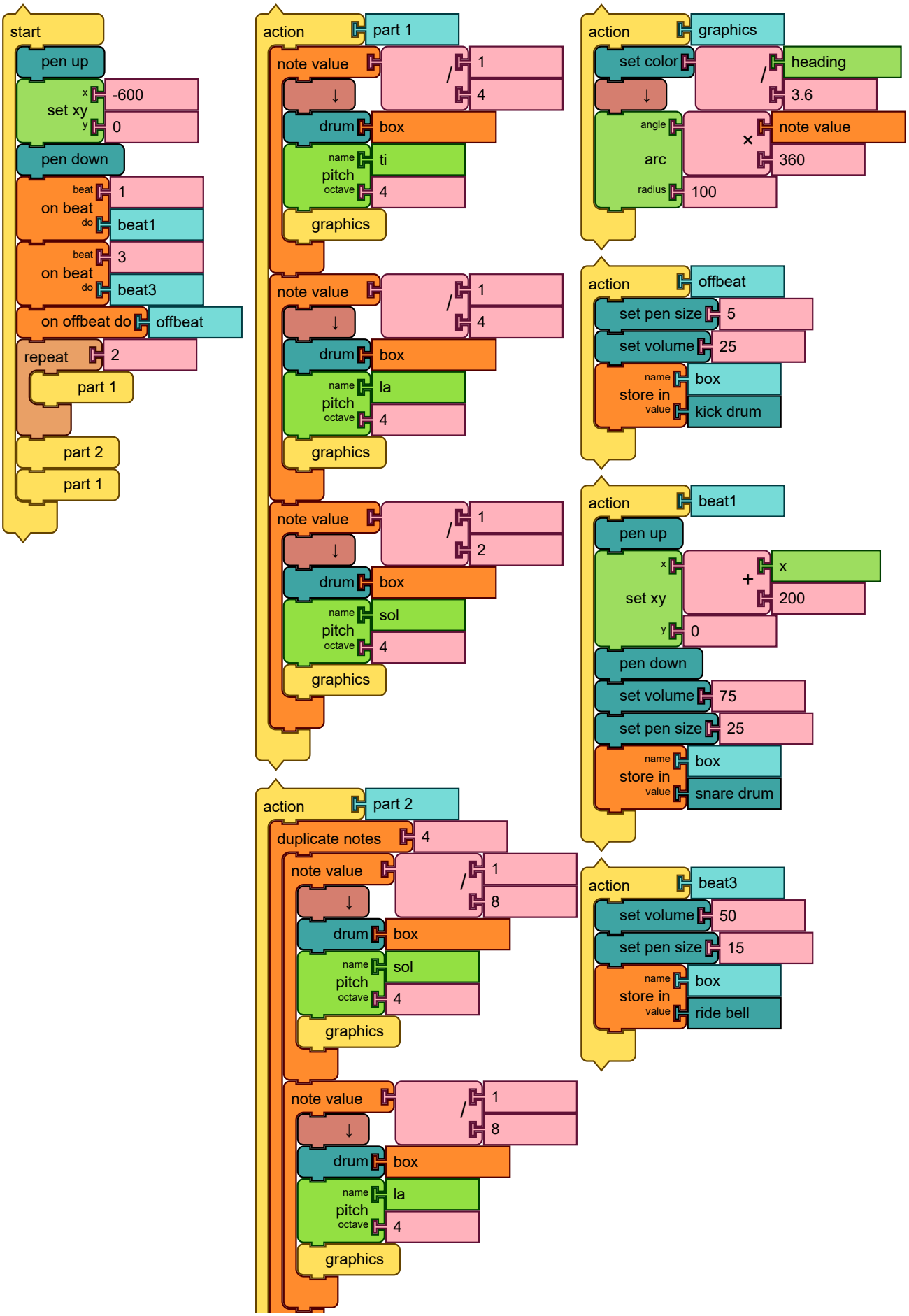
音乐的节奏是被 *Meter* 拼块定住 (音乐的节奏开始定为 4:4)。 我们可以使用 *Pickup* 拼块来播放在节奏开始之前的任何音符。



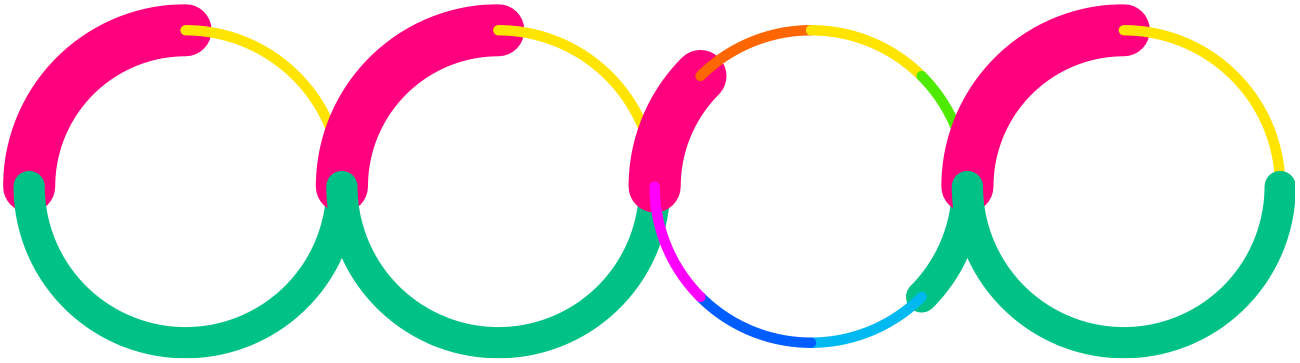
定住节奏可以帮助我们调一个音符的素质。 在下面的例子，在第 1 拍和第 3 拍的音符的声量调高，而在其他的拍时声量调低。



On-Beat-Do 和 *Off-Beat-Do* 拼块让我们指定在某一个拍时做出一个行动。 (注意行动会在任何一个在节奏拼块里的音符先执行。)

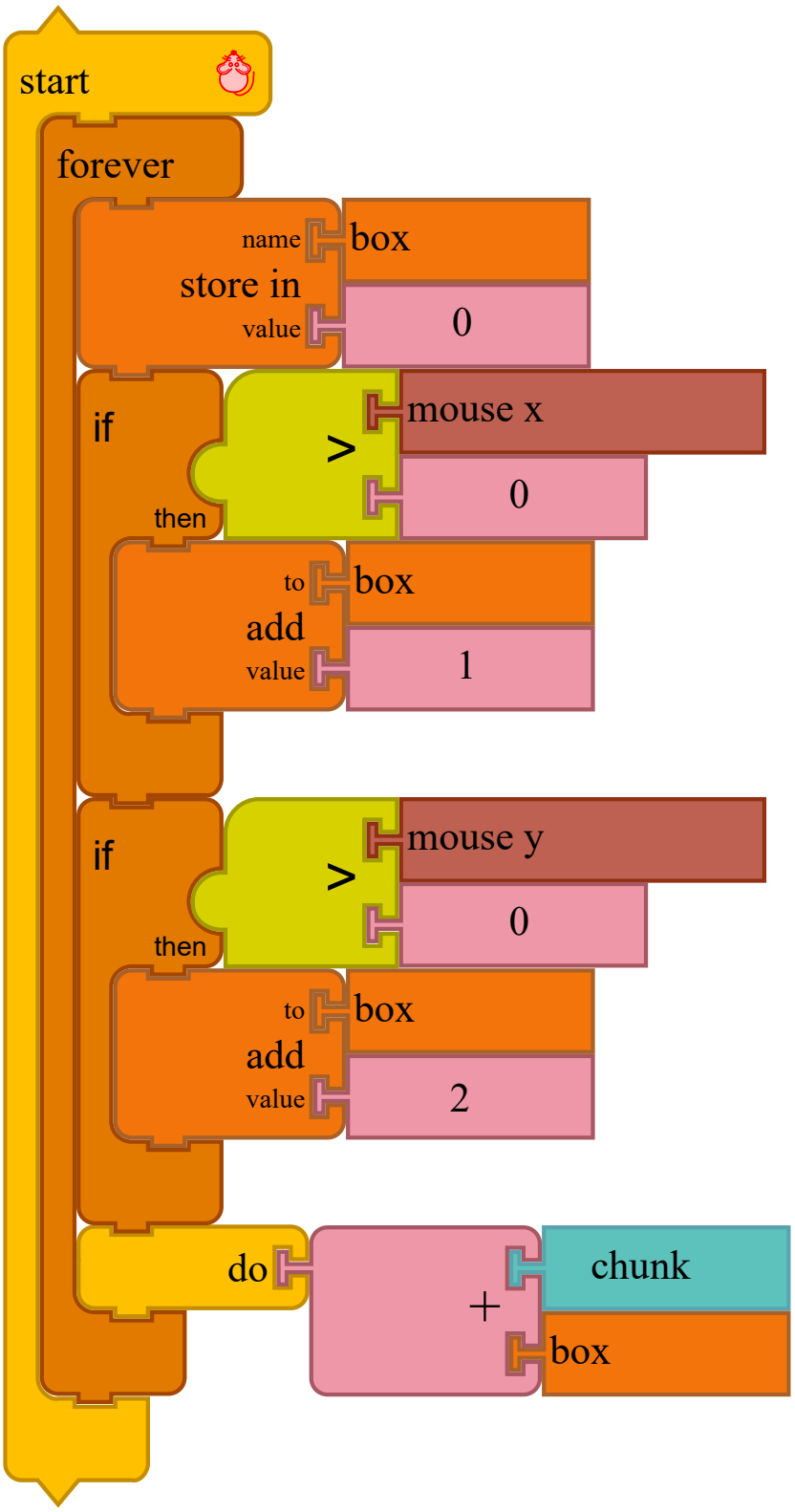

start
pen up
set xy
x
-600
y
0
pen down
on beat
beat
1
do
beat1
on beat
beat
3
do
beat3
on offbeat do
offbeat
repeat
2
part 1
part 2
part 1
action
part 1
note value
1
4
drum
box
name
pitch
octave
ti
4
graphics
note value
1
4
drum
box
name
pitch
octave
la
4
graphics
note value
1
2
drum
box
name
pitch
octave
sol
4
graphics
action
part 2
duplicate notes
4
note value
1
8
drum
box
name
pitch
octave
sol
4
graphics
note value
1
8
drum
box
name
pitch
octave
la
4
graphics
action
graphics
set color
heading
3.6
angle
arc
radius
100
note value
360
action
offbeat
set pen size
5
set volume
25
name
store in
value
box
kick drum
action
beat1
pen up
set xy
x
x
200
y
0
pen down
set volume
75
set pen size
25
name
store in
value
box
snare drum
action
beat3
set volume
50
set pen size
15
name
store in
value
box
ride bell

另外一个显示图像的方法就是靠节奏执行它们。 在上面的例子，我们在每一个音符执行同样的行动，但是行动的参数，如笔的宽度，依赖着现在执行的音符。 在第 1 拍，我们把笔的宽度定为 50 和把声量定为 75。 在第 3 拍，我们把笔的宽度定为 25 和把声量定为 50。 在其他的拍子上，我们把笔的宽度定为 5 和把声量定为 5。 我们创造的图像在下面。



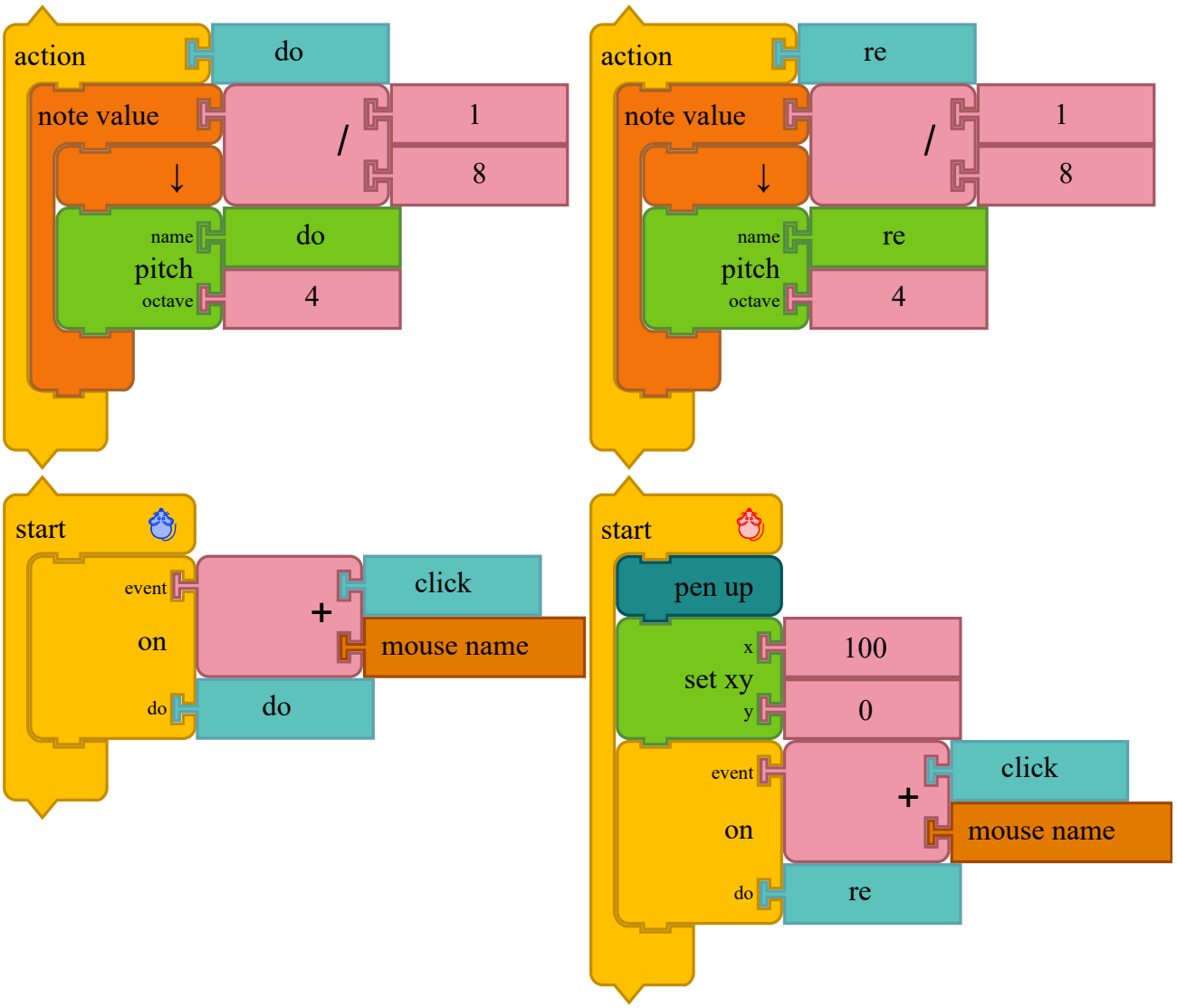
3.6 相互作用

《音乐拼块》拥有很多可以和自己引起相互作用的拼块，如记载鼠标的地位影响音乐的某一个方面。



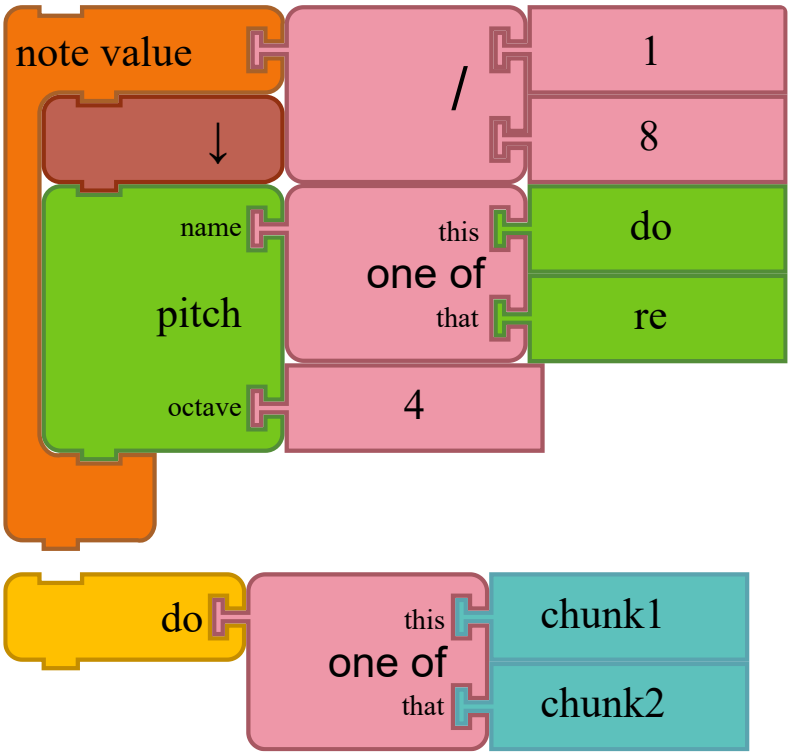
例如，我们可以交互的执行上面的拼块。 当鼠标在左下角时， chunk 的拼块被执行; 而鼠标在右下角时，chunk1 的拼块被执行； 鼠标在左上角时， chunk2 的拼块被执行; 和右上角时， chunk3 拼块被执行。

RUN LIVE (https://musicblocks.sugarlabs.org/index.html?id=1523028011868930&run=True)



在上面的例子，我们使用两个不同 “乌龟”，定下两个不同的 *click* 行动，一个简单的两个音符的钢琴被做出。 你可以做出一个八个音符的钢琴吗?

RUN LIVE (https://musicblocks.sugarlabs.org/index.html?id=1523107390715125&run=True)



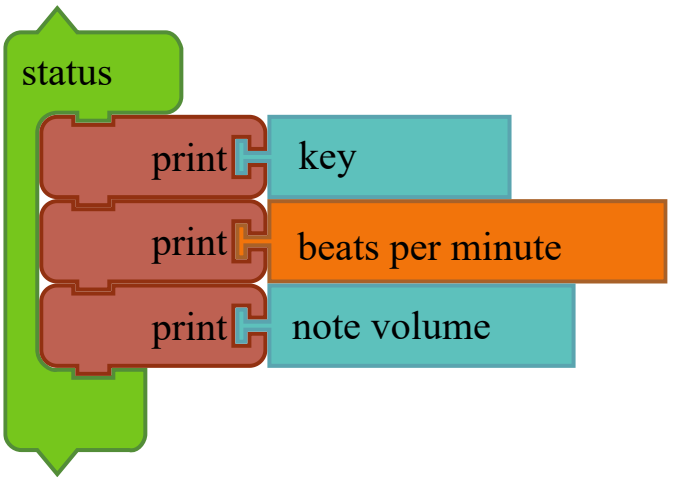
你也可以在你的程序上加上一点随机。 在上面的例子里，我们使用 *One-of* 拼块，每次执行 *Note Value* 拼块时，来随机定住 `Do` 或 `Re` 。 在下面的例子里，我们使用 *One-of* 拼块随机选择 `chunk1` 或 `chunk2` 。

部件

上一章 (3. 使用音乐设计程序) | 回去目录 | 下一章 (5. 《音乐拼块》之外)

说明书里的这一章将会解释《音乐拼块》的不同部件，你可以使用这些不同部件来提高你使用《音乐拼块》的经验。

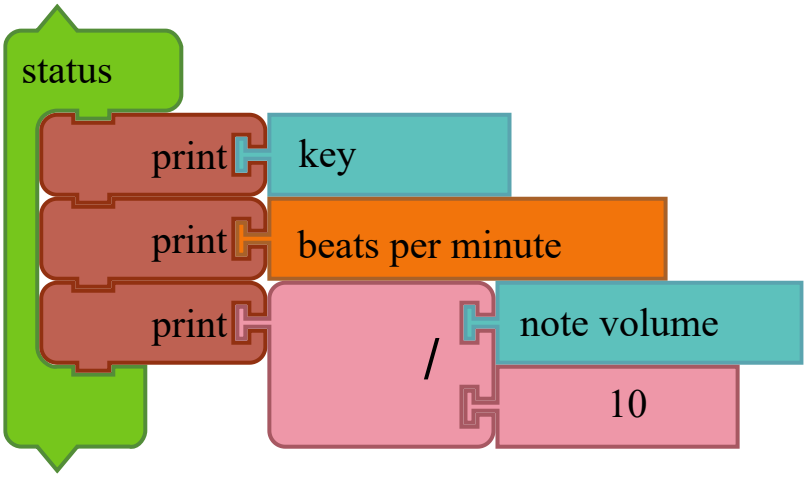
4.1 观察状况



	key	bpm	volume	note
	C Major	90	50	C4 8
	C Major	90	50	A5 4

Status Widget 是一个可以观察《音乐拼块》执行时状态的部件。 这个部件开始会显示音乐的节奏，声量和音调。 每一个音符也会在播放时被显示。 状态表格上的每一行代表着一个音色。

你也可以在 *Status* 部件上加上其他 *Print* 拼块，来显示其他的音乐因素,例如重复，移调，跳步, 断奏, 浑浊音符, 和 图像 因素, 如, x, y, 标题，颜色，色彩深浅和笔的宽度。

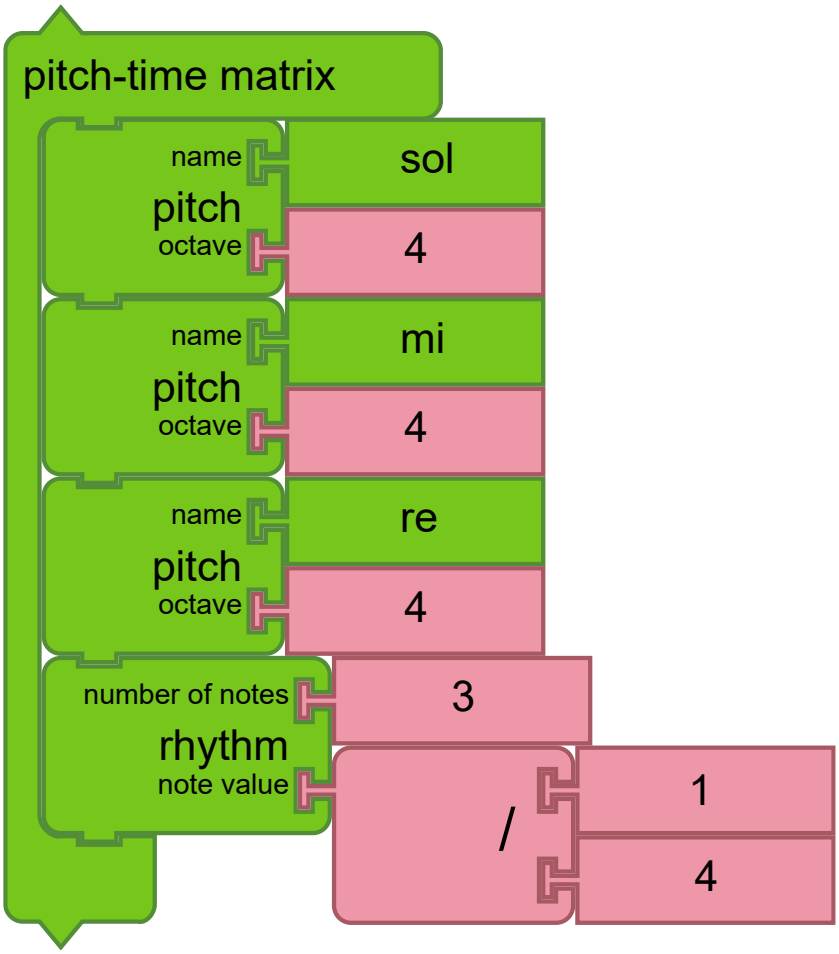


您可以在 *Status* 拼块中进行其他编程。 在上面的例子中，音量在被显示之前被除以10。

4.2 产生音乐砖块

使用 *Pitch-Time Matrix*, 可以以更快的速度产生大块的音符。

4.2.1 音调-时间矩阵



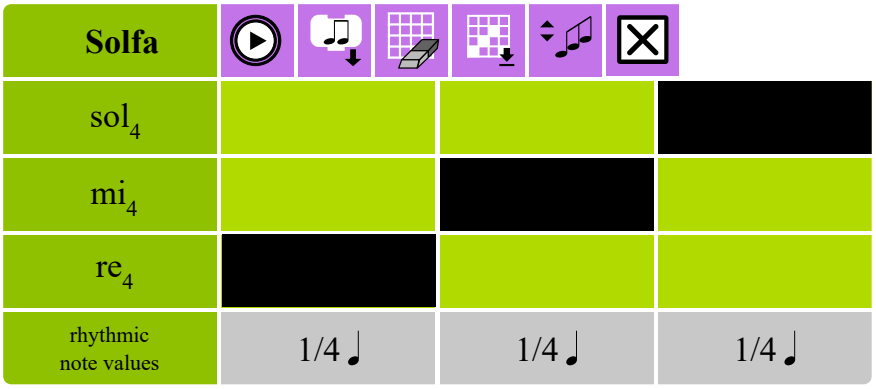
《音乐拼块》拥有一个拼块 *Pitch-time Matrix*, 作为开始写程序的蓝图。

在浏览器中启动《音乐拼块》后，点击开始在屏幕中间出现的 *Pitch-time Matrix* 堆栈。(暂时忽视 *Start* 拼块) 你会看到一个网格按节奏纵向组织，和按节奏横向组织。

Solfa			
sol_4			
mi_4			
re_4			
rhythmic note values	1/4 ♩	1/4 ♩	1/4 ♩

上图中的矩阵有三个 *Pitch* 块和一个 *Rhythm* 块，用于创建一个3 x 3 音高和时间的网格。

请注意，原本矩阵有五个 *Pitch* 块，因此，你会看见五行，每个音调一行。 （底部第六行用于指定与每个音符关联的节奏）。 也在原本情况下，有两个 *Rhythm* 块，它指定了六分之四音符之后是一半音符。 由于 *Rhythm* 块在 *Repeat* 块里面，有14（2 x 7）列供选择音符。



通过点击网格中的单元格，您应该可以听到几个单个音符（或和弦，如果你点击一个以上的单元格。 在这个图中，三个四分之一的音符被选中（黑色格子）， `Re 4`, 接着 `Mi 4` ,最后 `Sol 4`.

如果你点击 *Play* 按钮 (在格子中的第一行),你会听到一系类的音符被播放 (从左到右): `Re 4`, `Mi 4`, `Sol 4`.

如果你有一系列你喜欢的的音符 (一个 "chunk"), 点击 *Save* 按钮(在 *Play* 按钮的右边)。《音乐拼块》会用程序产生同样一系列的拼块来播放这些音符。 (以下有更多详情.)

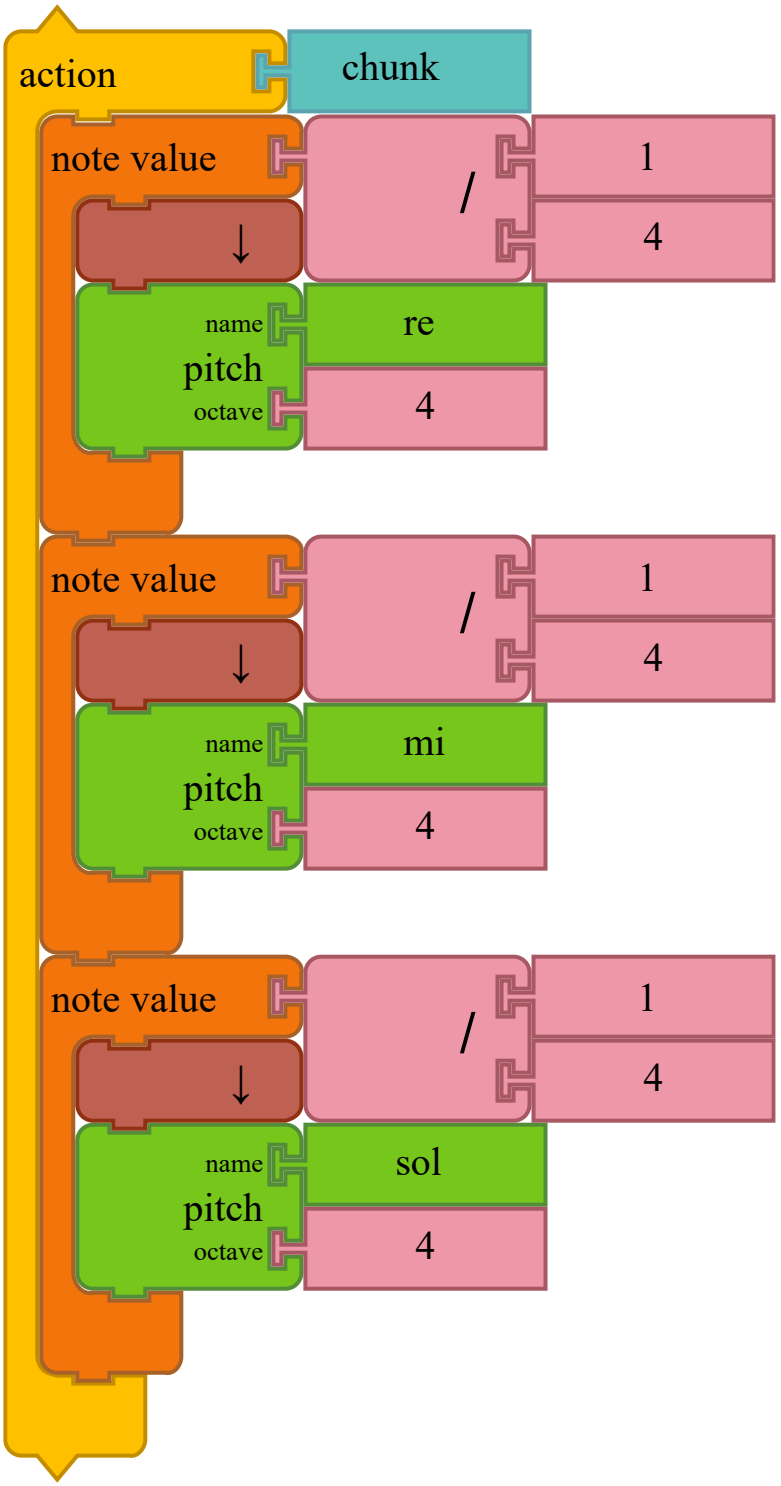
你也可以使用格子从新安排锁定的音符，保存其他的拼块。

Sort 按钮将把所有格子里的音调根据高低从新安排，出掉多余的 *Pitch* 拼块.

你可以把格子藏起来，只需要按 *Close* 按钮 (在最上面的一排，最右手边的按钮)

还有一个删除按钮，将清除网格。 你可以随时重新打开矩阵（它会记住的它以前的状态）。因为你可以定义你想要多少拼块，你可以放心随时试验。

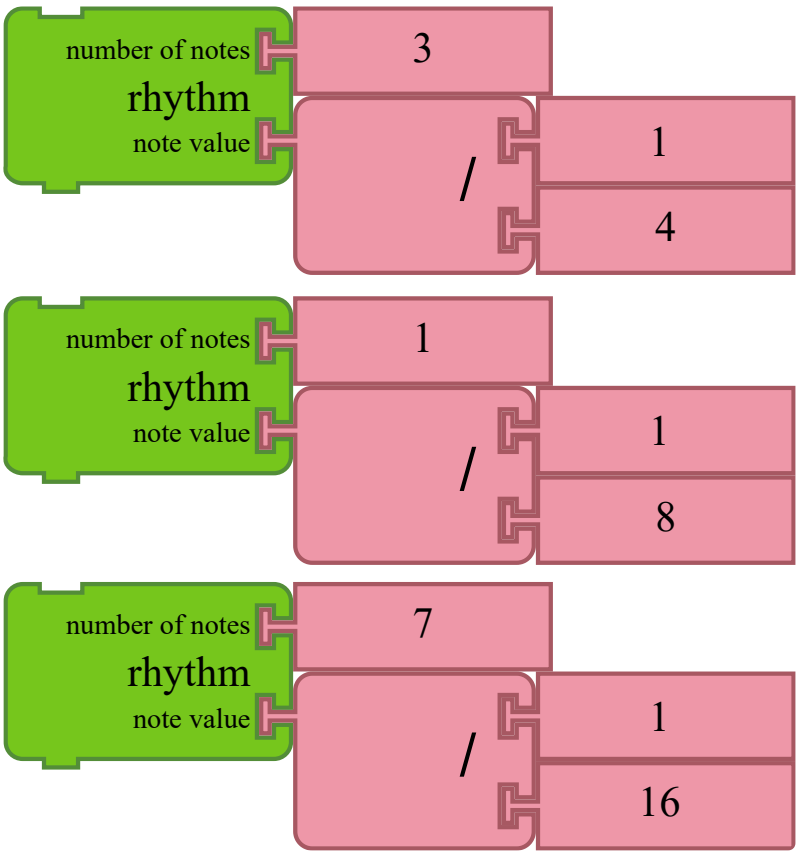
提示：您可以在 *Pitch-time Matrix* 拼块中放置一个拼块来生成该矩阵对应于该块。



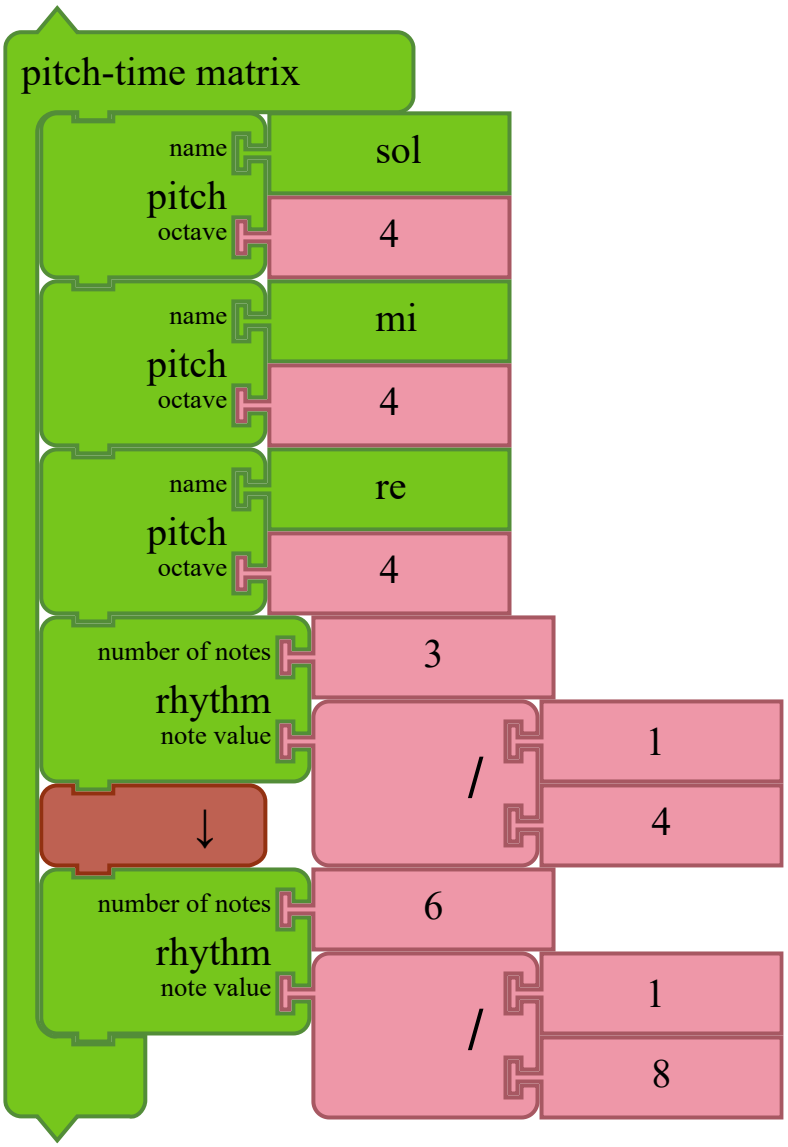
单击矩阵时创建的块是一堆拼块。 这些块是嵌套的：一个 *Action* 拼块包含三个 *Note Value* 拼块，每块包含一个 *Pitch* 拼块。 那个 *Action* 拼块有一个由矩阵自动生成的名字（您可以通过单击名称来重命名该操作）。 每个音符有一个持续时间（在这种情况下是4，代表四分音符）。 你可以尝试把不同的数字放进去，看看（听到）会发生什么。 每个音符块也有一个音调块（如果它是一个和弦，就会有多个 *Pitch* 块嵌套在音符块的夹子内）。 每音调块有一个音调名称（ `Re` ， `Mi` 和 `Sol` ）和一个音调八度; 在这个例子中，每个音高的八度是4。（尝试改变音高名称和音高八度）。

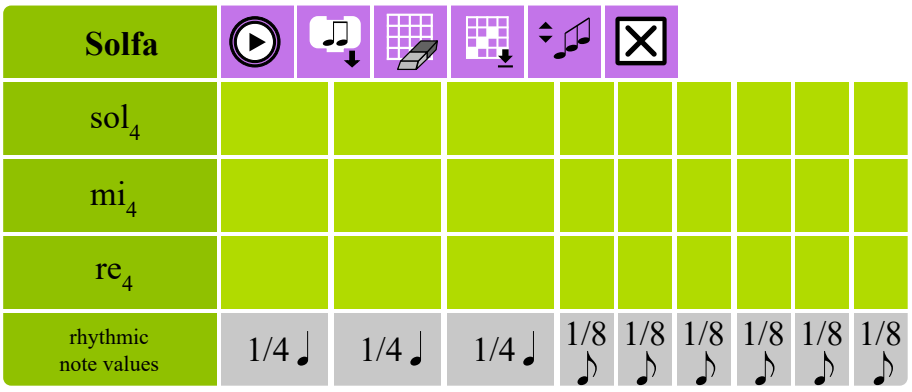
播放拼块，只需点击动作拼块（在单词上行动）。 你应该听到从上到下排列的音符演奏。

4.2.2 音律拼块



Rhythm 块用于生成节奏模式 *Pitch-Time Matrix* 块。 *Rhythm* 块的最高参数是音符的数量，底部的参数是的持续时间。 注意：在上面的例子中，三个四分音符列将在矩阵中生成。 在中间的例子中，一列八分音符会被产生。 在底部的例子中，七个列将生成16个音符。



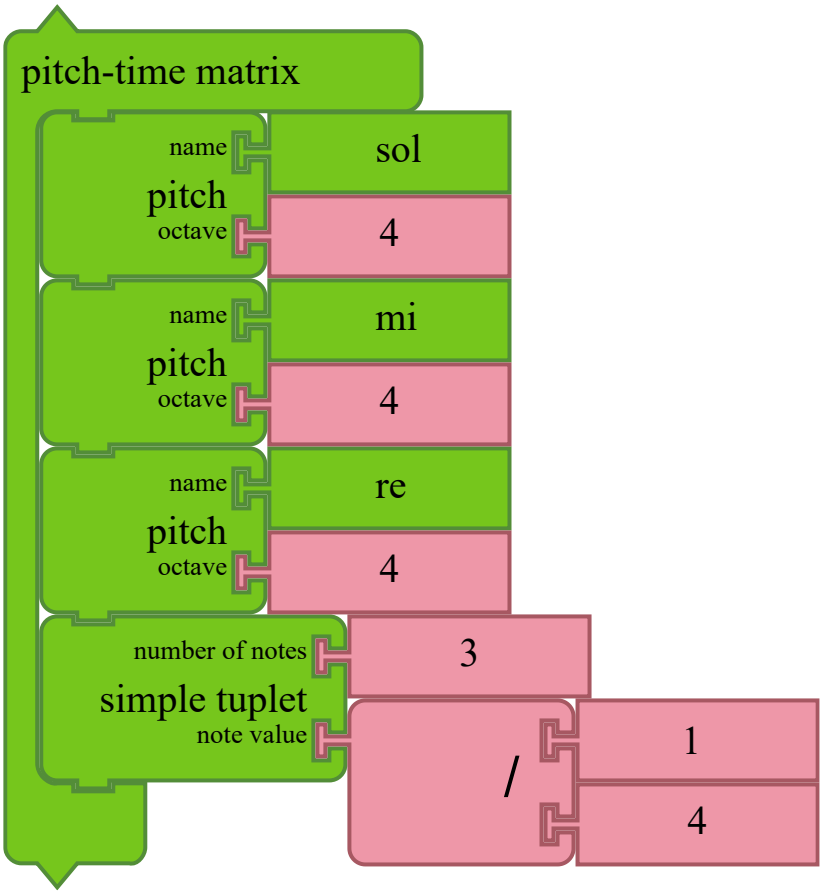


You can use as many *Rhythm* blocks as you'd like inside the

6 block. In the above example, two blocks are used, resulting in three quarter notes and six eighth notes.

你可以在 *Pitch-time Matrix* 拼块里面使用尽可能多的 *Rhythm* 拼块。 在上面的例子中，两个 *Rhythm* 块被使用，产生三个四分音符和六个八分之一音符。

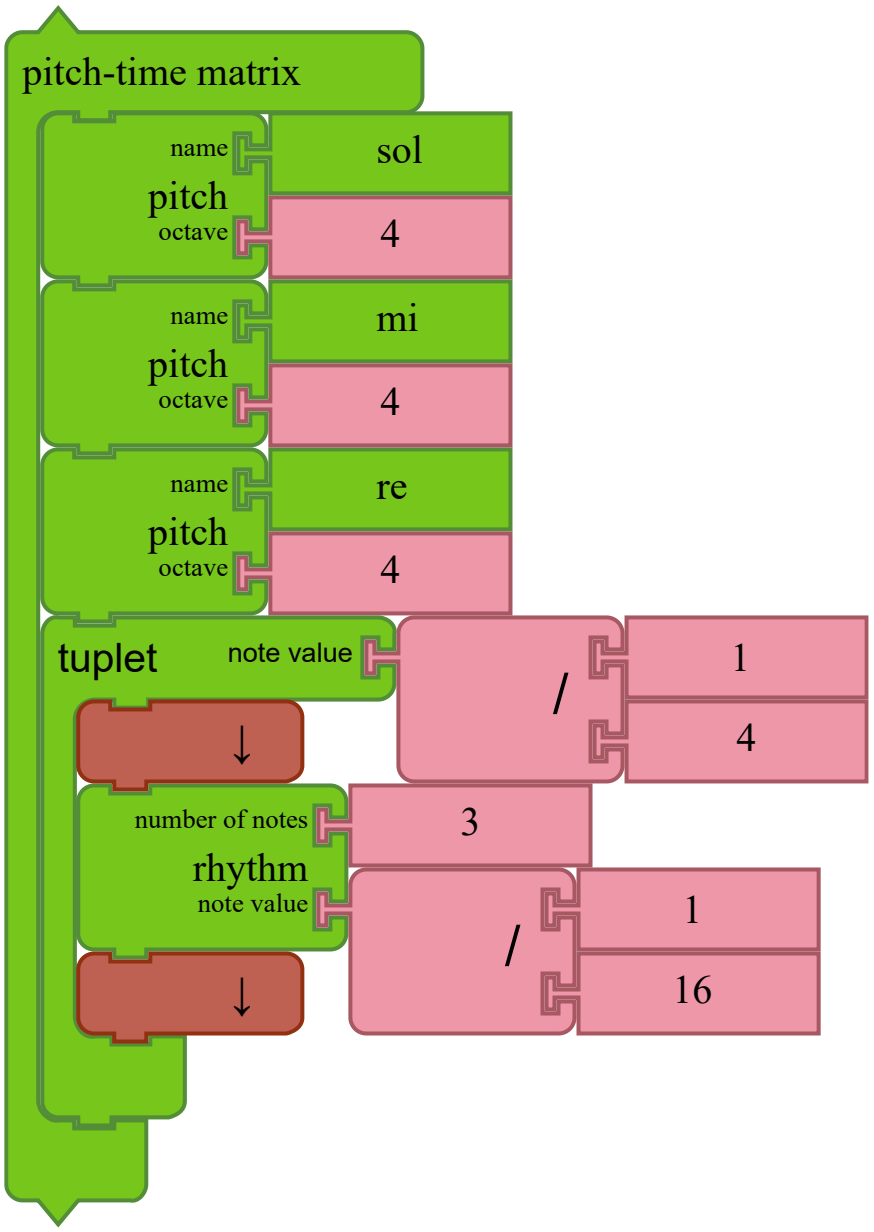
4.2.3 创造连音



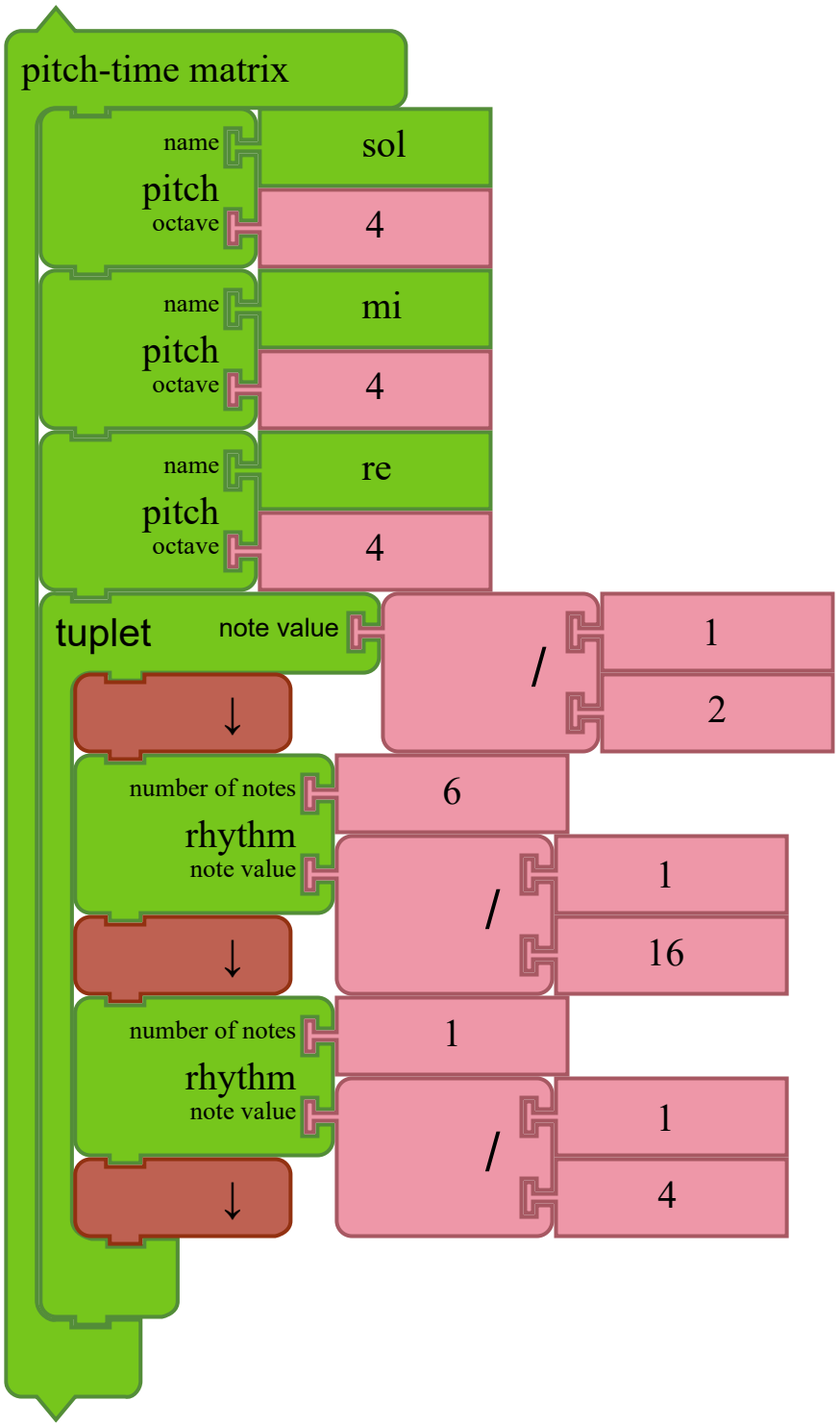
Solfa						
sol_4						
mi_4						
re_4						
tuplet note values	1/12	1/12	1/12			
tuplet value	3					
rhythmic note values	1/4 ♩					

连音符是可以缩放到特定的音符集合持续时间。 使用连音符可以很容易地创建一组不是基于 2 的音符。

在上面的例子中，三个四分音符（定义在 *Simple Tuplet* 拼块）在四分之一时间播放。 注意：结果是三个十二分音符。 （这个形式，这是在音乐中相当的常见，被称为 *triplet* 。 其他常见的连音符包括 *quintuplet* 和 *septuplet*。）



在上面的例子中，三个四分音符是在中定义的*Rhythm* 块嵌入 *Tuplet* 块。 与 *Simple Tuplet* 例如，他们是在一个季度的时间播放. 注意。结果是三个十二分音符。 这个更复杂的形式允许用于混合单个连音符中的多个节奏。

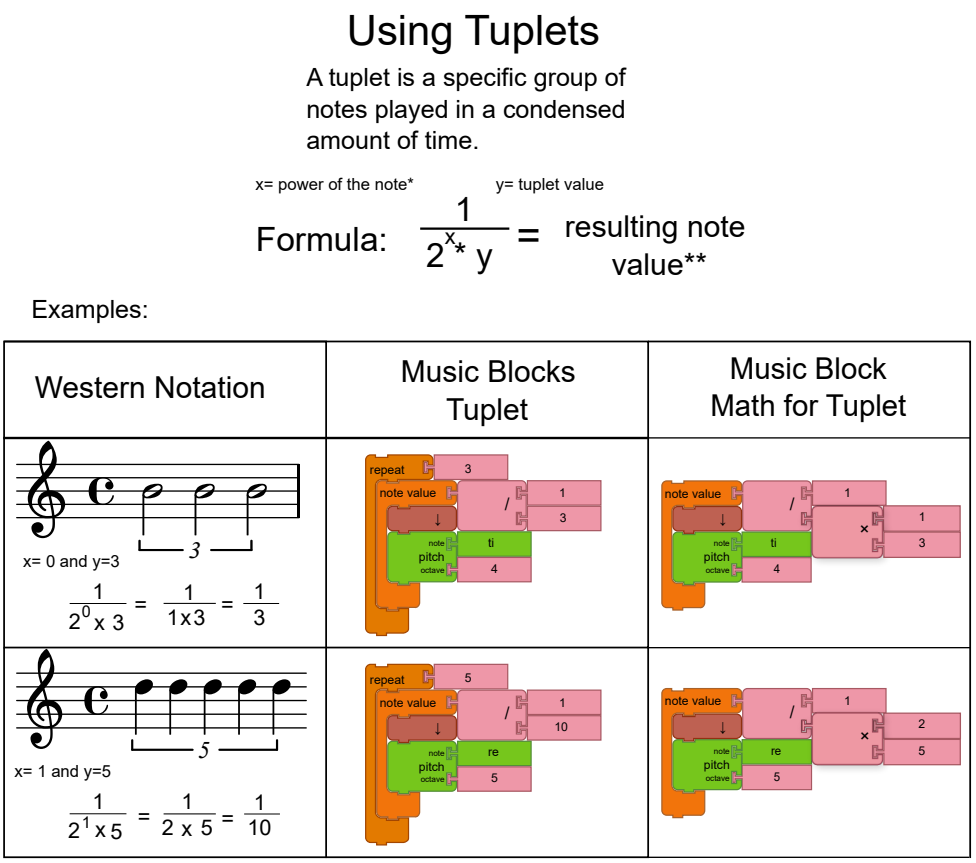

pitch-time matrix
name
sol
pitch
octave
4
name
mi
pitch
octave
4
name
re
pitch
octave
4
tuplet
note value
/
1
2
number of notes
6
rhythm
note value
/
1
16
number of notes
1
rhythm
note value
/
1
4

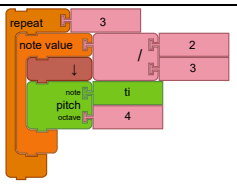
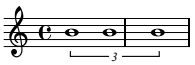
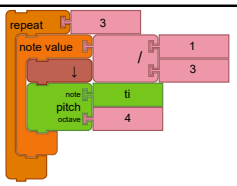
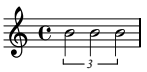
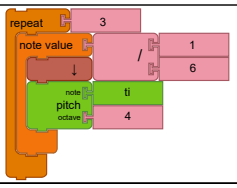
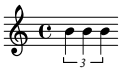
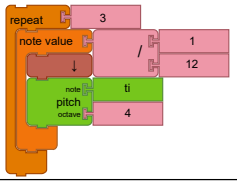
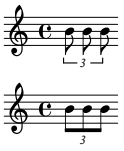
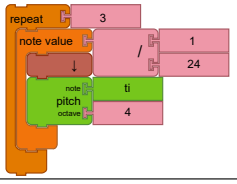
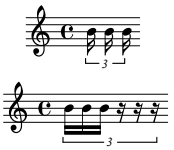
Solfa							
sol_4							
mi_4							
re_4							
tuplet note values	1/20	1/20	1/20	1/20	1/20	1/20	1/5
tuplet value	7						
rhythmic note values	1/2						

在上面的例子中，两个 *Rhythm* 块被嵌入到 *Tuplet* 拼块，导致更复杂的节奏。

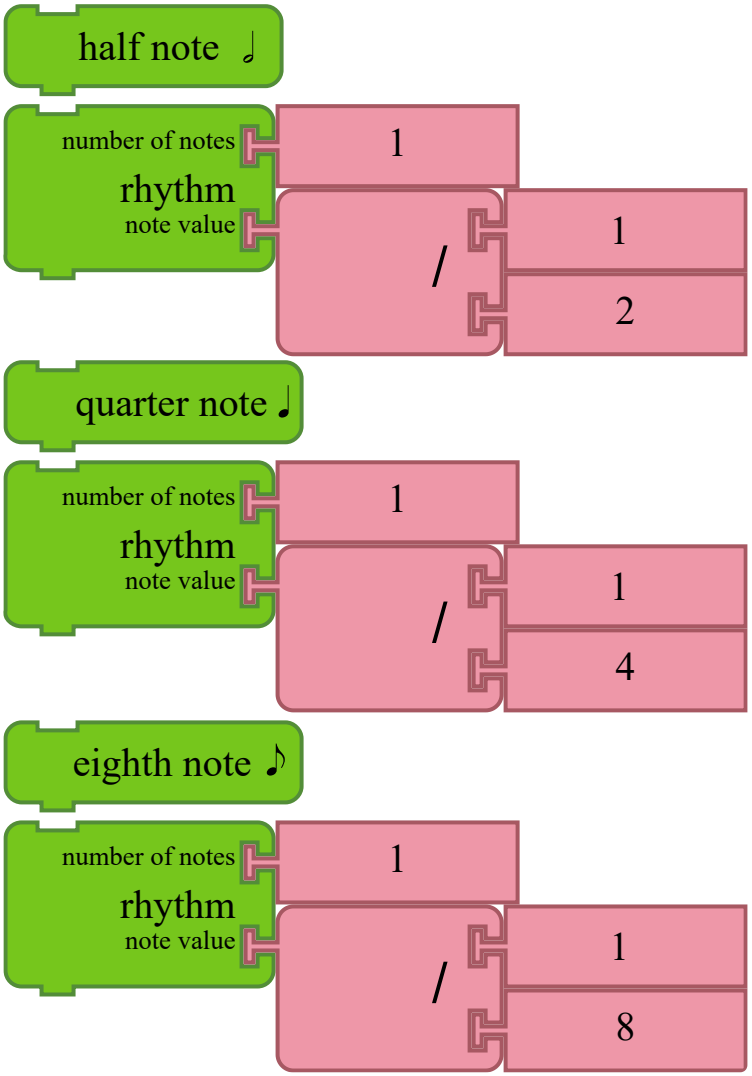
注意：您可以混合搭配 *Rhythm* 块和 *Tuplet* 拼块定义你的矩阵。

4.2.4 连音是什么?



Power of Two	Music Blocks Tuplet	Tuplet in Western Notation
$\frac{1}{2^{-1} \times 3} = \frac{2}{3}$		
$\frac{1}{2^{0} \times 3} = \frac{1}{3}$		
$\frac{1}{2^{1} \times 3} = \frac{1}{6}$		
$\frac{1}{2^{2} \times 3} = \frac{1}{12}$		
$\frac{1}{2^{3} \times 3} = \frac{1}{24}$		

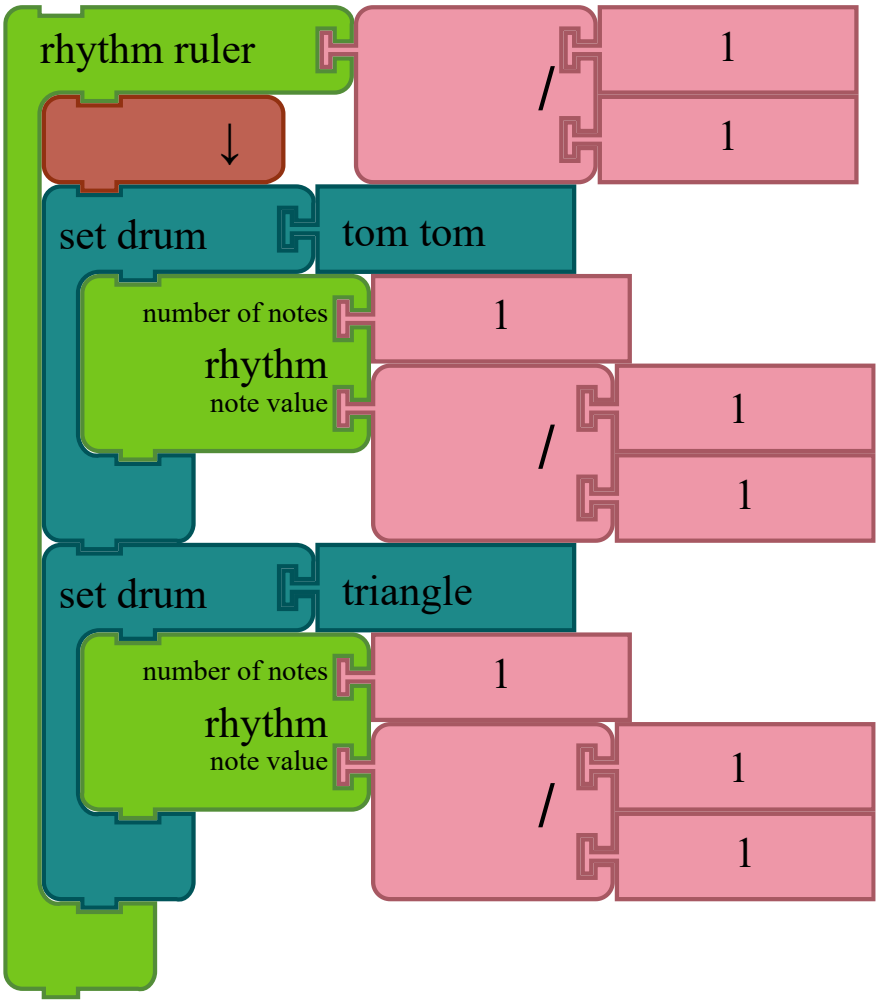
4.2.5 在矩阵里使用独自音符



定义网格时，您还可以使用单个的音符。这些块将会扩展到具有相应值的 *Rhythm* 拼块。

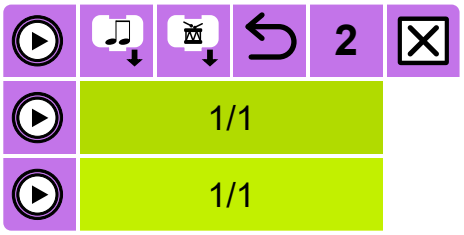
4.3 产生音律

Rhythm Ruler 拼块可以用来启动一个类似的小部件 - *Pitch-Time Matrix* 拼块。 小部件可以用来生成有节奏的图案。

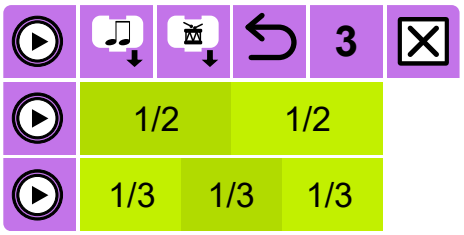


Rhythm Ruler 拼块的参数指定的持续时间将被细分以产生节奏模式。 原本情况下，它是1/1（整个音符）。

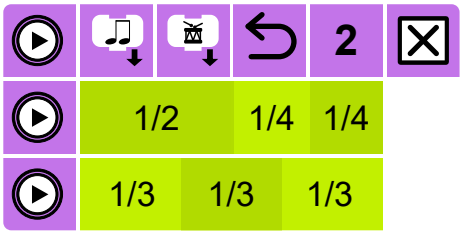
包含在 *Rhythm Ruler* 的夹子中的 *Set Drum* 拼块表示要同时定义的节奏数量。 在原本情况下，定义两个节奏。 嵌入的 *Pitch* 块定义每个节奏标尺的初始细分。



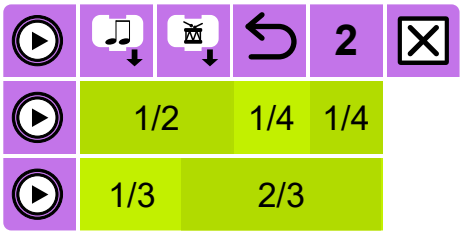
当 *Rhythm Ruler* 拼块被点击时，*Rhythm Ruler* 部件会被打开。 它包含每个节奏标尺的一行。 顶部的输入小部件来指定将有多少个细分点击时在单元格内创建。 原本情况下，2个细分被创建。



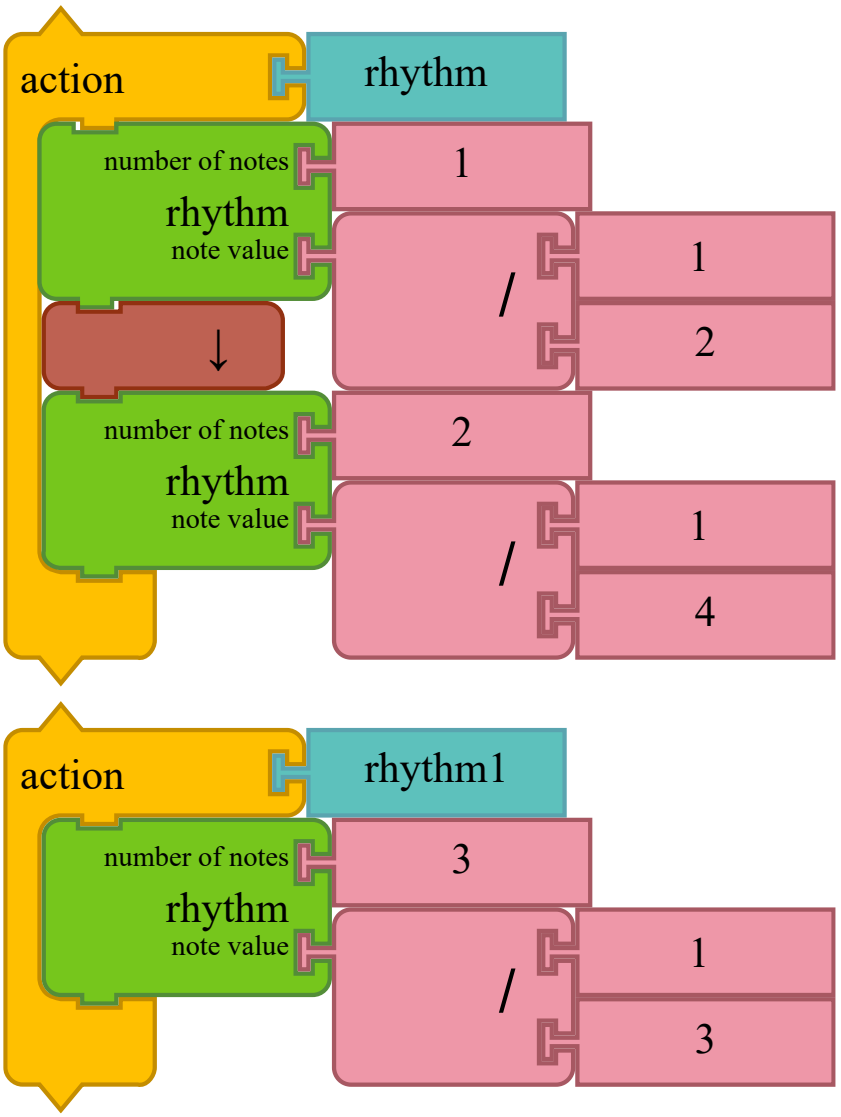
如上图所示，顶级节奏标尺已被分割分成两个半音和底部的节奏尺被分成了三个三分音符。 点击每一行左边的 *Play* 按钮将使用鼓为每个节拍播放节奏。 *Play-all* 小部件左上角的按钮将同时播放所有的节奏。



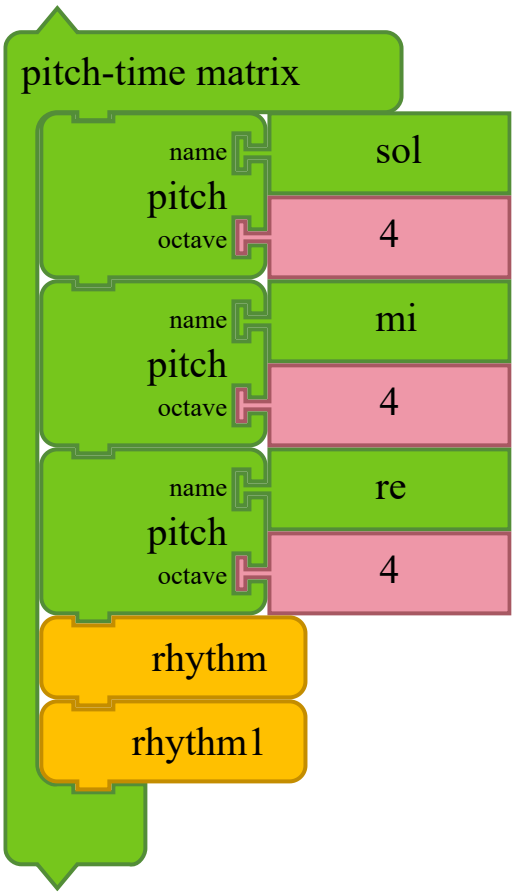
节奏可以通过单击进一步细分格子。 在上面的例子中，创建了两个四分音符点击其中一个半音。



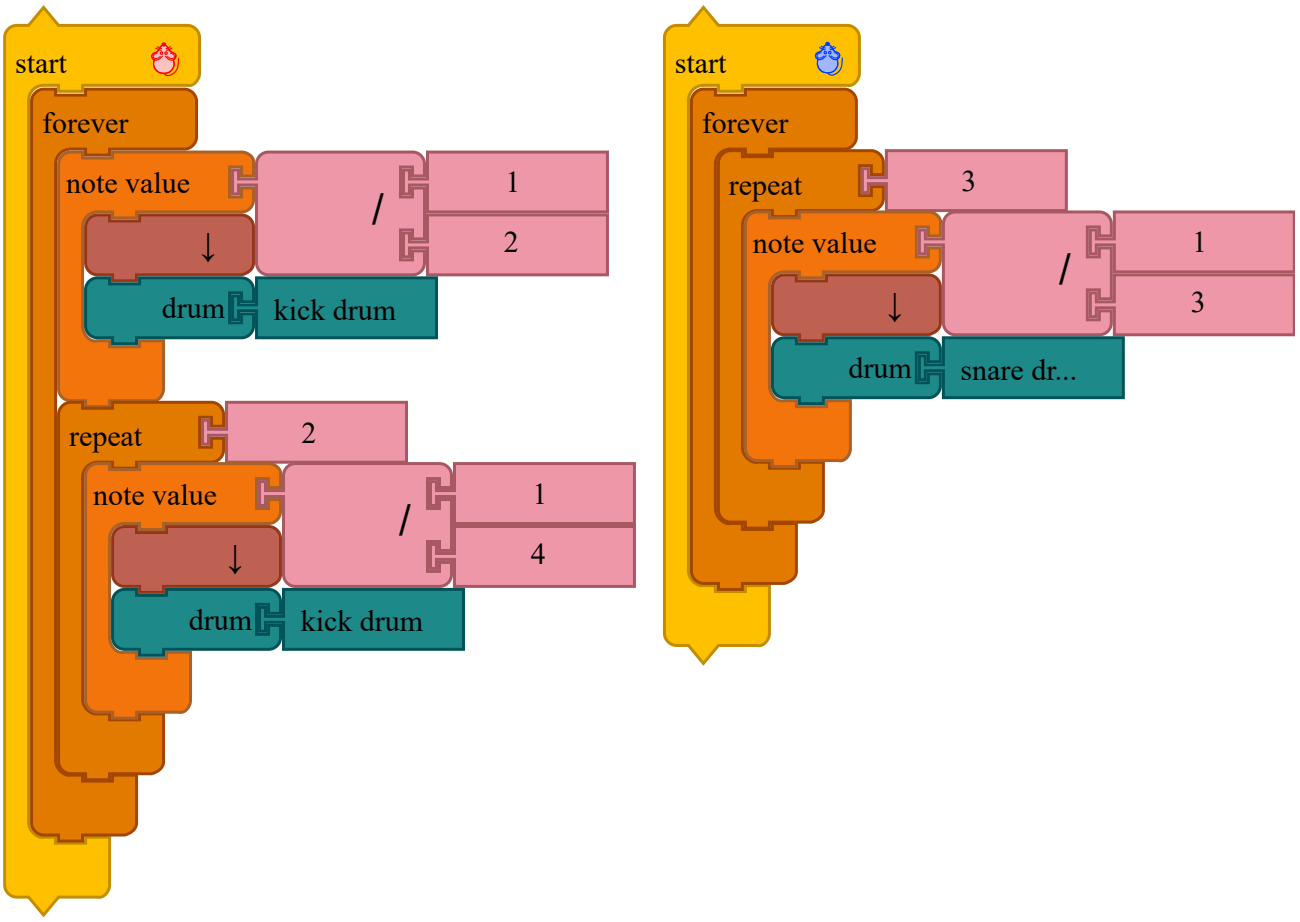
通过拖曳多个单元格，它们会被绑在一起。 在这个上面的例子，两个三分音符已经被并入三分之二音符。



Save Stack 按钮将导出节奏堆栈。



这些节奏叠加可以用来定义使用的节奏模式与 *Pitch-time Matrix* 拼块。



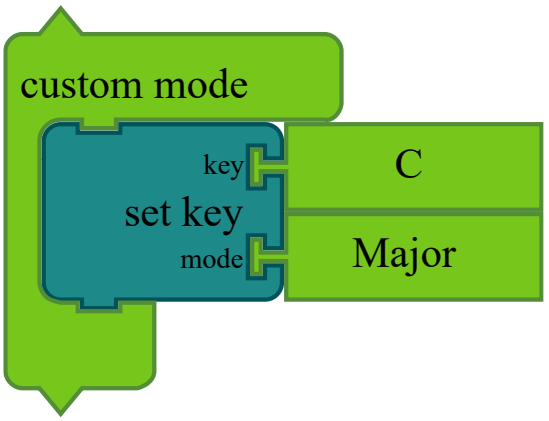
Save Drum Machine 按钮将导出 *Start* 堆栈会像鼓机一样播放节奏。

4.4 音乐模式

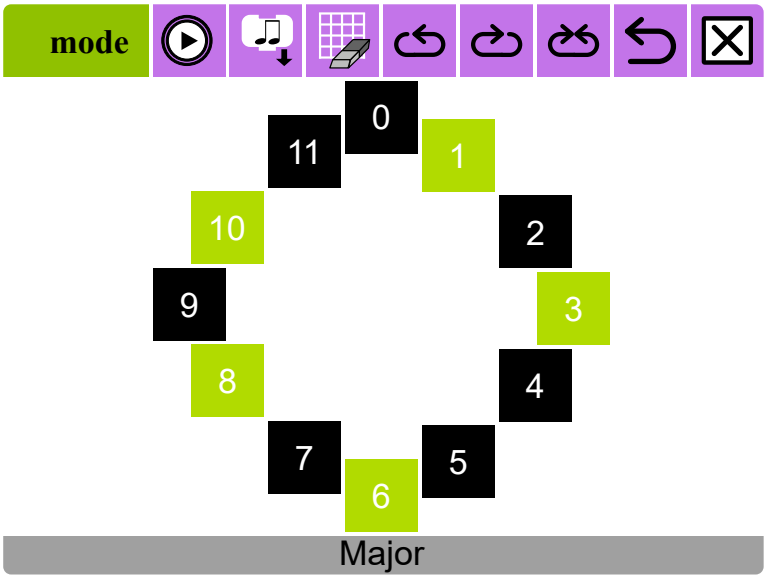
音乐模式被用来指定 [间隔]（#INTERVALS-AND-ARTICULATION）（或步骤）。 因为西方音乐基于12个半步每个八度，模式指定了每个规模之间有多少个半步。

原本情况下，音乐块使用 *Major* 模式，在 [Key]（#SETTING）中C，映射到钢琴上的白键。 *Major* 模式是 “2,2,1,2,2,2,1”。 其他许多常见的模式是内置《音乐拼块》，当然包括 *Minor* 模式，使用 '2，1，2，2，1，2，2' 作为它的区间。

请注意，并非每个模式都使用每个八度的7个音程。 例如，*Chromatic* 模式使用11个区间：“1,1,1,1,1,1,1,1,1,1,1,1'。 *Japanese* 模式只使用5个间隔： `1，4，2，3，2]，` 。 重要的是间隔的总和在一个八度是12个半步.



Mode 小部件可以让你探索模式和生成自定义模式。 使用 *Custom Mode* 拼块调用小部件，模式在 *Set Key* 指定的块将是小部件启动的时候的原本模式。



在上面的例子中，小部件已经以 *Major* 模式启动（原本设定）。 请注意，该模式中包含的音符由标示黑色的盒子，它们排列成圆形的tweleve图案半步完成八度。

由于 *Major* 模式的时间间隔是 “2,2,1,2,2,2,1”，因此，音符是
“0”，“2”，“4”，“5”，“7”，“9”，“11” 和 “12”。（高一个八度）

小部件控件沿顶部的工具栏运行。从左到右-

Play all, 这将使用当前模式进行播放;

Save，将当前模式保存为 *Custom* 模式并保存一堆 *Pitch* 拼块可以和 *Pitch-Time Matrix*拼块一起使用。

Rotate counter-clockwise, 这将旋转模式逆时针（见下面的例子）;

Rotate clockwise, 这将顺时针旋转模式（请参阅下面的例子）;

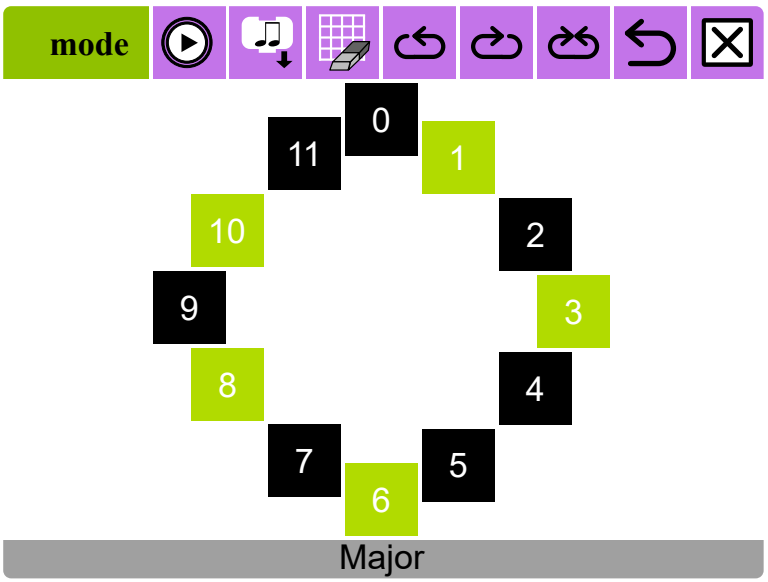
Invert, 这将反转模式（见下面的例子）;

Undo, 这会将模式恢复到以前的版本;和

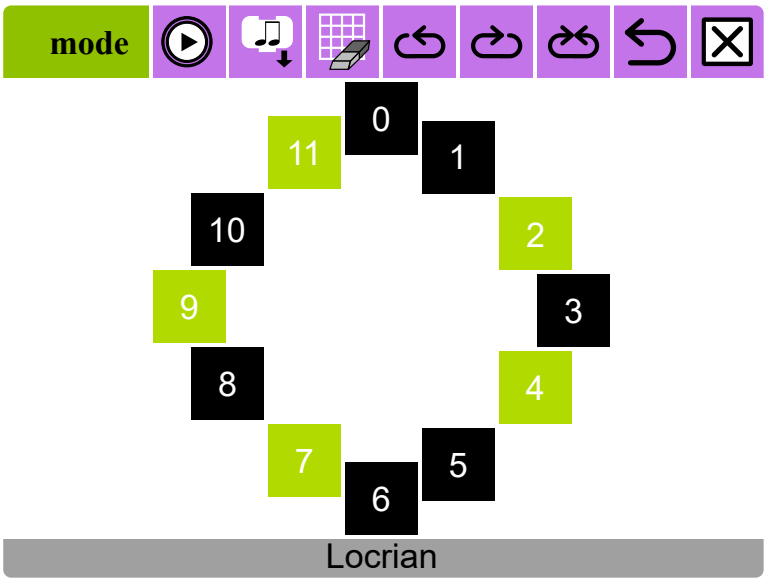
Close, 这将关闭小部件。

您也可以单击单个笔记来激活或停用它们。

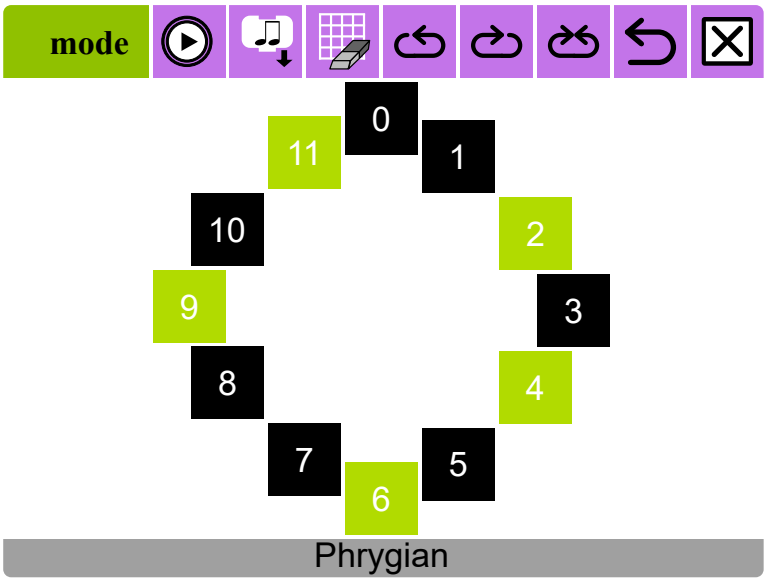
请注意，*Custom Mode* 拼块内的模式会随时更新 小部件内部的模式被改变



在上面的例子中，*Major* 模式已经顺时针旋转，把它变成 *Dorian* 。

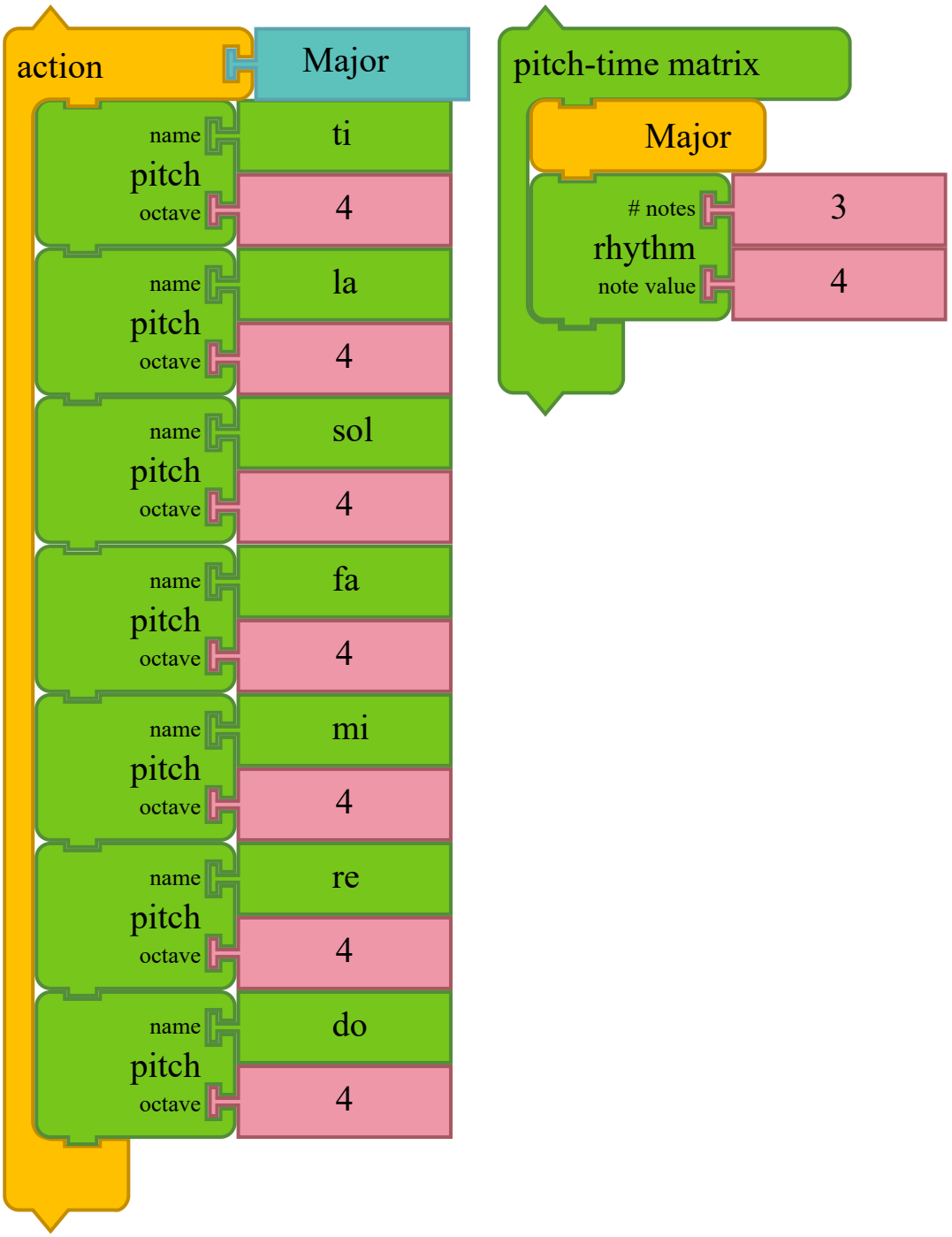


在上面的例子中，*Major* 模式已被旋转逆时针，将其转换成 *Locrian*。



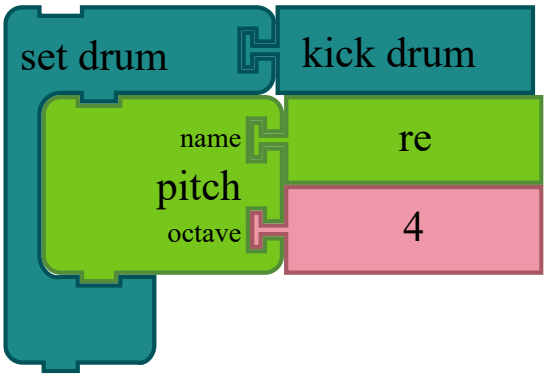
在上面的例子中，*Major* 模式已经被反转，转换它成 *Phrygian*。

注意：《音乐拼块》中的内建模式可以在 [musicutils.js] 中找到 (https://github.com/sugarlabs/musicblocks/blob/master/js/utils/musicutils.js#L68 (https://github.com/sugarlabs/musicblocks/blob/master/js/utils/musicutils.js#L68)).



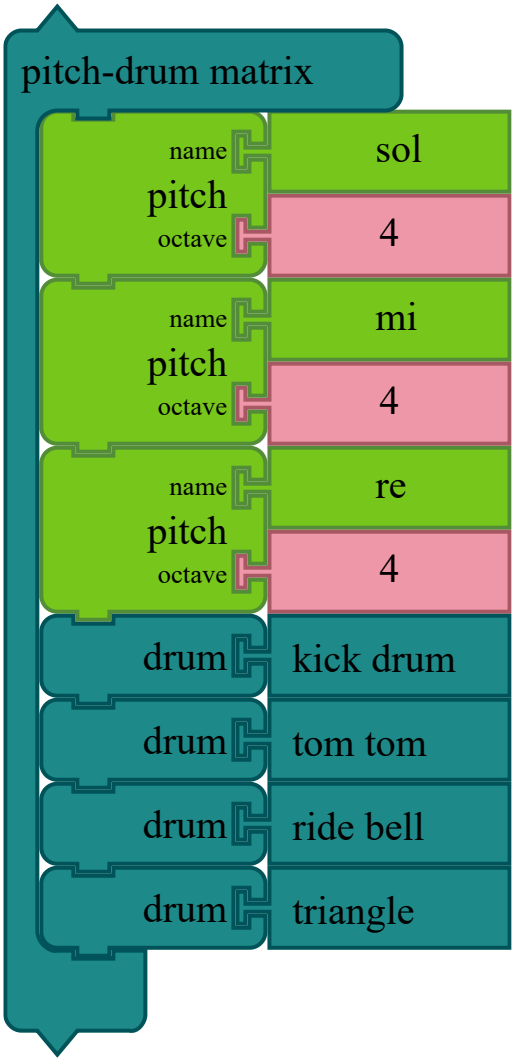
Save 按钮导出一堆代表模式的块，可以在 *Pitch-Time Matrix* 块中使用。

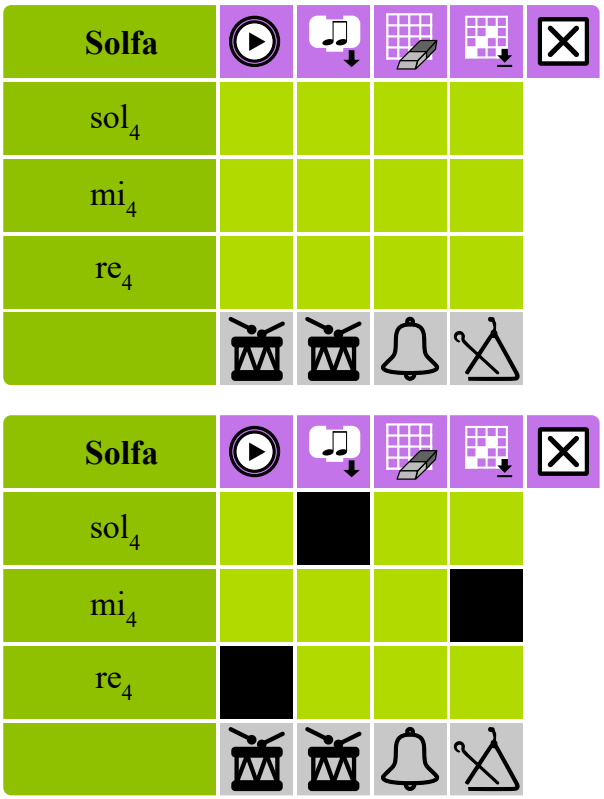
4.5 音调-鼓声矩阵

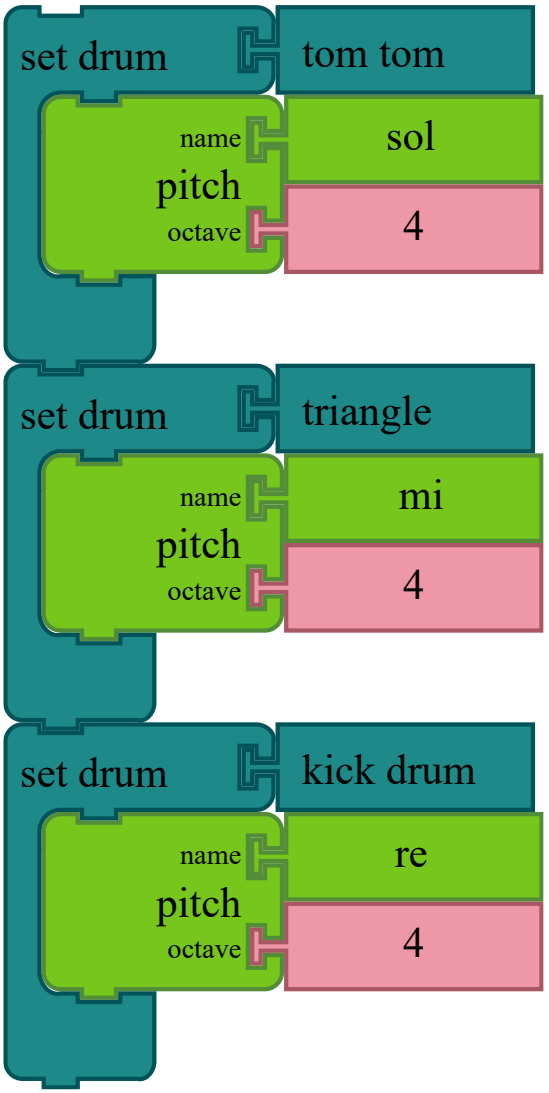


The *Set Drum* block is used to map the enclosed pitches into drum sounds. Drum sounds are played in a monopitch using the specified drum sample. In the example above, a `kick drum` will be substitued for each occurance of a `Re 4`.

Set Drum 块用于将封闭的音高映射到鼓中声音。 鼓声使用指定的鼓在单声道中播放。 在上面的例子中，一个 `kick drum` 将被代替每个 `Re 4` 的发生。




Solfa
sol4
mi4
re4
Solfa
sol4
mi4
re4



作为用 *Set Drum* 块创建映射的经验，我们提供 *Drum-Pitch* 矩阵。 你用它来映射间距和鼓。 输出是一个 *Set Drum* 块的堆栈。

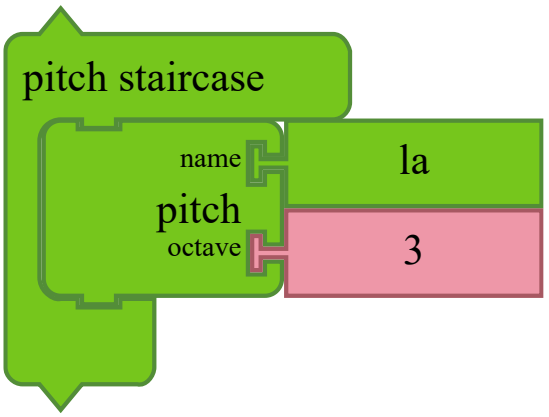
4.6 探索音调比例

The *Pitch Staircase* block is used to launch a widget similar to the *Pitch-time Matrix*, which can be used to generate different pitches using a given pitch and musical proportion.

The *Pitch* blocks contained in the clamp of the *Pitch Staircase* block define the pitches to be initialized simultaneously. By default, one pitch is defined and it have default note "la" and octave "3".

Pitch Staircase 块是用来启动一个类似于 *Pitch-Time Matrix*，可以用来产生不同的音高使用给定的音调和音乐比例的拼块。

包含在 *Pitch Staircase* 的夹子中的 *Pitch* 块同时定义音高初始化。 一个音高开始被定义并且具有默认音符 “la” 和八度音阶 “3”。



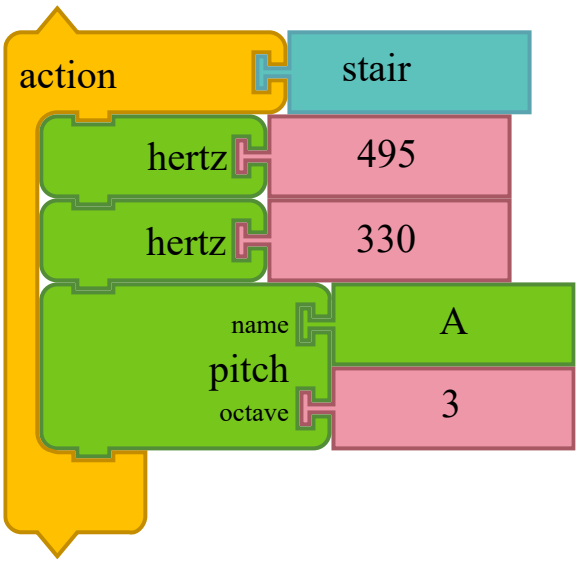
当 *Pitch Staircase* 块被点击时，*Pitch Staircase* 部件是初始化。 该小部件包含每个 *Pitch* 块包含的行在 *Pitch Staircase* 块的夹子中。 小部件顶部的输入字段指定用于创建新的音乐比例在楼梯间。 输入对应于分子和分母的比例。 开始的比例是 3：2。

Clicking on the *Play* button to the left of each row will playback the notes associated with that step in the stairs. The *Play-all* button on the upper-left of the widget will play back all the pitch steps simultaneously. A second *Play-all* button to the right of the stair plays in increasing order of frequency first, then in decreasing order of frequency as well, completing a scale.

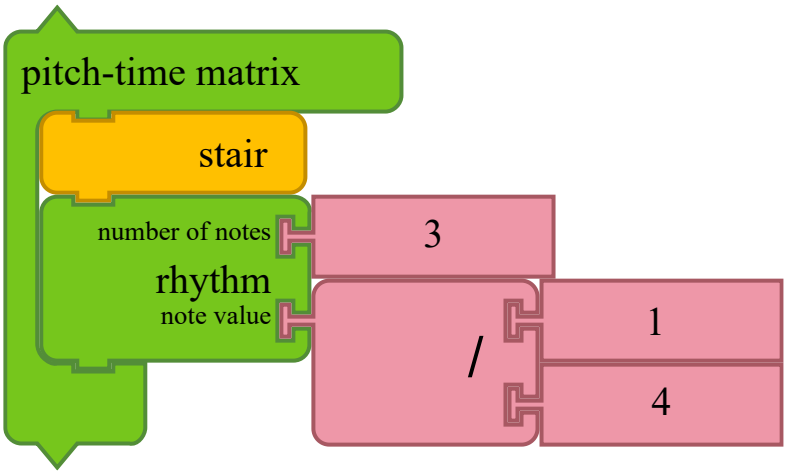
The *Save stack* button will export pitch stacks. For example, in the above configuration, the output from pressing the *Save stack* button is shown below:

点击每行左边的 *Play* 按钮将会播放与在楼梯的那个步骤相关的音符。 *Play-all* 小部件左上角的按钮将同时播放所有音高步骤。 第二个 *Play-all* 按钮按照频率的先后顺序播放，然后进入频率下降的顺序，完成一个规模。

Save Stack 按钮将导出音高堆栈。 例如，在上面配置，按下 *Save Stack* 按钮的输出如下所示：



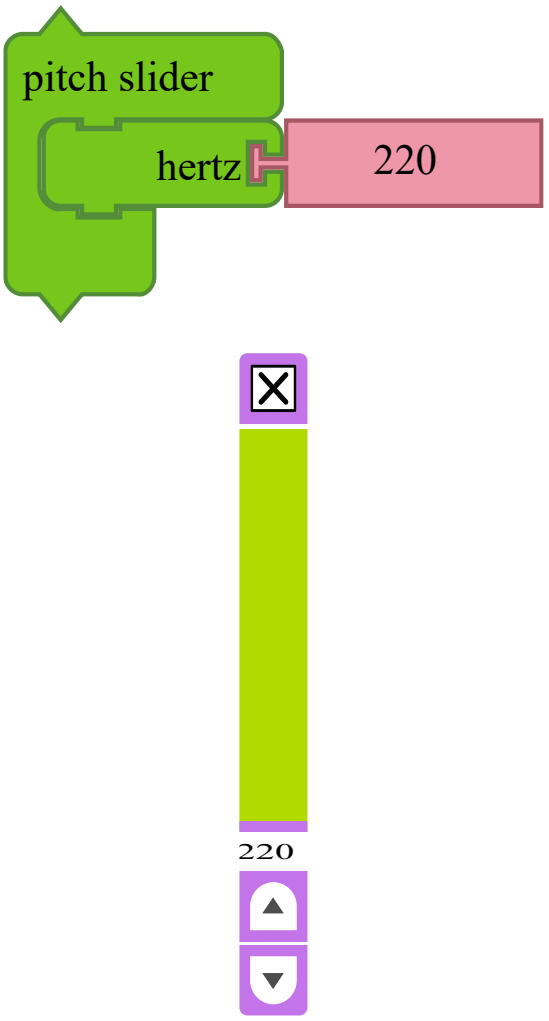
这些堆栈可以与 *Pitch-Time Matrix* 块一起使用来定义矩阵中的行。



4.7 产生音调

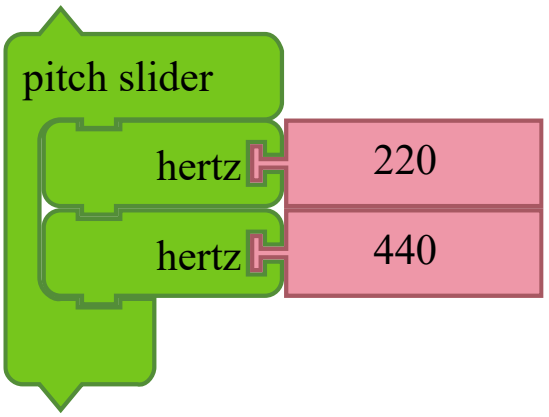
Pitch Slider 块用于启动一个已经习惯的小部件生成任意球。 它不同于 *Pitch Staircase* 部件，它被用来创建频率在不断变化指定八度的范围。

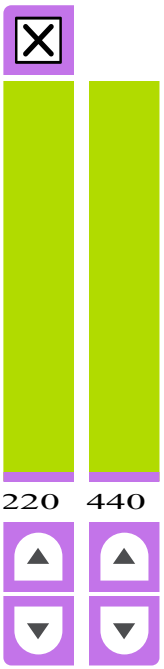
包含在 *Pitch Slider* 块的夹子中的每个 *Sine* 块定义了初始音调的一个八度。



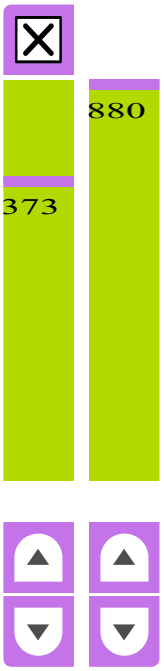
When the *Pitch Slider* block is clicked, the *Pitch Slider* widget is initialized. The widget will have one column for each *Sine* block in the clamp. Every column has a slider that can be used to move up or down in frequency, continuously or in intervals of 1/12th of the starting frequency. The mouse is used to move the frequency up and down continuously. Buttons are used for intervals. Arrow keys can also be used to move up and down, or between columns.

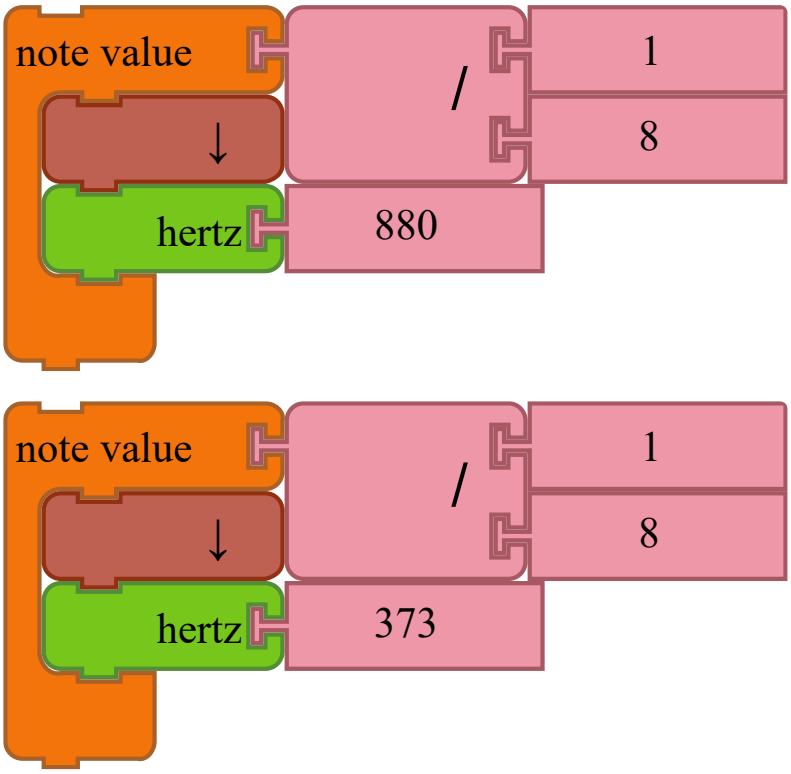
当 *Pitch Slider* 块被点击时，*Pitch Slider* 部件被初始化。 小部件每个 *Sine*块将有一列夹子。 每列都有一个滑块，可用于向上或向上移动频率，连续或间隔的1/12开始频率。 鼠标用于连续上下移动频率。 按钮是用于间隔。 箭头键也可以用来上下移动，或列之间。





点击一列将会提取相应的 *Note* 块，例如：



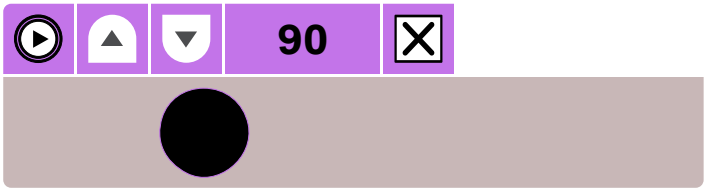


4.8 改换节奏

Tempo 块用于启动一个小部件，使用户能够可视化速度，定义为每分钟节拍（BPM）。 当 *Tempo* 块被点击，*Tempo* 小部件被初始化。

The *Master Beats Per Minute* 块包含在夹子里 *Tempo* 块设置小部件使用的初始速度。 这个确定微件中球的向前移动速度。 如果 BPM 是 60，那么球会移动一秒钟整个小部件，往返需要两秒钟。

小部件的第一行包含 *Play/Pause* 按钮和 *Speed Up* 和 *Slow Down* 按钮，以及一个输入栏用于更新音符播放速度。

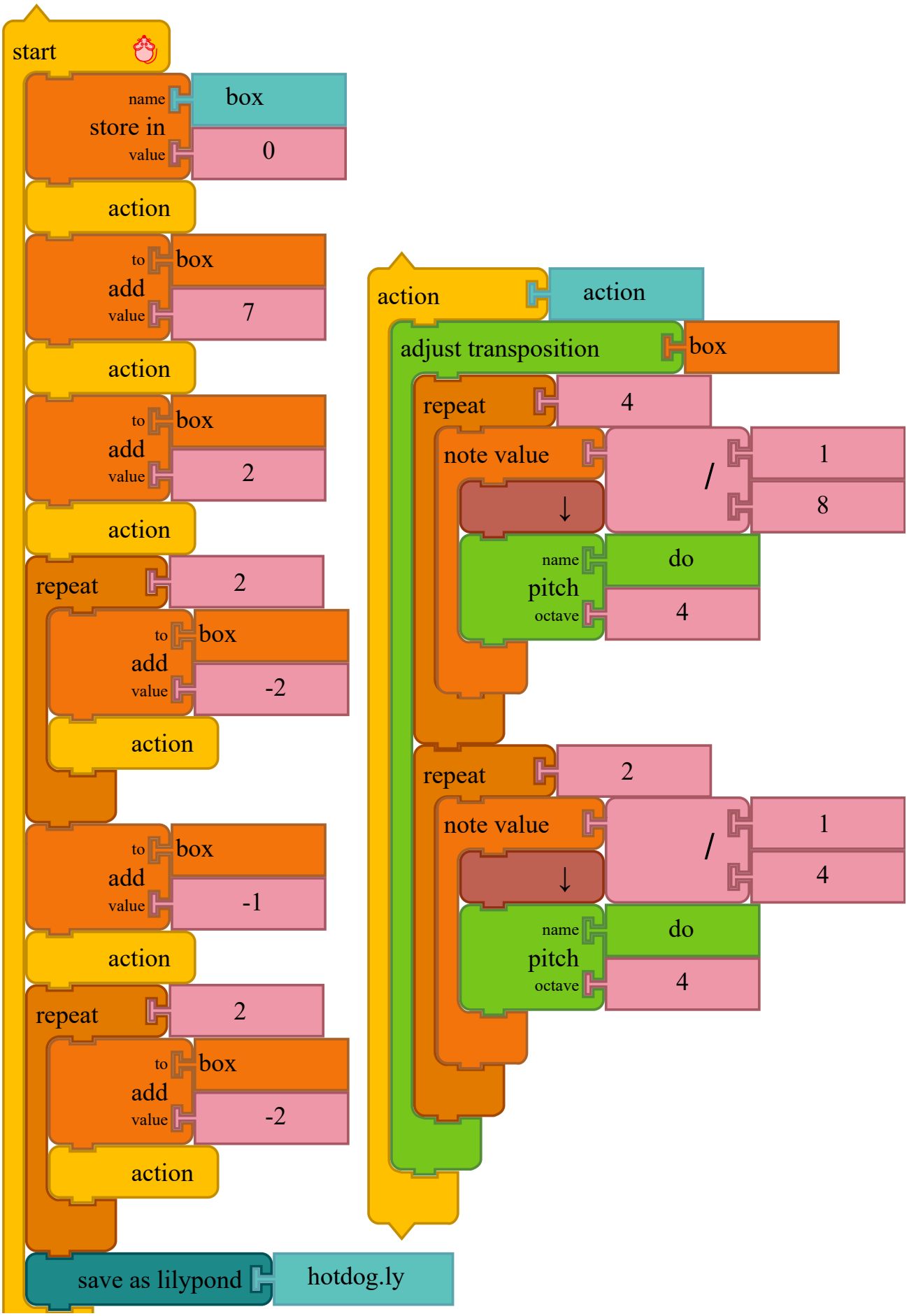


您也可以通过点击小部件间隔连续两次来更新音乐速度：新的BPM被确定为两次点击之间的时间。 例如，如果点击之间有1/2秒，新的BPM将被设置为120。

《音乐拼块》之外

上一章 (4. 部件) | 回去目录

《音乐拼块》 是一个航点，而不是目的地。 其中一个目标是把学习者指向其他强大的工具。 一个这样的工具是 Lilypond (http://lilypond.org), 一个音乐雕刻程序。


start
name
store in
value
box
0
action
to
add
value
box
7
action
to
add
value
box
2
action
repeat
2
to
add
value
box
-2
action
to
add
value
box
-1
action
repeat
2
to
add
value
box
-2
action
save as lilypond
hotdog.ly
action
action
adjust transposition
box
repeat
4
note value
/
1
8
name
pitch
octave
do
4
repeat
2
note value
/
1
4
name
pitch
octave
do
4

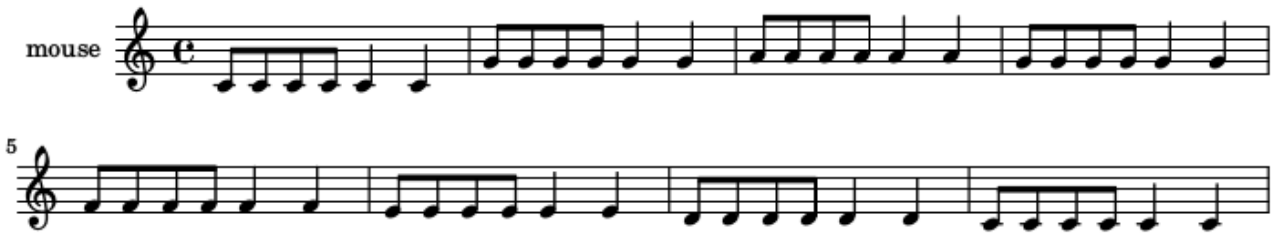
The *Save as Lilypond* block will transcribe your composition. The output of the program above is saved to `Downloads/hotdog.ly` . There is also a button on the secondary toolbar.

Save As Lilypond 块将转录您的作文。 该上面程序的输出保存到 `Downloads / hotdog.ly` 。 你还可以在辅助工具栏上 *Save As Lilypond* 按钮。

```
\version "2.18.2"

mouse = {
c'8 c'8 c'8 c'8 c'4 c'4 g'8 g'8 g'8 g'8 g'4 g'4 a'8 a'8 a'8 a'8 a'4
a'4 g'8 g'8 g'8 g'8 g'4 g'4 f'8 f'8 f'8 f'8 f'4 f'4 e'8 e'8 e'8 e'8
e'4 e'4 d'8 d'8 d'8 d'8 d'4 d'4 c'8 c'8 c'8 c'8 c'4 c'4
}

\score {
<<
\new Staff = "treble" {
\clef "treble"
\set Staff.instrumentName = #"mouse" \mouse
}
>>
\layout { }
}
```



RUN LIVE (https://musicblocks.sugarlabs.org/index.html?id=1523043053377623&run=True)