Toward meaningful algorithmic music-making for non-programmers

Matt Bellingham
School of Performing Arts
University of Wolverhampton
matt.bellingham@wlv.ac.uk

Simon Holland
Music Computing Lab
The Open University
s.holland@open.ac.uk

Paul Mulholland
Knowledge Media Institute
The Open University
p.mulholland@open.ac.uk

Abstract

Algorithmic composition typically involves manipulating structural elements such as indeterminism, parallelism, choice, multi-choice, recursion, weighting, sequencing, timing, and looping. There exist powerful tools for these purposes, however, many musicians who are not expert programmers find such tools inaccessible and difficult to understand and use. By analysing a representative selection of user interfaces for algorithmic composition, through the use of the Cognitive Dimensions of Notations (CDN) and other analytical tools, we identified candidate design principles, and applied these principles to create and implement a new visual formalism, programming abstraction and execution model. The resulting visual programming language, Choosers, is designed to allow ready visualisation and manipulation of structural elements of the kind involved in algorithmic music composition, while making minimal demand on programming ability. Programming walkthroughs with novice users were used iteratively to refine and validate diverse aspects of the design. Currently, workshops with musical experts and teachers are being conducted to explore the value of the language for varied pragmatic purposes by expressing, manipulating and reflecting on diverse musical examples.

Introduction

Algorithm composition can be defined as the 'partial or total automation of music composition by formal, computational means' (Fernández and Vico, 2013), and typically involves structural elements such as indeterminism, parallelism, choice, multi-choice, recursion, weighting, and looping (Baratè, 2008). At PPIG 2014 (Bellingham et al., 2014) we presented a review of existing tools, such as Max (Puckette, 1991) and SuperCollider (McCartney, 2002), for manipulating these and other elements of music. This review, using the Cognitive Dimensions of Notations framework (Green and Petre, 1996), resulted in the following findings. First, we found that most existing software requires the user to have a considerable understanding of programming constructs—represented either graphically (e.g Max, Pure Data) or textually (e.g. SuperCollider, ChucK, Csound): such constructs require a significant learning overhead. Second, in some software, users are required to have an understanding of musical notation and/or music production equipment such as mixing desks and patchbays. Third, several systems imposed working practices unconducive to compositional processes. Fourth, in some cases the user was unable to define, or subsequently change, the musical structure. Finally, complex visual design in graphical programming languages led to patches with multiple connections, making them difficult to read and navigate. These findings led to the development of a prototype visual programming language (Bellingham et al., 2017) designed to allow structural elements of the kind involved in algorithmic music composition to be readily visualised and manipulated, while making little or no demand on programming ability. This system, called Choosers, centres around a novel non-standard programming abstraction (the Chooser) which controls indeterminism, parallelism, choice, multi-choice, recursive data, weighting, and looping. We have performed two programming walkthroughs (Bellingham et al., 2018) to test the ability of self-taught music producers without programming skills to use Choosers to carry out a range of rudimentary algorithmic composition tasks; to identify usability and user experience problems in the current design; and to identify tensions and trade-offs in the interaction design of the system. These experiments were carried out using a Wizard of Oz interface supported by a fully implemented back-end written in SuperCollider. Through this empirical work we demonstrated the ability of non-programmers to work with this type of notation. We are now interested in understanding the range of musical material that can be expressed using Choosers.

This paper considers the ability of Choosers to represent musically meaningful pieces while promoting structural malleability at the surface. To enable understanding of the more musically meaningful examples that form the central part of the paper, we first present a brief overview of the principal elements of Choosers.

Choosers; annotated practical examples

Sound samples are shown in boxes, and can be auditioned by clicking on them. Samples can be assembled into sequences using arrows, shown in Figure 1. Samples in a sequence play in the order indicated by the direction of the arrows. Only a single arrow can enter or exit each element in a sequence. This deliberate limitation reflects the fact that parallelism and choice are dealt with elsewhere in the language. Boxes and sequences can be put inside other boxes, thereby packaging them into a single unit.

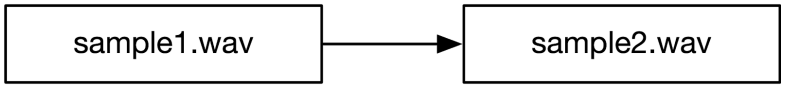

Figure 1 - Samples shown in boxes, and sequence shown using arrows.

Soundable Chooser

Boxes referring to samples or sequences can be snapped together vertically to create what are known as Choosers. Figure 2 shows a Chooser with two lanes, each containing a sample. The number in the nose cone indicates that, at run time, just one of the lanes will be selected at random (subject to the restrictions described below). On different runs, different choices may be made. By manipulating the number in the nose cone, any number of lanes from 0 to 2 can be chosen randomly to play simultaneously. A Chooser can have any number *n* of lanes. By manipulating the number in the nose cone, any number of lanes from 0 to *n* can be chosen randomly at run time and played simultaneously. Each lane has a weight associated with it; in Figure 2, changing the weight of one of the lanes to 2 would result in that lane being twice as likely to be chosen as the other. Any sample can be set to loop indefinitely when selected on a particular run, or to play just once by turning the lane's loop setting on or off. Alternatively, a finite number of loops can be specified by adding a number inside the loop icon. Indefinite looping of a single sample may not always be desired, and for this reason we now introduce Time Choosers.

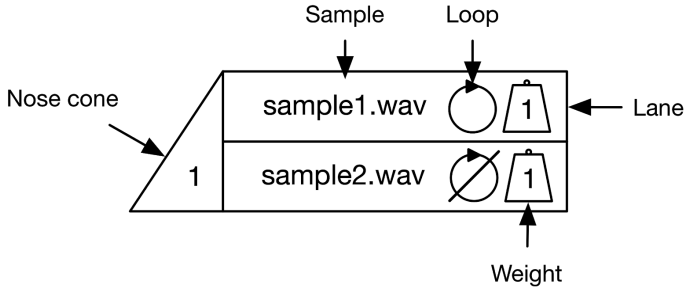

Figure 2 - A Soundable Chooser which allows control over indeterminism, parallelism, choice, multi-choice, weighting, and looping.

Time Choosers and Full Choosers

Figure 3 shows a Time Chooser, containing a duration (typically measured in bars or beats), attached to the bottom of a Soundable Chooser. This creates a Full Chooser, or just a Chooser for short. When the Full Chooser shown in Figure 3 is played, if the looping drums or bass samples are chosen on a given run, they will not play indefinitely but will be cut off after 4 bars by the Time Chooser. If the Time Chooser duration is cleanly divisible by the sample duration, every repetition of the sample will run to completion, but if the Time Chooser duration is not cleanly divisible by the sample duration

(for example, if the bass sample in Figure 3 had a duration of 3 bars), the Time Chooser would cut playback mid-sample. This is called a hard stop.

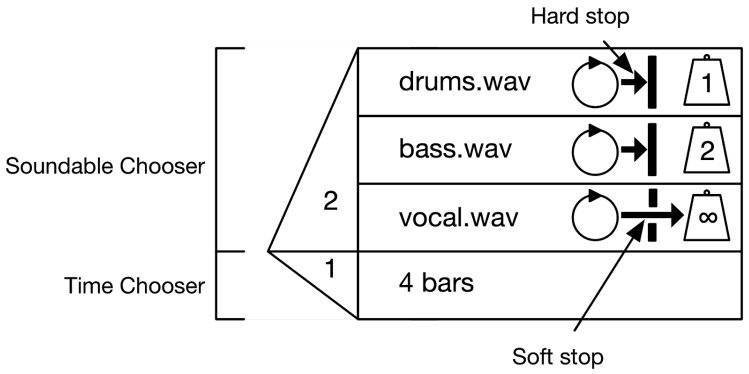

Figure 3 - A Full Chooser, consisting of a Soundable Chooser on the top and a Time Chooser on the bottom. The duration of the Soundable Chooser is moderated by the Time Chooser.

In Figure 3, the drums and bass samples are set to a hard stop, and the vocal sample is set to a soft stop. In contrast to a hard stop, when a Time Chooser's duration has elapsed, a chosen lane with a soft stop will continue to play until the end of its current loop. A Time Chooser can be used alone as part of a sequence—however, when used in this way it will simply result in a rest of the specified duration. A Time Chooser's nose cone can be set to either one or zero. If set to one, a single time lane will be chosen at run time. If it is set to zero, no time lanes will be selected and the Soundable Chooser will run as though there is no Time Chooser. This allows for quick low viscosity changes of arrangement, with the possibility of infinite playback if the Soundable Chooser lanes are set to loop. If the Soundable Chooser is not set to loop, the sample(s) will play and the Chooser will be released when they have finished playing, regardless of length.

Choosers uses 'infinity' as a maximal setting in three contexts; weight, time-lane duration, and Soundable Chooser nose cone. The effect of infinite weight, as seen in Figures 3 and 4, can be explained using a 'priority boarding' analogy. In Figure 3, the vocal sample has infinite weight and so must be the first selection, leaving one further selection to be made between the drums and bass samples. If the nose cone number is lower than the number of infinite-weighted lanes then the selection will only occur between the infinite-weighted lanes, and each lane has an equal weighting. If a lane has a weight of zero it has 'no ticket' and cannot be selected, regardless of the nose cone number. Infinite duration in a time lane is an alternative to running a Soundable Chooser with no corresponding Time Chooser. Finally, when used in a Soundable Chooser's nose cone, infinity will select all available lanes.

We will now present examples of Choosers in practice to explore the range of musical structures that can be expressed and how musically meaningful variations can be made and auditioned in range of different genres. In an attempt to avoid dancing about architecture we offer audio from the examples in this paper[1].

Example 1

Figure 4 shows a simple example in the rock genre. The sequence shown in Figure 4 is populated by the named Choosers shown underneath them; note that Choosers may be named via a fin at the top of each Chooser, with these names used to refer to them elsewhere in the sequence. Thus, the sequence shown at the top of Figure 4 sounds the same as a sequence created by drawing a line directly between the two Choosers. Note that a named Chooser can be referenced from multiple locations in a visual program, meaning that changes to the named Chooser will be reflected across all referents.

[1] https://figshare.com/s/d2edaf6486812ff1f596

Multiplication notation (e.g. ‘x2’, as seen in the sequence at the top of Figure 4) is very commonly used and understood by musicians (e.g. lead sheets, chord charts) and it allows for low-viscosity auditioning.

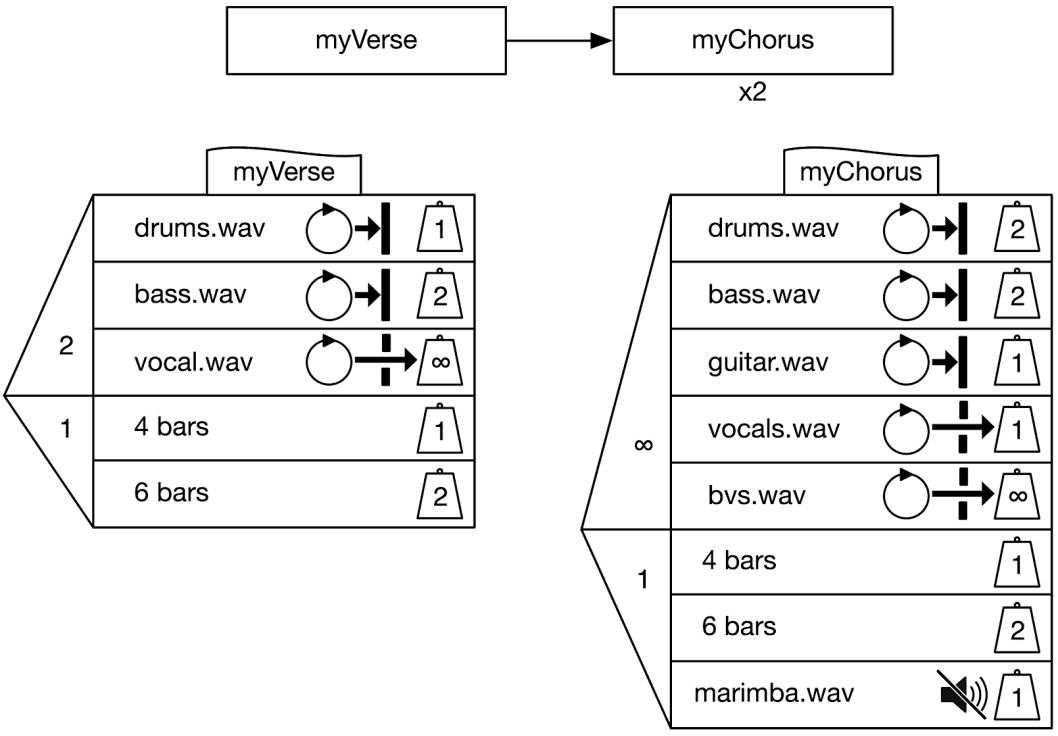

Figure 4 - An example showing sequence, nesting, naming via a fin at the top of the Choosers, and nondeterministic duration.

In a classic paper on the nature of musical creativity and algorithmic music composition, Bruce Jacob (1996) considers the ‘hard work’ of iteratively trying various options before making a final choice. A goal of our system is to support the user during this ‘hard work’ phase to make quick, impactful, and non-destructive changes, and to audition the results. By limiting options, surfacing key parameters, and maximising combinatorial usage, the possible interactions between the settings of Soundable and Time Choosers can make the results more varied than might be imagined. In precision-timed rock examples, the different effects of hard and soft stops provide a wide range of musical results. For example, consider the Chooser named ‘myVerse’ in Figure 4; the Soundable Chooser nose cone is currently set to 2, meaning that two of the three lanes will be selected when it is run. If the number were set to 0 then no lanes would be selected, but as the Time Chooser’s nose cone is set to 1 then a duration will be selected and the Chooser will run silently for that duration. If the Soundable Chooser nose cone is set to 2 and the Time Chooser nose cone is set to 0 then no duration will be selected, meaning that the Soundable Chooser will run without being stopped prematurely. If both nose cones are set to 0 then the Chooser will be skipped. Note that the Time Chooser nose cone can only be set to either 0 or 1; multiple durations cannot be selected. A soundable file can be used in a Time Chooser lane and, if selected, the sample’s length will become the duration used by the Time Chooser. An example of this can be seen on the lower right of Figure 4; a marimba sample is used in a lane in the Time Chooser. The sample can be used for duration only, or for both duration control and playback, by setting the sample to play or mute via the speaker icon.

Example 2

When there is one or more levels of nesting then considerably more complex structures and meaningful modifications become possible. The next example, shown in Figure 5, shows a piece of music in which the Chooser named ‘top’ refers to the two instruments (piano and violin) which are, in turn, populated by named Choosers. The piano is split into ‘lefthand’ and ‘righthand’ Choosers, which

each select one of two possible samples. Note that the ‘lefthand’ and ‘righthand’ child Choosers are set to not loop, while the ‘piano’ Chooser is set to loop. If we focus on ‘lefthand’, we can see that the ‘piano’ Chooser will trigger the choice of one of two samples for playback. Once complete, the ‘lefthand’ Chooser will be triggered again due to the loop setting of the ‘piano’ Chooser, with the potential for a different sample choice. An alternative would be for the ‘lefthand’ Chooser’s lanes to be set to loop; this would result in the choice of one sample which would then loop continuously.

The Choosers shown in Figure 5 can be manipulated to yield significant musical changes, such as:

- Changing the nose cone of ‘top’ or ‘piano’ to 1 will result in the selection of just one lane. Alternatively, setting the weight of a lane to 0 will remove it from the selection.
- The ‘...hand’ Choosers can be set to loop, and the corresponding lanes in the ‘piano’ Chooser set to not loop. This will result in the selection and looping of one sample, rather than continual reselection. Additionally, the nose cone values of the ‘...hand’ Choosers can be set to 0 (no selection) or 2 or infinity (playback of both samples simultaneously). The same techniques can be used with the ‘violin’ Chooser.

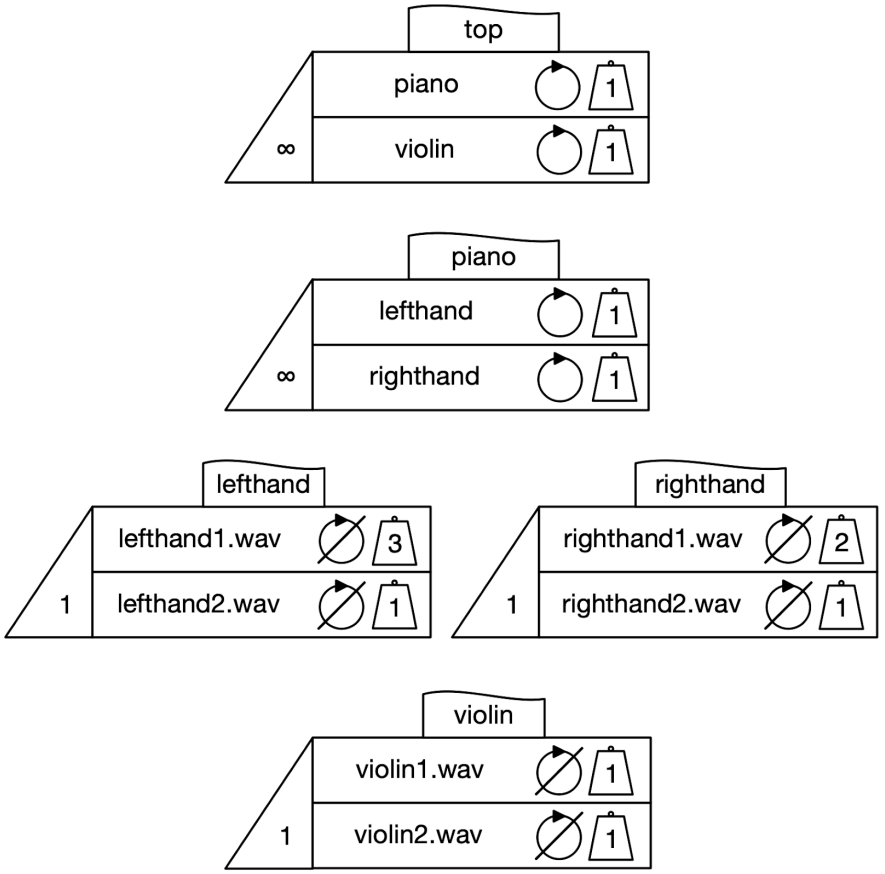

Figure 5 - Piano and violin example. The ‘lefthand’ pattern is three beats long and the ‘righthand’ pattern is four beats long, meaning that they will align every twelve beats. The ‘violin’ sample is eight beats long.

A central aspect of music is timing. By adding Time Choosers to the nested structure shown in Figure 5 we are able to manipulate time at different levels; an example is shown in Figure 6. Here we see that the ‘lefthand’, ‘righthand’, and ‘violin’ Choosers have been converted into Full Choosers by the addition of Time Choosers, with the durations mirroring the duration of the samples. Note that the ‘violin’ Chooser now has an added blank lane in the Soundable Chooser with a weight of 2; if selected a blank lane will ‘play’ silently for the duration set by the Time Chooser, which in this case will introduce a rest of 8 bars.

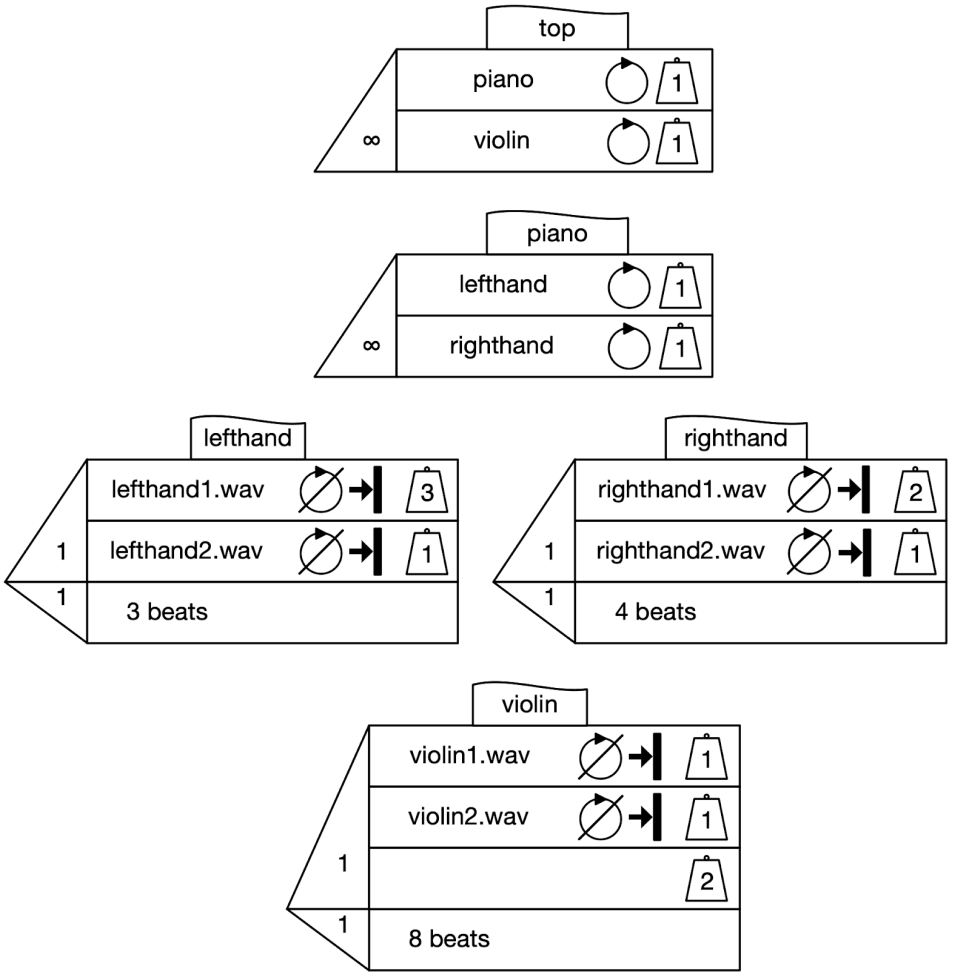

Figure 6 - The piano and violin example shown in Figure 5 with additional Time Choosers added to moderate duration. Note the use of a blank lane in the violin Chooser; if selected this lane will result in a rest (silence) for the duration of the Chooser.

As shown, Figure 6 builds on Figure 5 by adding duration controls, meaning that the user can now change these durations to make structural changes to the music. Examples of these changes include:

- Changing the 'violin' Time Chooser's duration to 10 beats; if one of the eight-beat-long samples is selected this will introduce a two-beat rest before the next selection, and if the blank lane is selected there will be a 10 beat rest;
- Changing the duration of 'lefthand' to 4 beats; this will introduce a 1-beat rest at the end of each sample, and will align the duration with 'righthand'. A duration of 5 beats will introduce a 2-beat rest, and so on;
- More interestingly, changing one of the '...hand' Choosers to a fractional value can create phase music effects. For example, changing 'lefthand' to 3.5 beats creates an alternating syncopated/synchronised pattern. A duration of 3.05 beats creates a slower, tape-phase-style shift.

Example 3

The final example, shown in Figure 7, includes two doubly-nested Choosers. In order to clearly show the nesting hierarchy we present the Choosers in columns to be read from left (parent) to right (children), but the design of Choosers allows users to freely place objects as they wish. The example in Figure 7 shows beat-level durations in the child Choosers of the 'perc' Chooser (upper row).

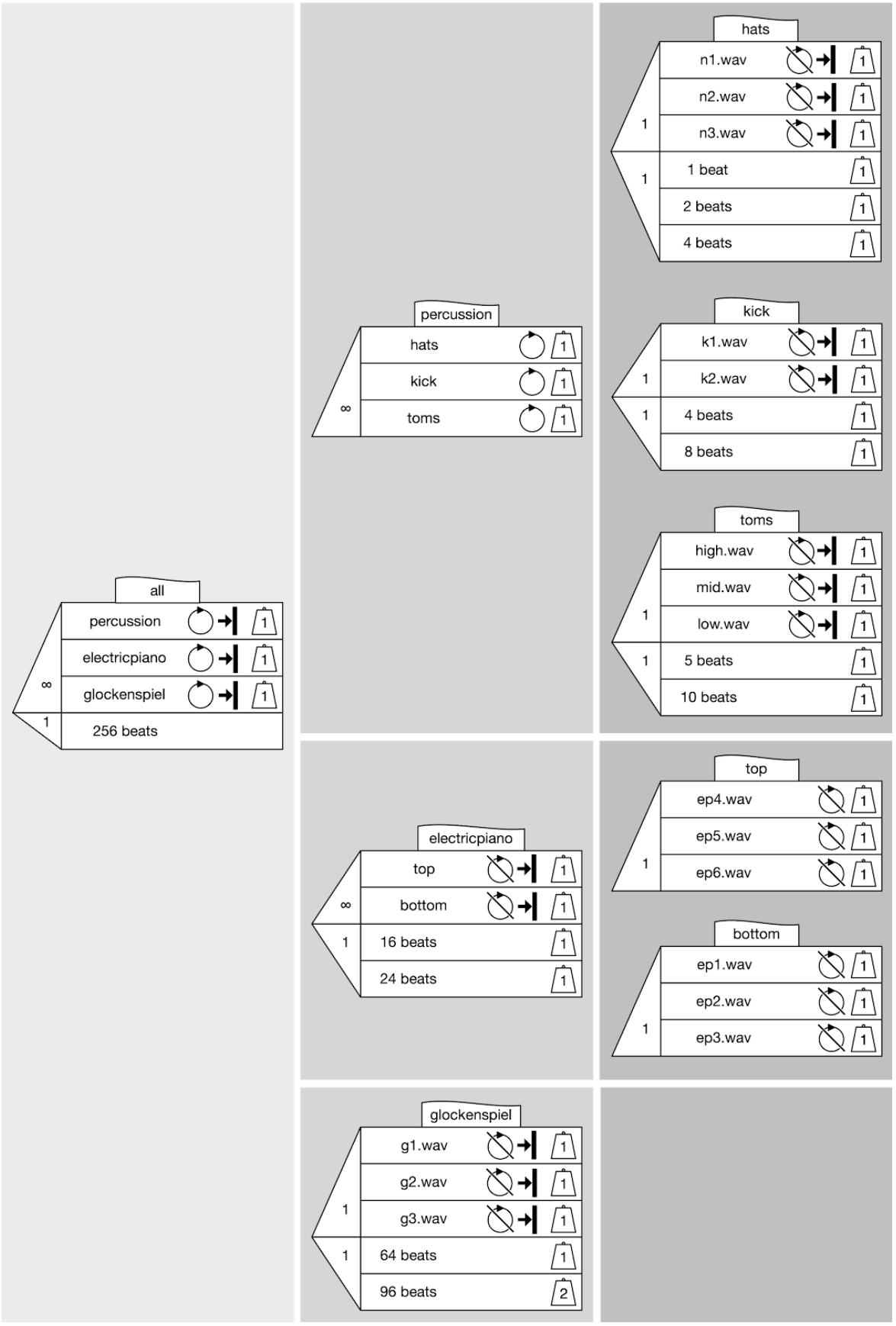

Figure 7 - An example showing doubly-nested Choosers. The nesting hierarchy has been visualised in columns and rows for the reader's benefit; in fact, Choosers and sequences may be laid out freely.

Consider the 'hats' Chooser shown in the upper right of Figure 7; when run it will select and play one of three samples without looping and, after a duration of either 1 or 2 beats, will be triggered again by the looping lane in its parent Chooser (‘perc’). This, in combination with the ‘kick’ and ‘toms’ Choosers, forms the rhythmic pattern for the piece. The ‘electricpiano’ Chooser contains two nested Choosers, each selecting one of three electric piano samples; the 'electricpiano' Chooser will trigger new selections every 8 or 12 beats. Finally, the ‘glock’ Chooser is a singly-nested Chooser which will play one of three glockenspiel samples every 24 or 48 beats. The parent Chooser, ‘all’, has a total duration of 64 beats, after which a hard stop message will immediately stop all lanes. The structure of the piece can be changed as we have seen in the previous examples:

- Durations for each Full Chooser can be changed via Time Chooser lanes; for example, the 'hats' Chooser's durations can be shortened to create a denser, more rapidly-changing pattern. Note that the duration control for the electric piano part is in the 'electricpiano' Chooser, which triggers a selection from child Soundable Choosers (‘top’ and ‘bottom’) when it is triggered by the ‘all’ Chooser. Compare this to the ‘perc’ Chooser, in which the child Choosers ('hats', ‘kick’, and ‘toms’) control the duration of each separate musical part. Each child Chooser makes a selection of both sample and duration, plays the selected sample for the selected duration, and is then triggered again by the ‘perc’ Chooser's looping lanes.
- Nondeterministic choices can be controlled via Chooser nose cones and weights. Lanes can be given a zero weight to remove them from selection, and infinite weight can be added to ensure the selection of one or more lanes given a sufficient nose cone value. For example, consider the changes that could be applied to the ‘all’ Chooser on the far left of Figure 7; the Soundable Chooser nose cone value could be reduced to 2, meaning that two of the three lanes will be selected. The user could change the weight of ‘perc’ to infinity, resulting in the selection of ‘perc’ plus one other lane. If desired, the weight of ‘glock’ could be changed to 0 to remove it from selection.
- Let us now consider a specific change to the drum pattern in the upper right of Figure 7. If the user wanted to remove ‘toms’ from selection they could do so in one of four ways; the ‘toms’ lane in the 'perc' Chooser could be given a weight of zero; the Soundable Chooser nose cone of the ‘toms’ Chooser could be given a weight of zero (resulting in a rest of either 5 or 10 beats if the Time Chooser is not also set to zero; if both nose cones are set to zero the Chooser will be skipped completely); or the weights of all three lanes in the ‘toms’ Chooser could be set to zero.
- If the user wanted one glockenspiel sample to be selected and played repeatedly, rather than a new selection each time, the ‘glock’ lane in the ‘all’ Chooser could be set to not loop, and the lanes in the ‘glock’ Chooser set to loop.
- The ‘all’ Chooser has a Time Chooser with a duration of 64 beats, with all Soundable Chooser lanes set to a hard stop. This sets the total length of the piece. If the user wanted infinite playback they could change the Time Chooser nose cone to zero; set the duration to infinity; or remove the Time Chooser from ‘all’.

Work-in-progress - control panel, variables

Choosers were designed to be ‘tweakable’; they surface musically useful tools which allow for meaningful structural changes to be performed while minimising both viscosity and premature commitment. One musically useful example is to allow the user to change the length of a sample, and the start point for playback, via a control panel that can be opened for any sample in the system. A sketch of the control panel is shown in Figure 8.

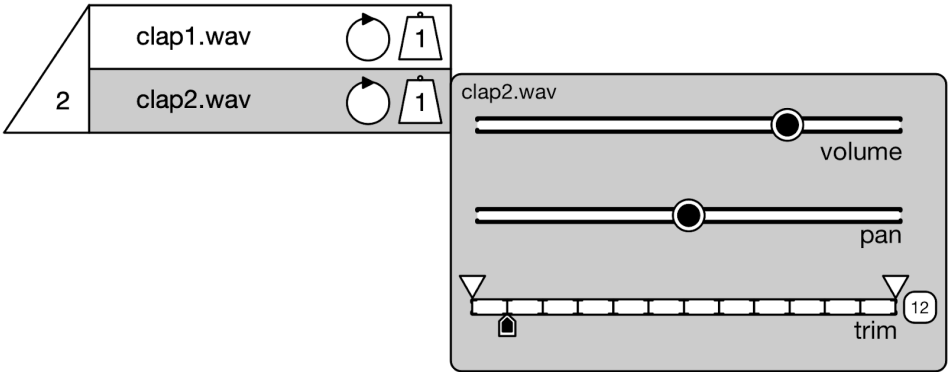

Figure 8 - A control panel to access trim (downwards arrows) and anchor point (upwards arrow).

The control panel allows the user to make a quick manual adjustment before re-running the system to audition the changes. The panel contains standard playback controls such as volume and pan. Alongside these the user can make use of a trim control, combining control over the length of the sample via 'top' and 'tail' controls (downwards-pointing arrows on the top of the horizontal line) and the starting playback position of the sample shown via an anchor control (the upwards-pointing arrow underneath the horizontal line). The user can set an optional grid, set to '12' in Figure 8, which introduces a visual grid and to which the trim and anchor controls will conform. The grid can be disabled by clicking on the number, allowing freehand control of the trim and anchor settings. One notable use of this system is *Clapping Music* by Steve Reich, which can be performed making use of the Chooser in Figure 8.

An extended version of the system, beyond the scope of this paper, puts this and many other features under programmable control by end users. This extension allows variables to be assigned to numbers, samples, sequences, or choosers, and allows such variables to be used in diagrams wherever those elements might appear (e.g. trim and anchor points, nose cone values, weights, durations, and choices of sample). An interesting feature of these extensions is that simple textual expressions that alter the value of variables may appear in Soundable Chooser lanes. Such expressions are executed when the corresponding Soundable Chooser lane is selected and played, and are subject to the same nondeterministic and time-structuring controls as the soundable elements. In this way, a rich variety of nondeterministic and highly structured outcomes can be manipulated.

Conclusions

In earlier research (Bellingham et al, 2014, 2017, 2018) we identified and then implemented a number of design principles to allow non-programmers access to algorithmic composition tools. Specifically, we sought to leverage parsimony in order to enhance learnability; to surface musically meaningful actions, and to make them quick and easy; to allow both bottom-up and top-down construction; and to make use of progressive disclosure to allow for advanced use without harming usability for beginners. In this paper we have illustrated these principles in the context of exploring the range of non-deterministic algorithmic musical expression facilitated, the kinds of structural manipulations supported, and the extent to which structural malleability has been brought to the surface.

The design of Choosers allows for user preference where possible; for example, it allows for both top-down and bottom-up construction and low-viscosity changes in structure, leading to the simple graphical sequencing design shown in Figures 1 and 4. Figure 4 shows a sequence which is then populated via the named Choosers, which is an alternative to drawing a sequence arrow directly between the two Choosers. The latter would enhance the closeness of mapping; reduce hard mental operations; increase role expressivity; and reduce hidden dependencies. However, the version shown in Figure 4 presents a different set of tradeoffs; it is better suited to more complex Choosers and arrangements; viscosity is reduced and visibility enhanced; the closeness of mapping to the musical

arrangement is improved; and both juxtaposability and hard mental operations are improved when used in complex examples.

Choosers is capable of making complex music, with no limit in principle to the length of a piece, time resolution, or depth of nesting. The next phase of the project will be to test Choosers with expert users and educators, and we welcome feedback on the design thus far.

References

Baratè, A. (2008) *Music Description and Processing: An Approach Based on Petri Nets and XML*, INTECH Open Access Publisher [Online]. Available at http://www.researchgate.net/profile/Adriano_Barate/publication/221786820_Music_Description_and_Processing_An_Approach_Based_on_Petri_Nets_and_XML/links/004635268b80f75f6b000000.pdf.

Bellingham, M., Holland, S. and Mulholland, P. (2014) 'A Cognitive Dimensions analysis of interaction design for algorithmic composition software', Boulay, B. du and Good, J. (eds), *Proceedings of Psychology of Programming Interest Group Annual Conference 2014*, University of Sussex, pp. 135–140 [Online]. Available at http://www.sussex.ac.uk/Users/bend/ppig2014/15ppig2014_submission_10.pdf.

Bellingham, M., Holland, S. and Mulholland, P. (2017) 'Choosers: designing a highly expressive algorithmic music composition system for non-programmers', *2nd Conference on Computer Simulation of Musical Creativity* [Online]. Available at http://hdl.handle.net/2436/621151.

Bellingham, M., Holland, S. and Mulholland, P. (2018) 'Choosers: The design and evaluation of a visual algorithmic music composition language for non-programmers', Church, L. and Basman, A. (eds), *Proceedings of the 29th Annual Workshop of the Psychology of Programming Interest Group - PPIG 2018* [Online]. Available at http://www.ppig.org/sites/ppig.org/files/PPIG-2018-proceedings.pdf.

Bullock, J., Beattie, D. and Turner, J. (2011) 'Integra Live: a new graphical user interface for live electronic music', *International Conference on New Interfaces for Musical Expression* [Online]. Available at http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.449.533&rep=rep1&type=pdf.

Fernández, J. D. and Vico, F. (2013) 'AI methods in algorithmic composition: A comprehensive survey', *Journal of Artificial Intelligence Research*, pp. 513–582 [Online]. Available at http://www.jair.org/papers/paper3908.html.

Green, T. R. and Petre, M. (1996) 'Usability Analysis of Visual Programming Environments: a 'Cognitive Dimensions' Framework', *Journal of Visual Languages and Computing*, vol. 7, pp. 131–174.

Jacob, B. L. (1996) 'Algorithmic Composition as a model of creativity', *Organised Sound*, Cambridge University Press, vol. 1, no. 3, pp. 157–165.

McCartney, J. (2002) 'Rethinking the Computer Music Language: SuperCollider', *Computer Music Journal*, vol. 26, no. 4, pp. 61–68 [Online]. Available at http://dx.doi.org/10.1162/014892602320991383.

Puckette, M. (1991) 'Combining Event and Signal Processing in the MAX Graphical Programming Environment', *Computer Music Journal*, The MIT Press, vol. 15, no. 3, pp. 68–77 [Online]. Available at http://www.jstor.org/stable/3680767.

Wilson, S., Cottle, D. and Collins, N. (2011) *The SuperCollider Book*, MIT Press.