Designing a highly expressive algorithmic music composition system for non-programmers

Matt Bellingham[1], Simon Holland[2] and Paul Mulholland[3]

Here we present an algorithmic composition system designed to be accessible to those with minimal programming skills and musical training, while at the same time allowing the manipulation of detailed musical structures more rapidly and more fluidly than would normally be possible for such a user group.

These requirements led us to devise evolve non-standard programming abstractions such as the chooser (panel 3) which in a single basic element encourages indeterminism, parallelism, choice, multi-choice, recursion, weighting, and looping.

The system has general musical expressivity, but for simplicity here we focus on manipulating samples.

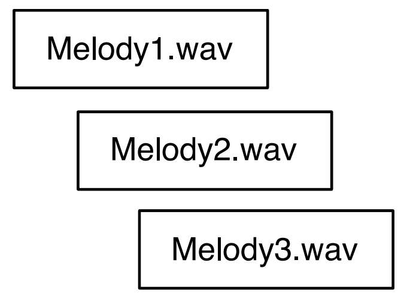
The system is designed with the following principles in mind:
- Parsimony – a small number of consistent powerful ideas do the work combinatorially;
- Musically desirable actions are simple and quick to do;
- Both bottom up and top down construction are allowed in any combination;
- Affordances for all users – children to experts – via progressive disclosure.

The system is being tested with undergraduate Music Technology students who are typically not programmers, and they are often not traditional musicians. While they may be conversant with some elements of music theory, their background is often as self-taught music producers with experience of making music electronically using music sequencers/DAWs.

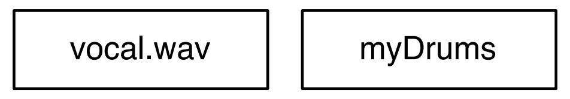
1 The play area, adding content, and single-lane choosers

The largest central area of the user interface is the play area. It is here that the user is able to drag in samples and to create choosers.

Once added, samples are shown as a rectangle containing the name of the sample. This is called a chooser. The name will become more obvious later when the choosers contain more than one sample. A single-lane chooser is the most basic type. Choosers will contain a sample name if they are added in this way.

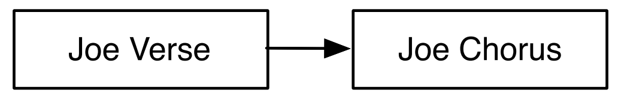


Optionally they can contain the name of another chooser.



2 Sequence

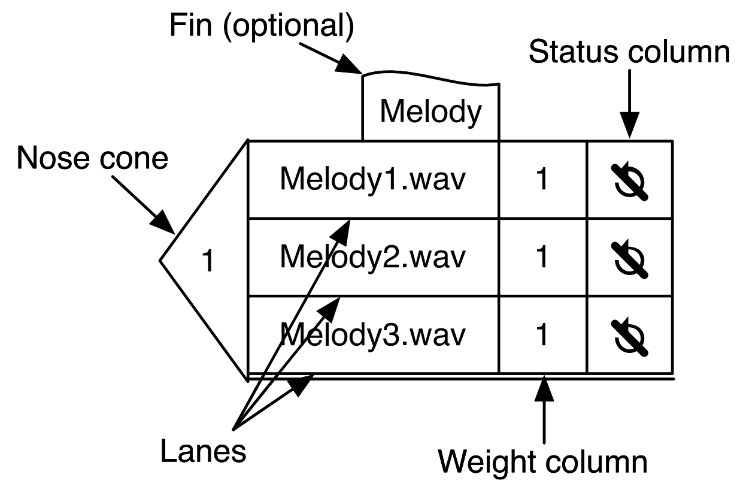
A sequence is shown by an arrow. The direction of the arrow denotes the order of the sequence. Once the first chooser has finished playing the second chooser will begin.



Note that this example uses single-lane choosers which reference choosers named 'Joe Verse' and 'Joe Chorus'. The system allows the use of such references even when a chooser of that name does not yet exist; this allows the user to create placeholders without having to immediately populate them. Importantly, this allows users to work using either a top-down or bottom-up approach. A top-down approach is when the user begins with a high-level outline of the piece (an ABA structure, for example) and then populates those sections, slowly working down to the note level. A bottom-up approach is the opposite: the user begins at the note level and builds musical sections before organising the macro structure of the piece.

3 Multi-lane choosers: combining indeterminism, choice, parallelism, multi-select weights, looping, and abstraction

Once you have more than one sample you can create a more interesting type of chooser. Samples can be snapped together vertically to create a multi-lane chooser.

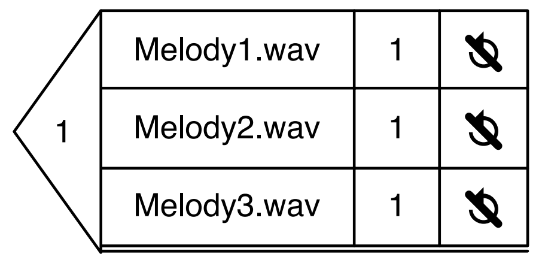


Each sample will become a lane in the chooser. Once there are two or more lanes the chooser will display three additional elements: the nose cone on the left, and the weight column and status column on the right. The nose cone allows the user to select how many of the lanes will play simultaneously. The weight column is shown immediately to the right of the sample name. This selects the weight of each lane; the likelihood of a lane to be picked. The status column is shown on the far right of the chooser, and allows for control over how the chooser deals with looping and stopping playback. Optionally, the user can name the chooser using the fin.

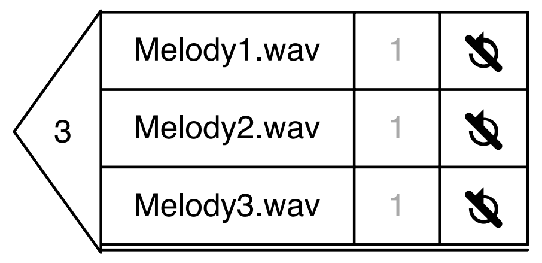
4 The nose cone: controlling indeterminacy, parallelism, and choice

The nose cone allows for indeterminacy, parallelism, and choice.

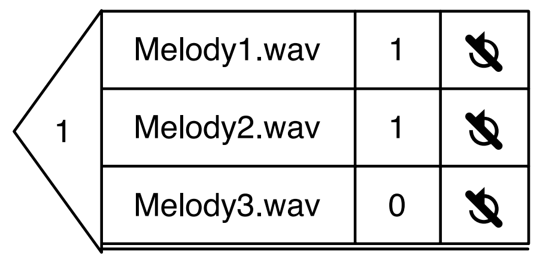
This example shows a chooser with three lanes. A nose cone number of 1 results in the software randomly selecting one of the three lanes.



If the nose cone number is 2 then two lanes will be selected and played together. If the select triangle is set to 3 then all three lanes will be played together.



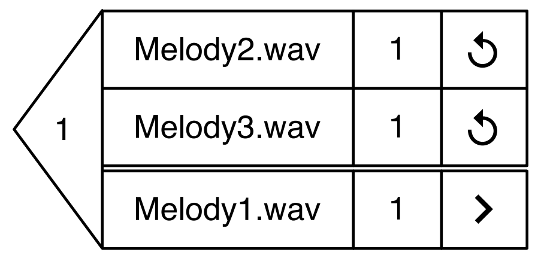
Both the nose cone and the weight column can be set to zero. If the weight of a lane is zero the lane cannot be selected, as shown in the example below.



If the nose cone is set to zero then the entire chooser is skipped.

5 Setting the duration using the length of a sample

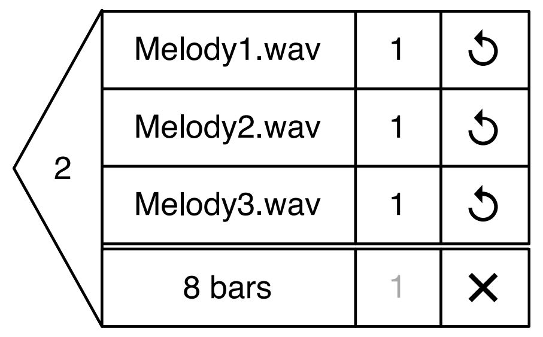
Dragging one of the lanes to below the double line at the bottom of the chooser creates a timing master lane. The duration of the timing master lane becomes the duration of the chooser. In this instance, once Melody 1 has finished playing all other lanes in the chooser will be stopped. There are two ways in which the other lanes are stopped. The user can opt for lanes to be stopped immediately (a hard stop, shown with ✕ in the status column of the timing master) or to allow lanes to play until the samples finish (a soft stop, shown with ›). This choice is made using the status column on the lane which has been chosen as the timing master.



The timing master must be chosen every time the chooser is run. For example, if the nose cone is set to 1 (as shown above) then the timing master lane will be selected each time. If the nose cone is set to 2 then the timing master lane plus one other lane will be selected, and so on.

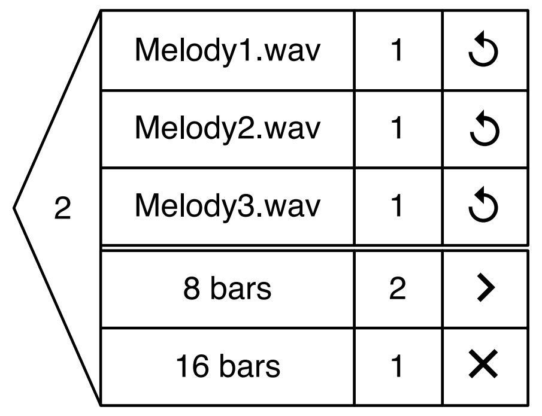
6 Setting the duration using a clock lane

If the user wants a chooser to play for a given musical duration they can add a clock lane and make that the timing master.

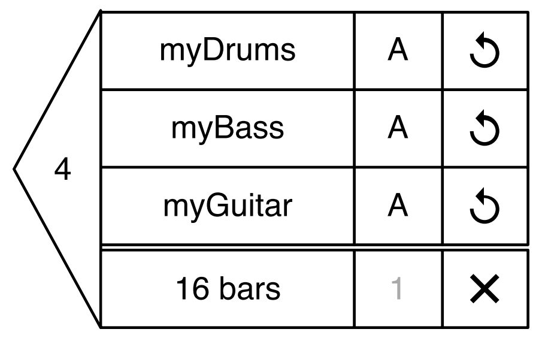


The image above shows a chooser containing three samples, and a clock lane with a duration of eight bars as the timing master. Note that the clock lane is set to a rude stop, meaning that all other lanes will be stopped as soon as 8 bars has elapsed. The lanes containing samples are set to loop if they are shorter than 8 bars duration. If they are longer than 8 bars duration they will be stopped by the timing master. The nose cone number is 2, meaning that the timing master plus one other lane will be chosen. As the timing master lane will always be chosen the weight of the lane is immaterial and is therefore greyed out.

There can be more than one timing master lane in a chooser, but only one will be selected when the chooser runs. The selection is made using the same lane weight system as for playable lanes.

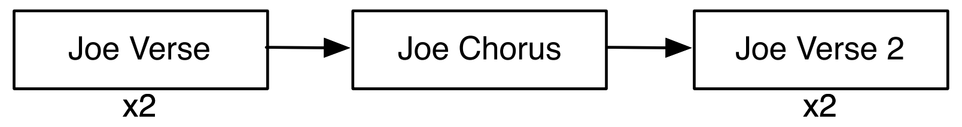


Sometimes it is desirable to ensure that a chooser lane is always selected for playback. Any playable lane with 'A' in the weight column will always be selected and played. The number in the nose cone is constrained by these mandatory lanes; for example, in the example below the nose cone number can only be 4 (play all) or 0 (skip the chooser entirely).



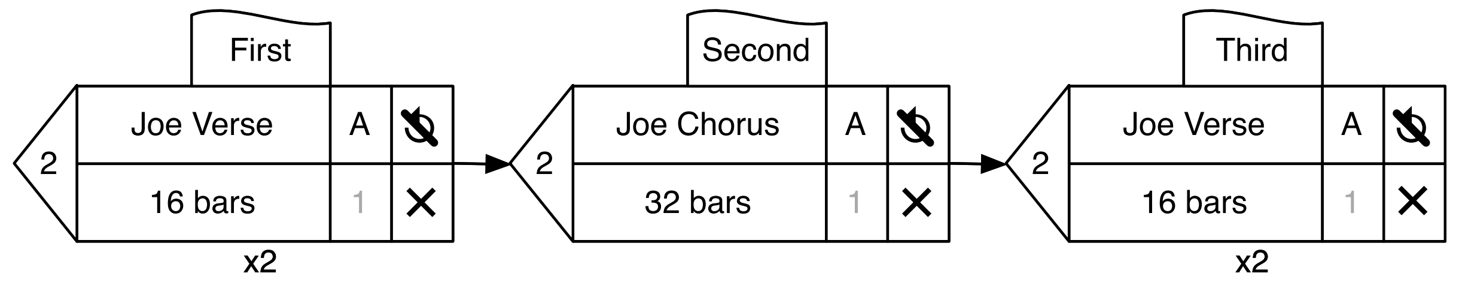
7 Repetition

Repeats can be added via multiplication notation below the chooser.



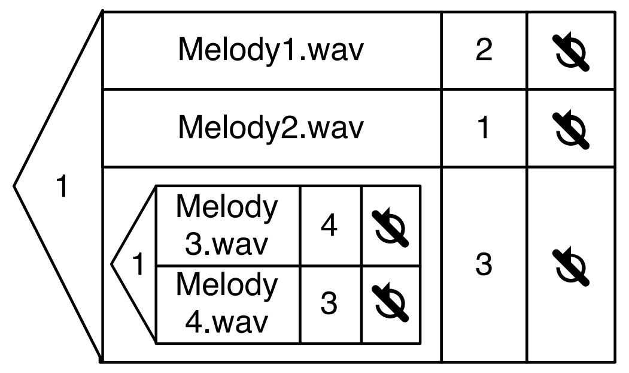
If a chooser without a timing master is repeated in this way all lane loops are defeated.

The example below shows the same arrangement as above but with durations added to each section using clock lanes as timing masters.



8 Nesting and recursion

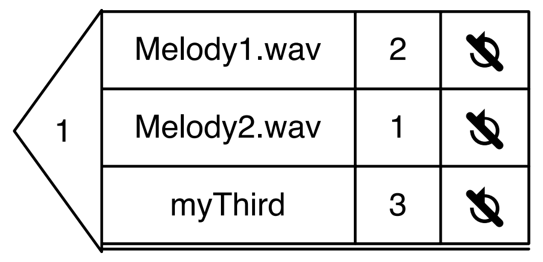
One chooser can be nested inside another chooser, either visually or by referencing a named chooser.

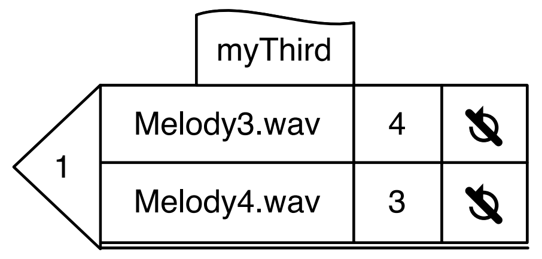


In this example, the third lane of the parent chooser hosts a nested child chooser. The result of the child chooser will be playback of either Melody 3 or Melody 4. On playback, the parent chooser will select and play back one of Melody 1, Melody 2, or the result of the child chooser.

9 Naming and referencing playable choosers

An alternative to visually nesting a chooser is to reference a chooser's name inside a lane of the parent chooser. The example below is the functionally identical to the previous example.

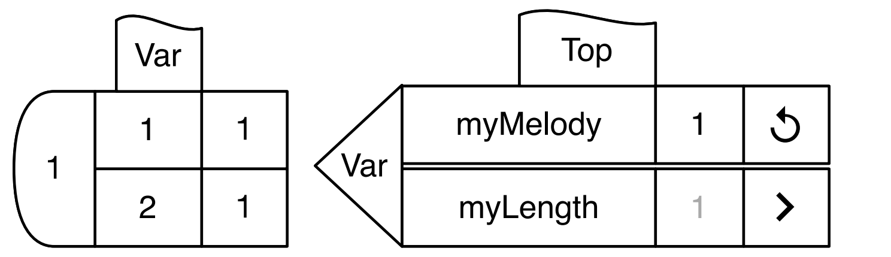




This notation can be preferable to visual nesting in some situations, and the user is free to use either notation in any combination.

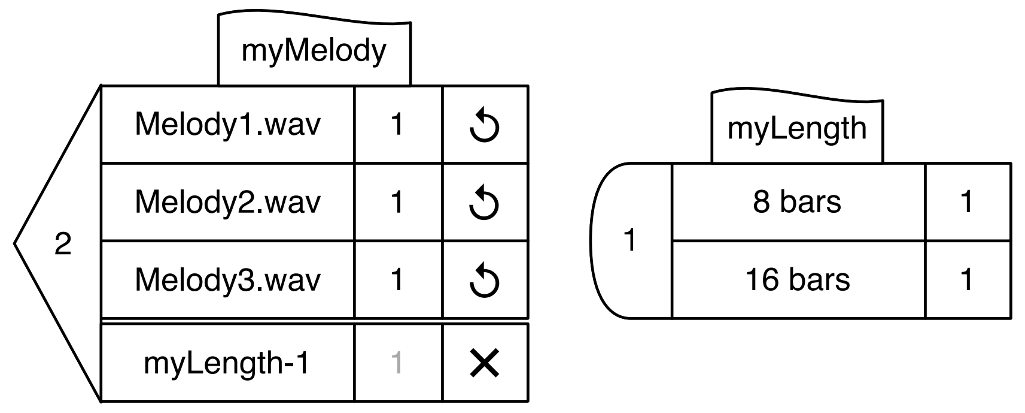
10 Variables

Variables allow the user to easily reuse material by defining once and creating multiple references to a single definition. Non-playable variables can contain integers, status types and duration information. A rounded nose cone makes them visually distinct from choosers. They are named via the fin, and do not require a status column.



The guiding principle of indeterminacy means that, by default, variables are re-run every time they are referenced.

The user is able to reuse previous values of variables by issuing a number after the variable name when it is referenced. In this way indeterminacy can be controlled. The example below shows a variable, 'myLength', which is nested inside the leftmost chooser, named 'myMelody'. The 'myLength' variable is run when it is first used in 'myMelody'. From that point, any reference to 'myLength-1' will always use the same value.



Future development

At present the design is being iteratively refined using the programming walkthrough method and implemented using SuperCollider as the back end. Initial end-user testing using a Wizard of Oz methodology is planned for early 2017.

References

Bellingham, M., Holland, S. and Mulholland, P. (2014a) A Cognitive Dimensions analysis of interaction design for algorithmic composition software, In Proceedings of Psychology of Programming Interest Group Annual Conference 2014, du Boulay, B. and Good, J. (eds.), University of Sussex, pp. 135–140, [online] Available from: http://www.sussex.ac.uk/Users/bend/ppig2014/15ppig2014_submission_10.pdf.
Bellingham, M., Holland, S. and Mulholland, P. (2014b) An analysis of algorithmic composition interaction design with reference to cognitive dimensions, The Open University, [online] Available from: http://computing-reports.open.ac.uk/2014/TR2014-04.pdf.
Green, T. R. and Petre, M. (1996) Usability Analysis of Visual Programming Environments: a `cognitive dimensions' framework, Journal of Visual Languages and Computing, 7, pp. 131–174.
Bell, B., Citrin, W. V., Lewis, C. and Rieman, J. (1992) The Programming Walkthrough: A Structured Method for Assessing the Writability of Programming Languages; CU-CS-577-92, Computer Science Technical Reports, Paper 554, [online] Available from: http://scholar.colorado.edu/csci_techreports/554.

1: Department of Music, University of Wolverhampton, UK. matt.bellingham@wlv.ac.uk
2: Music Computing Lab, Centre for Research in Computing, The Open University, UK
3: Knowledge Media Institute, Centre for Research in Computing, The Open University, UK