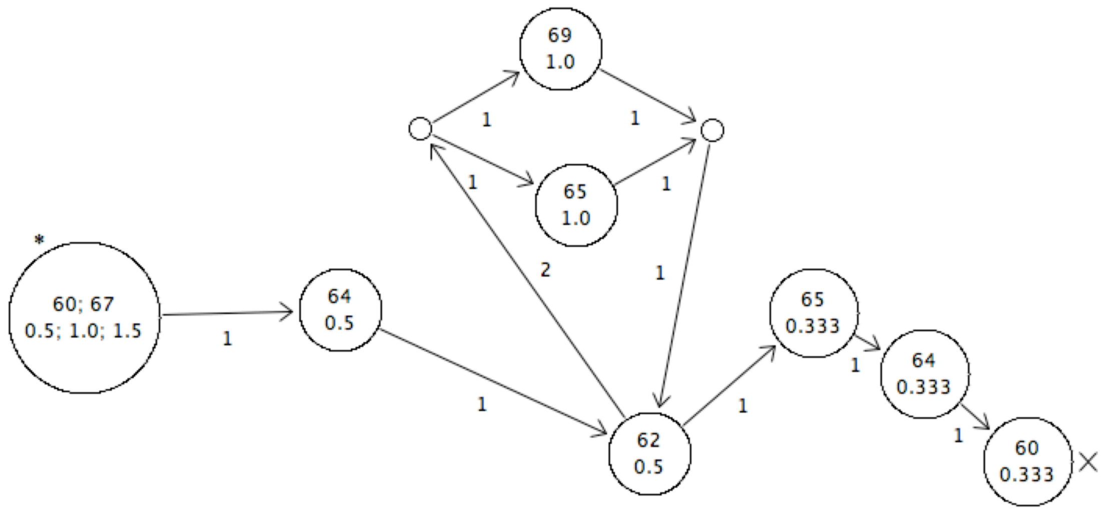
Documentation for “Ramify”

Marc Evans

Ramify is a computer program, created in Java, for Markov-based algorithmic musical composition. It is essentially an editing environment, in which the user creates a network of musical notes and relationships, and the program outputs possible results (or “ramifications”) of that network.

An Introductory Example

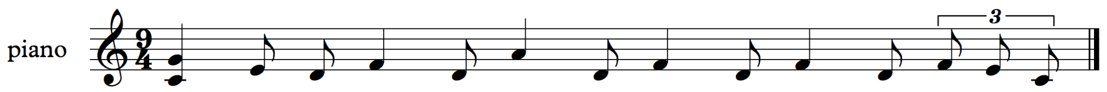
You can see here an example network, or “meta-score”, consisting of several note objects, each with any number of in-going and out-going connections:



The upper number for each note object defines the pitch (using standard midi values) and the lower number defines the length of the note in beats. A note object with multiple pitches - separated by semi-colons – will create a chord, while multiple lengths will result in a random selection from the different options. Thus, the left-most note object seen here will create a dyad between middle C and the G above, with a length of either an eighth-, quarter-, or half-note.

When the program executes, we begin on the note with an asterisk, and proceed along the network, with the each connection's number determining the probability that that path is chosen. Note that these probabilities are normalized, so that if a note has two options, one with probability 2 and the other with probability 1, then the first option has a 2/3 probability of occurring and the second option 1/3. Also notice that sometimes it is useful to create a simple node that takes a number of inputs and sends them to the same outputs. When a note with an X, or with no outgoing connections, is reached, the line terminates.

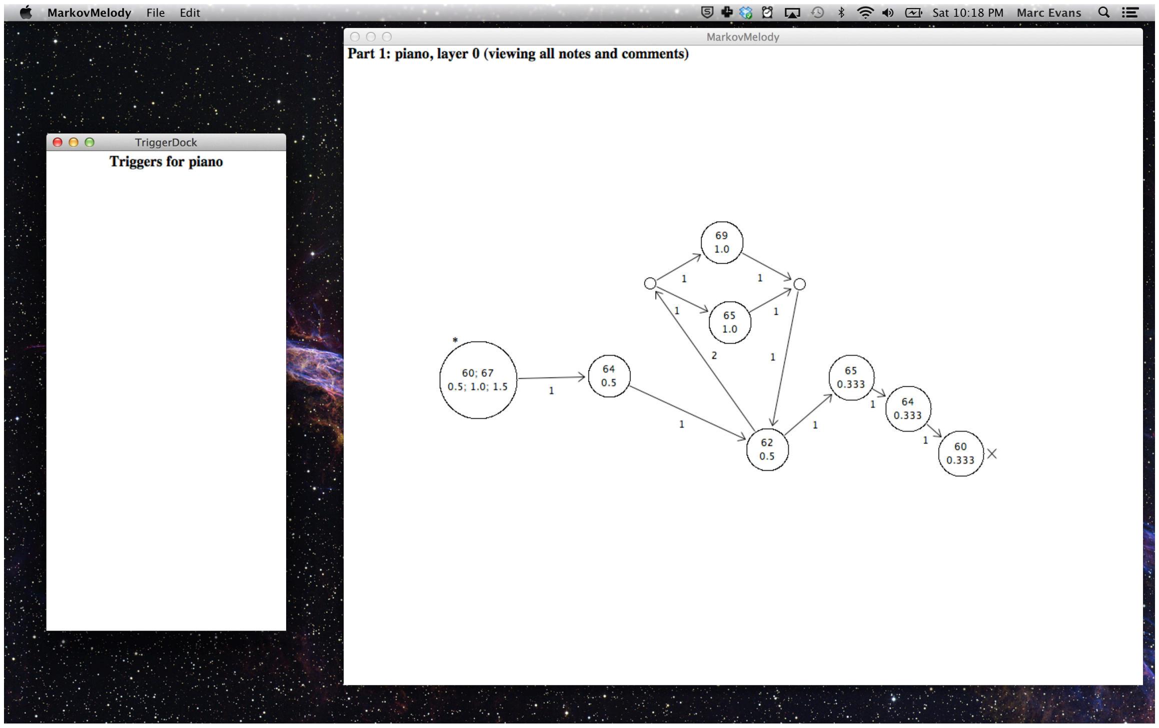
We can generate one possible outcome of this network by typing control-p, entering a few parameters, and saving the result as a music XML file. Let's see what results from this simple program:



The middle portion of this hopelessly uninteresting melodic line is of indeterminate length; this corresponds to the loop in the meta-score.

Basic Editing

The program consists of two windows: the main window holds the network itself, and the side window displays supplemental information.



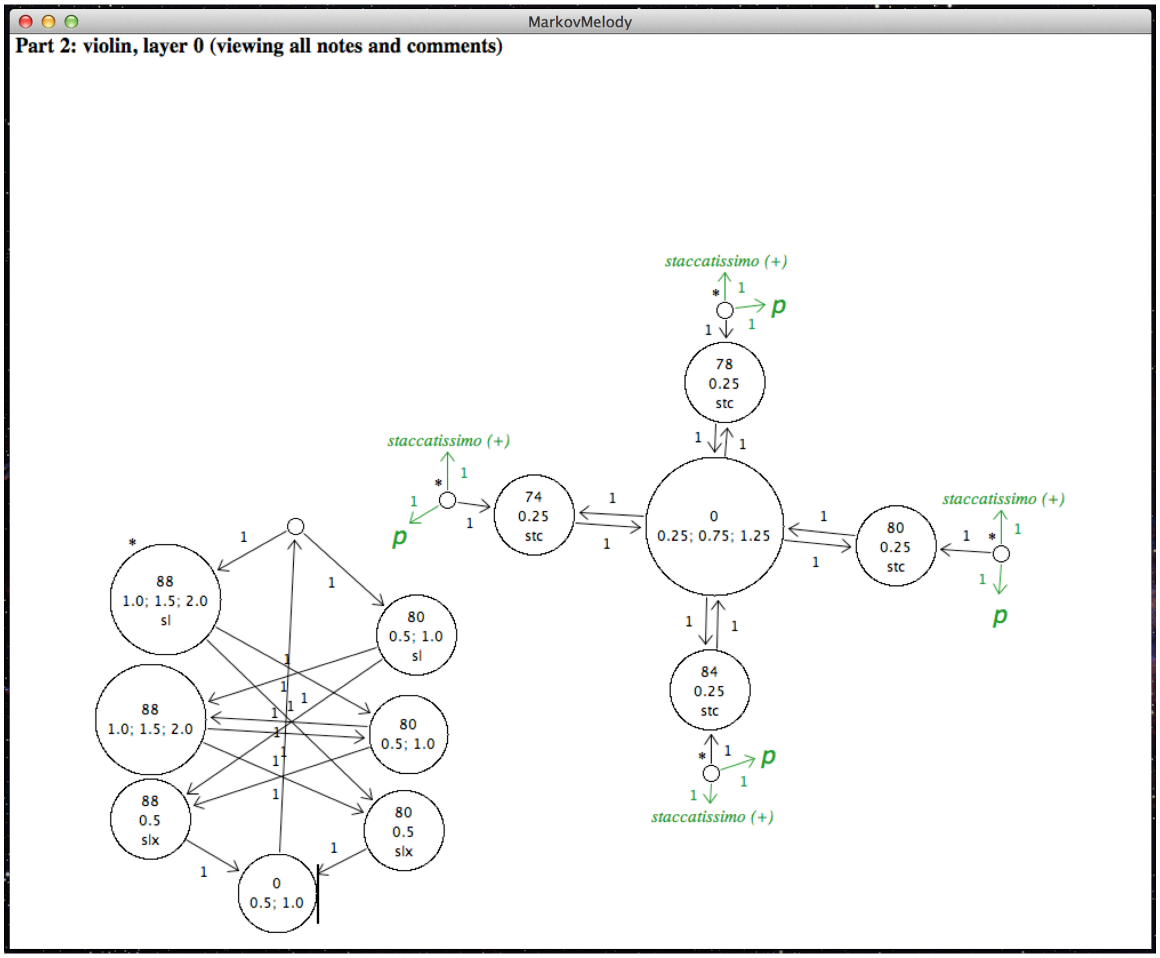
The editing tools one might reasonably expect are possible: note objects can be redefined, selected, moved around on the canvas, copied and pasted. When a group of notes is copied, whatever interior connections that group had are preserved, while exterior connections are lost.

In addition, a number of musical operations are possible, including transposition, inversion, and retrograde, the latter of which reverses the direction of all connections in the selection, with interesting results.

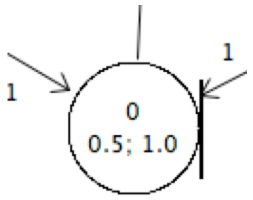
Multiple Parts

Since monophonic music has somewhat limited interest, the program is designed to allow any number of instruments or parts. If we type command-i while editing the previous meta-score, a dialog pops up asking the user to enter the name of the new instrument to add. This instrument – in our example we will choose violin – is edited on a new canvas, and the user can switch back and forth between instruments with a key command.

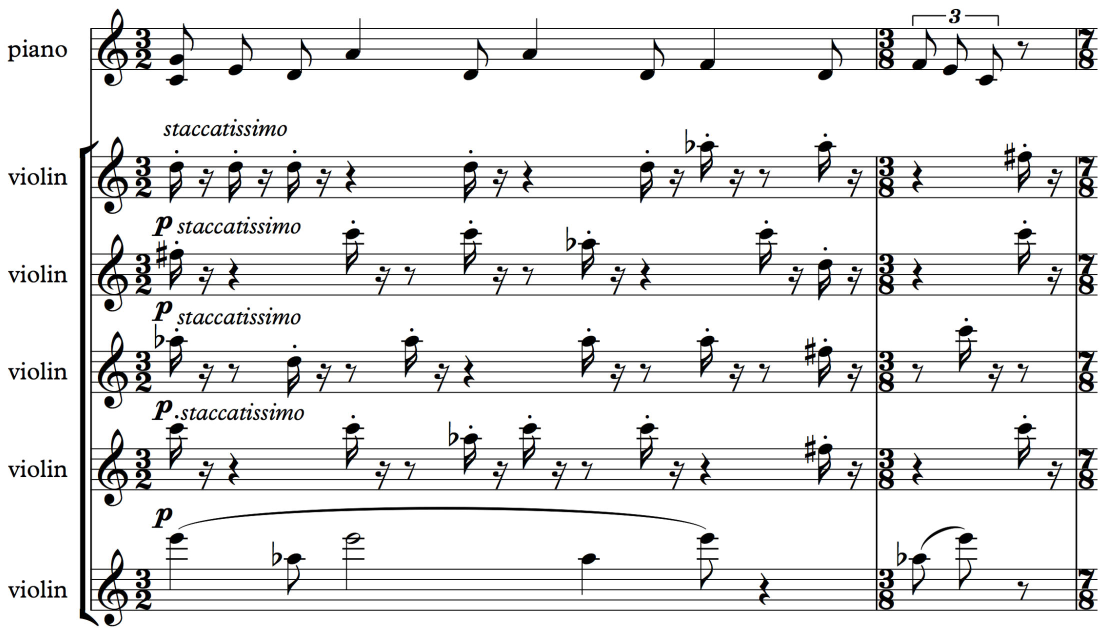
Every note with an asterisk will create a new line for the instrument in whose canvas that note resides. Thus, what we see here will add a violin section with 5 independent parts, four of which start at different points in the same network, and one of which winds its way though a separate and unconnected network:



Note that a pitch of zero indicates a rest, and that it is also possible to add slurs (the beginning indicated by "sl," the end by "slx"), articulations (in this case staccato, indicated by "stc"), and expressive markings, the latter of which can be indeterminately connected using a similar probability network to the notes[1]. Also note that expressive markings can be attached to a simple node, in which case they are transferred to the next note that occurs in the line. This is done here to avoid the marking from being repeated every time the note is reached. Finally, note that measures are created by adding bar lines to the end of note objects, as with the rest in the lower left:



Let's see what this addition has done to our example:



When the program exports, either the user can specify a maximum length, or the program can continue until all lines have terminated. In this case, a maximum length had to be entered, since the violin parts would loop indefinitely otherwise.

[1] One major difference between connections for expressive markings and for note objects is that the probabilities of connections from note objects to expressive markings are calculated independently of one another, i.e. they are not competing**.** Thus a note with a 0.5 probability connection to “p” and a 0.5 connection to “dolce” will have a 25% chance of having no marking, a 25% chance of having just “p”, a 25% chance of having just “dolce”, and a 25% chance of having both. If we *want* them to compete, that is also possible, by creating a connection first to a green marking node, and then to the two markings. Expressive markings branching off of nodes do compete.

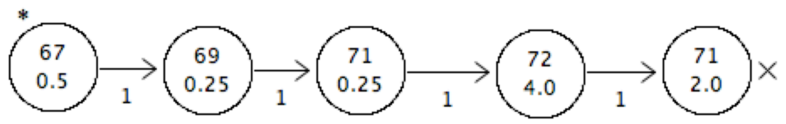
Interactions Between Lines

In the previous example, the different lines, though designed to work together, did not interact with one another. They were concurrent, but not collaborating. To allow such interaction, I have created three mechanisms that work in tandem: cues, cut-offs, and triggers.

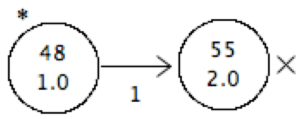
Cues and Cut-Offs

The first two of these -- cues and cut-offs -- are useful for contrapuntal coordination. Consider the following meta-score, featuring a line in the violin, and another in the cello (note that these are not shown on the same page in the editor):

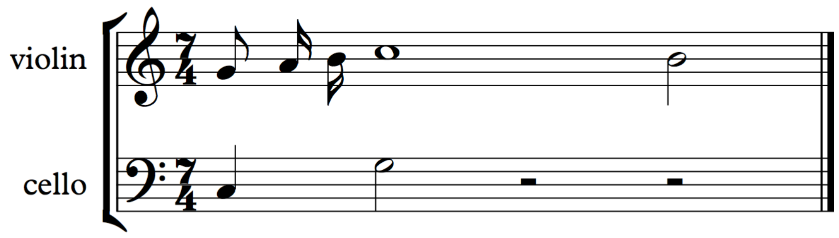
Part 1: violin, layer 0 (viewing all notes and comments)



Part 2: cello, layer 0 (viewing all notes and comments)

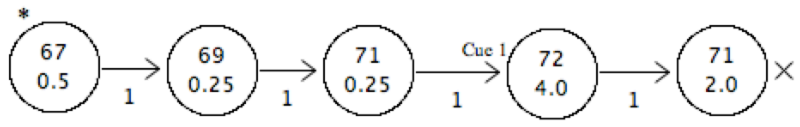


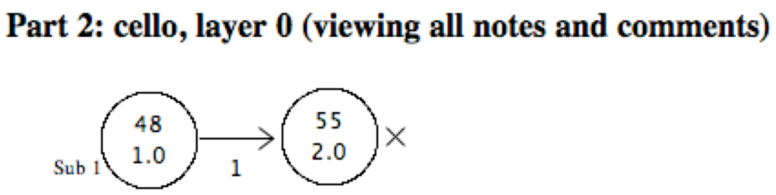
The result is two uncoordinated melodic fragments:



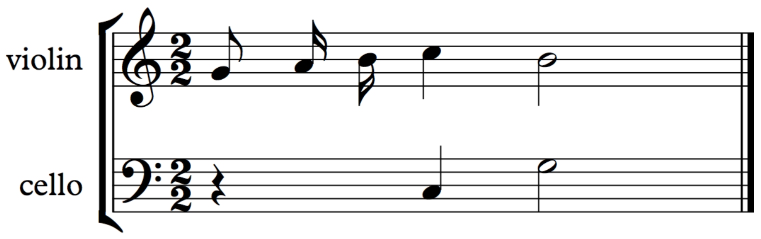
However, instead of having the cello part start at the beginning of the score, we can choose to have it cued by the fourth note in the violin part, so that it begins at the same time as that note. In the editor, this is represented by the text "cue 1" to the upper left of the cuing violin note object, and the text "sub 1" to the lower left of the cued cello note object:

Part 1: violin, layer 0 (viewing all notes and comments)

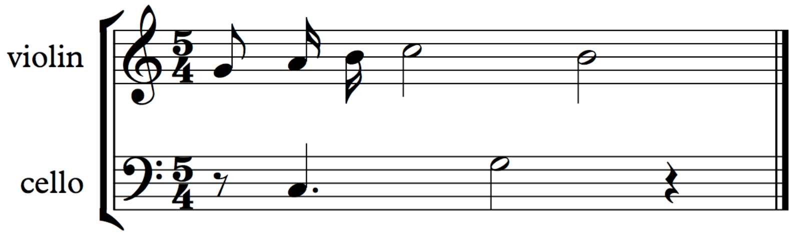




To further coordinate, I can also have the second note in the cello part cut off whatever note is in the violin when it occurs[2]. This is not represented in the main window of the editor, but is displayed in the side window when the relevant note is selected. The result is now more contrapuntally favorable:



When we create a cue or cut-off, we also have the option to time-modify it, so that it occurs before or after the note that sets it off. Here is the result if we time modify the cello cue so that it occurs a half-beat earlier than the note that cues it, and the cut-off so that it occurs a beat late:



The latter of these produces a classic suspension figure[3].

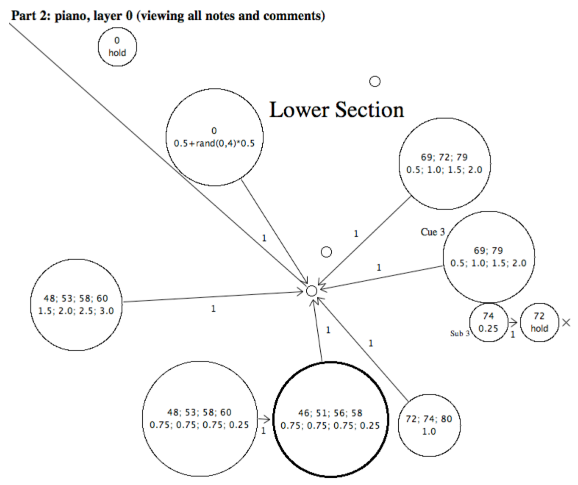
Time modified cues and cut-offs allow events in one part to occur at any time in relation to events in the other part. Note that, though this simple example was completely deterministic – and thus the parts could have been coordinated by just planning the counterpoint out in advance – in general the output of my program is not. This is why such methods of inter-part communication are crucial, because we can't plan on what a particular line will do, or how long it will take to do it.

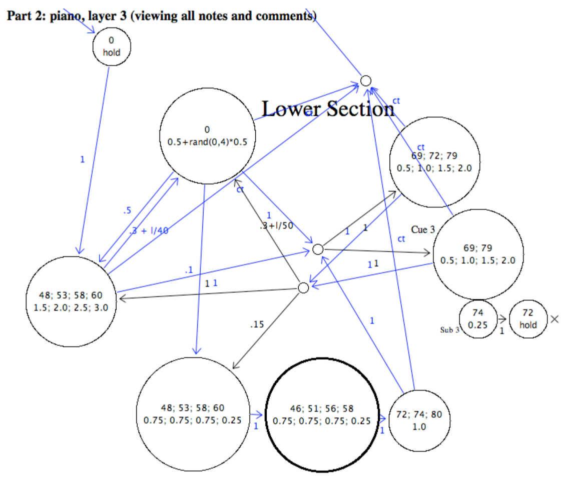
[2] It is also possible to set up a note to cut of a specific note in another part, rather all the notes in that part.

[3] Attentive readers will notice that I also had to adjust the first cello note to be a dotted-eighth-note to get the alignment I wanted.

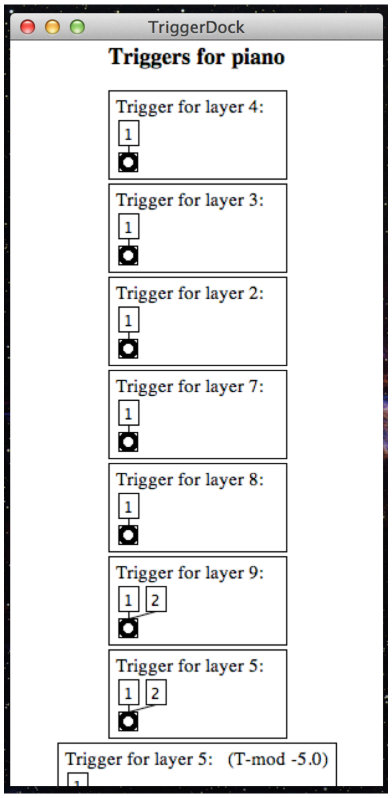
Triggers

Triggers are another important method of inter-part communication. Each part can have any number of different layers of connections, and you can see here two different layers from piano part of the meta-score for Encerrado, my work for oboe and piano:





The picture on the left shows the bottom-most layer, Layer 0, and the picture on the right shows Layer 3. By default, all new layers will start as replicas of the bottom layer. When changes are made to an upper layer, those changes are highlighted in blue for easier reading of the meta-score.


TriggerDock

Triggers for piano

Trigger for layer 4:

Trigger for layer 3:

Trigger for layer 2:

Trigger for layer 7:

Trigger for layer 8:

Trigger for layer 9:

Trigger for layer 5:

Trigger for layer 5: (T-mod -5.0)

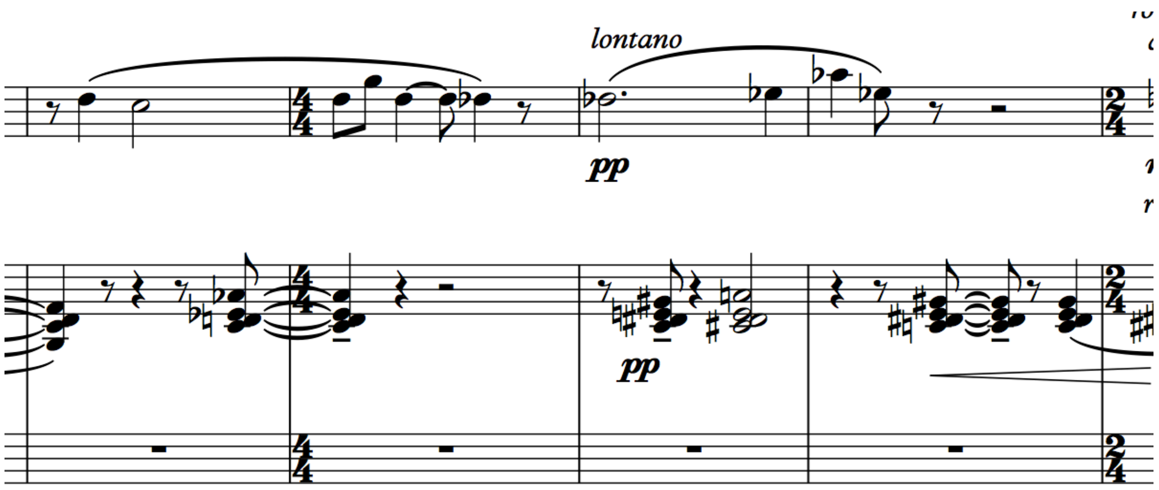
Particular sets or sequences of notes in another part can trigger a different layer to be set off, and we can view these triggers in the trigger dock window at the side.

It is possible to create triggers that are:

1) set off by any of several notes objects,
2) only set off when all of the triggering note objects have been struck, or
3) set off by a combination of these two options.

This works by the following system: each trigger has any number of targets, and note objects shoot arrows at these targets. When all the targets have been struck, the trigger goes off. If there is only one target shot at by many different note objects, then we have situation 1). If there are many targets, each with a separate note object that shoots at it, we have situation 2). Intermediate schemes will produce situation 3). It is also possible to specify that the targets must by struck in order, so that only a specific order of events will lead to the trigger going off.

Triggers allow one part to change its nature based on the events occurring in another part, and so one natural usage is in accompaniment. Below is an excerpt from my work, *Encerrado*, for Oboe and Piano:



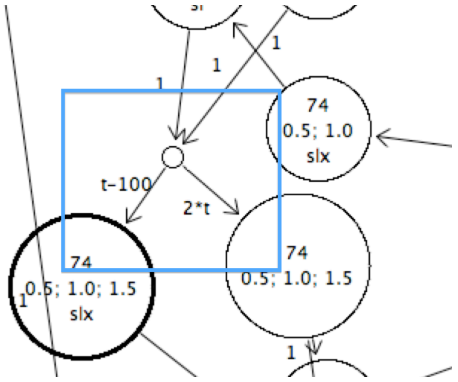
When the oboe reaches the *lontano* section, the D-flat triggers the accompanying piano part to change harmony.

Like cues and cut-offs, triggers can be time modified, so that a change in connection layer can anticipate or lag behind the developments that caused it.

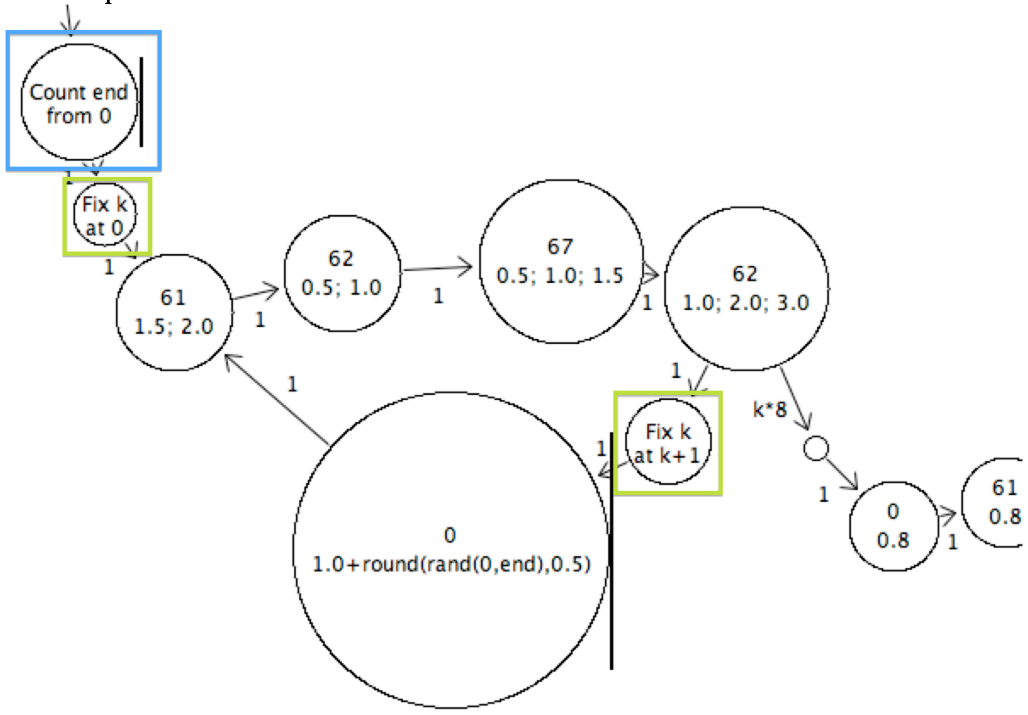
Variables

Having allowed the parts to communicate with one and other, another important element in creating a piece of music is shaping its development over time. In *Ramify*, this can be done with variables.

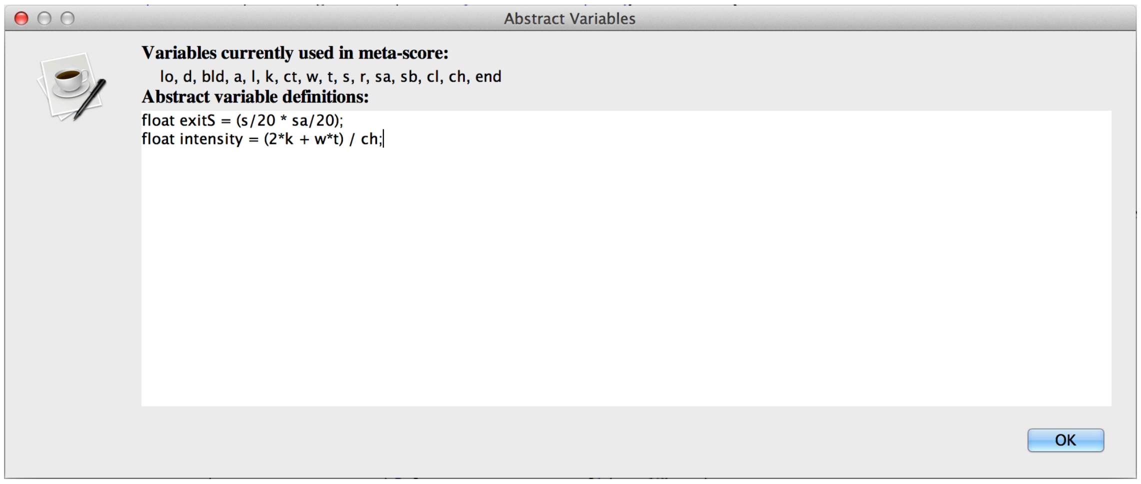
Any pitch, rhythm, or connection can be made time variable, by incorporating the parameter t, which represents the number of beats passed since the beginning of the score.



Furthermore, other variables can be defined by placing special variable nodes in the score, which can either set a fixed value (boxed in green below) for the associated variable, or start counting the beats since the node has been hit (boxed in blue). In the example below, the counter in blue is used to keep track of how much time, in beats, we have spent in this "end" section of the piece as a whole, while the variable nodes boxed in green keep track of how many times we have gone through a particular loop[4].



Abstract variables can then be defined from combinations of the basic variables in the meta-score, allowing us to follow more complex trends over the course of a piece.


Abstract Variables

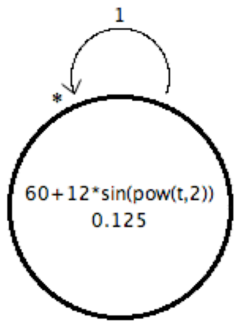
Variables currently used in meta-score:
lo, d, bld, a, l, k, ct, w, t, s, r, sa, sb, cl, ch, end
Abstract variable definitions:

```
float exitS = (s/20 * sa/20);
float intensity = (2*k + w*t) / ch;
```

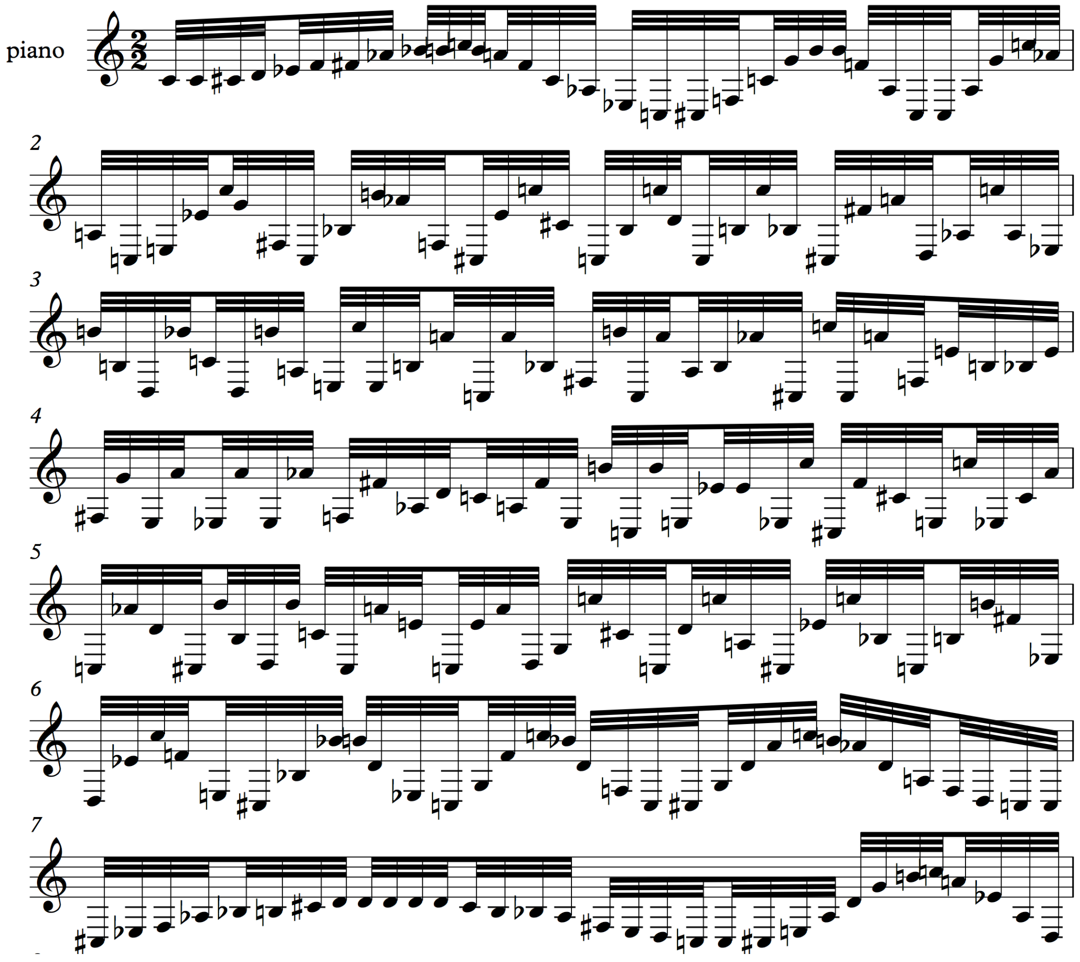
[4] They start by setting k to 0, and then incrementing it by one every time we go through the loop. Note that the probability of exiting the loop is k*8, so it gets increasingly likely to exit.

Complex Results, Simple Processes

Often, with the right generative process, apparently simple programs using variables can produce very interesting results. Here, we see a program that creates a piano line following the function sin(t^2):



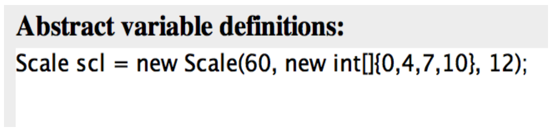
It consists only of a single note object that connects right back to itself, but the result is fascinating:



One avenue of exploration that this suggests is to investigate the musical results of different mathematical function shapes. From my preliminary experiments, it seems that sinusoids in particular form pleasing shapes; this should come as no surprise, given how fundamental and ubiquitous they are in our natural world.

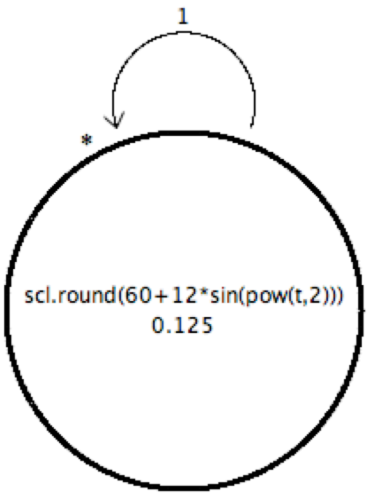
Rounding to Scales

As the previous example has already made clear, pitches, lengths, and connection probabilities can incorporate various mathematical functions. In addition, there are some special added functions, with particular musical value. For instance, it is possible to define a scale in the abstract variables window, and then to round the pitch of our interesting sinusoid to lie within that scale[5]:

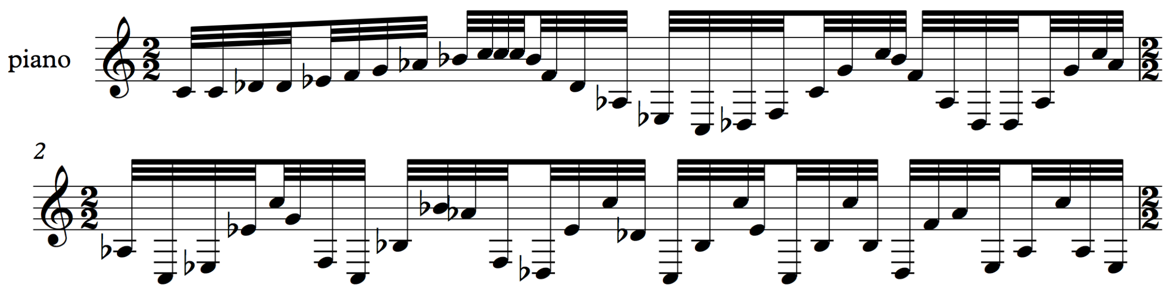


Abstract variable definitions:

```
Scale scl = new Scale(60, new int[]{0,4,7,10}, 12);
```



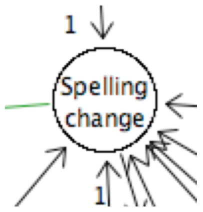
In this way, wild results can be reined into a particular desired harmonic landscape:

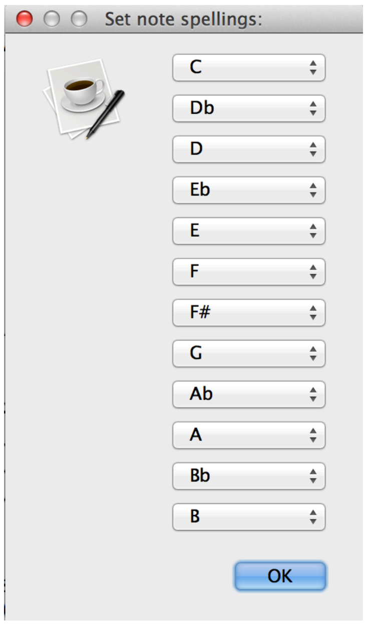


5 Note that a scale is defined by a starting note, a sequence of intervals above that starting note, and an interval at which it repeats. Thus, a C-major scale is defined by the starting note 60, the intervals {0,2,4,5,7,9,11}, and the repeat interval of 12 (since it repeats every octave). Different repeat intervals allow for the possibility of many different kinds of scales than the traditional modes.

A Brief Note to the Orthographers

For those who have been plagued this entire time by the question of how the program knows whether to write a C-sharp or a D-flat, the answer is that there are special spelling nodes in the score:





C
Db
D
Eb
E
F
F#
G
Ab
A
Bb
B

When such a node is created or edited, a dialog comes up allowing the user to assign the desired spelling to each note of the chromatic scale.

By default, the spelling is what we see in the dialog to the right. When a line passes through a spelling node, it will, from then on, spell notes as it has been told it to, until it encounters another spelling node. The advantage of this solution is that often there will be regional consistency in the spelling of notes (e.g. if we are in a certain key), and so one node can cover a whole section of music. In addition, by treating pitches as numbers and not as letters, we allow the sorts of mathematical manipulations that led to such interesting results in the previous section.

The Future of Ramify

I have only just begun to explore the possibilities of this program, and I can see much work, and play, in my future as I explore its ramifications. Though I am in no rush to expand the program, I can imagine many possible directions for it. For instance it could feature a hierarchical structure similar to pure data, in which patches are nested inside of patches. It could also feature non-note content: there's no reason why nonstandard notation, or even sound recordings could not be connected in a similar network structure. Also, mappings could be made from the two-dimensional placement of notes in the meta-score to musical parameters such as dynamics and articulation.

For now, however, there is enough to be done learning to engage with this program to make music in an intuitive and satisfying way. I hope you enjoy the musical results of these efforts as much as I have been.