Choosers: The design and evaluation of a visual algorithmic music composition language for non-programmers

Matt Bellingham
matt.bellingham@wlv.ac.uk

Outline

Aims of the project

Review of existing software

Principles

Key graphical abstractions and examples of notation

User test methodology

Reflection on design issues

Candidate solutions

Introduction and motivation

- An algorithmic composition system;
  - The partial or total automation of music composition by formal, computational means (Fernández and Vico, 2013);
  - Non-deterministic – multiple outcomes can result from the same algorithm.
- Designed to be accessible to those with minimal programming skills and little musical training;
- Allows the manipulation of detailed musical structures;
- Permits indeterminism, parallelism, choice, multi-choice, recursion, weighting, and looping;
- Implemented as a set of SuperCollider classes to enable end-user testing.

Review of existing software

- Algorithmic Composition Toolbox (Berg, 2012)
- AthenaCL (Ariza, 2011)
- ChucK (Wang and Cook, 2013)
- Csound (Vercoe, 2014)
- Cylob Music System (Jeffs, 2010)
- Fractal Tune Smithy (Walker, 2011)
- FractMusic (Diaz-Jerez, 2012)
- Harmony Improvisator (Synleor, 2013)
- Impro-Visor (Keller, 2012)
- Impromptu (Sorensen, 2010)
- Improviser for Audiocubes (Percussa, 2013)
- Lexikon-Sonate (Essl, 2010)
- Maestro Genesis (Szerlip and Hoover, 2012)
- Max (Cycling ’74, 2014)
- Mixtikl (Intermorphic, 2013)
- Musical Algorithms (Middleton, 2004)
- MusiNum (Kindermann, 2006)
- Noatikl (Intermorphic, 2011)
- Pure Data (Puckette, 2011)
- SoundHelix (Schürger, 2012)
- Strasheela (Anders, 2012)
- SuperCollider (McCartney, 2014)
- Usine (Sens, 2013)
- Wolfram Tones (Wolfram Research Labs, 2011)

Review of existing software

In a previous paper (Bellingham et al. 2014) we reviewed a representative selection of software capable of algorithmic music composition. The key findings were:

- Most software requires understanding of complex constructs in either graphical or text-oriented programming languages;
- Users are often required to have an understanding of musical notation and/or music production equipment;
- Several pieces of software imposed working practices not conducive to the variety of compositional processes;
- Complex visual design in graphical programming languages led to code which was difficult to read and navigate.

Principles

Parsimony — a small number of consistent powerful ideas do the work combinatorially;

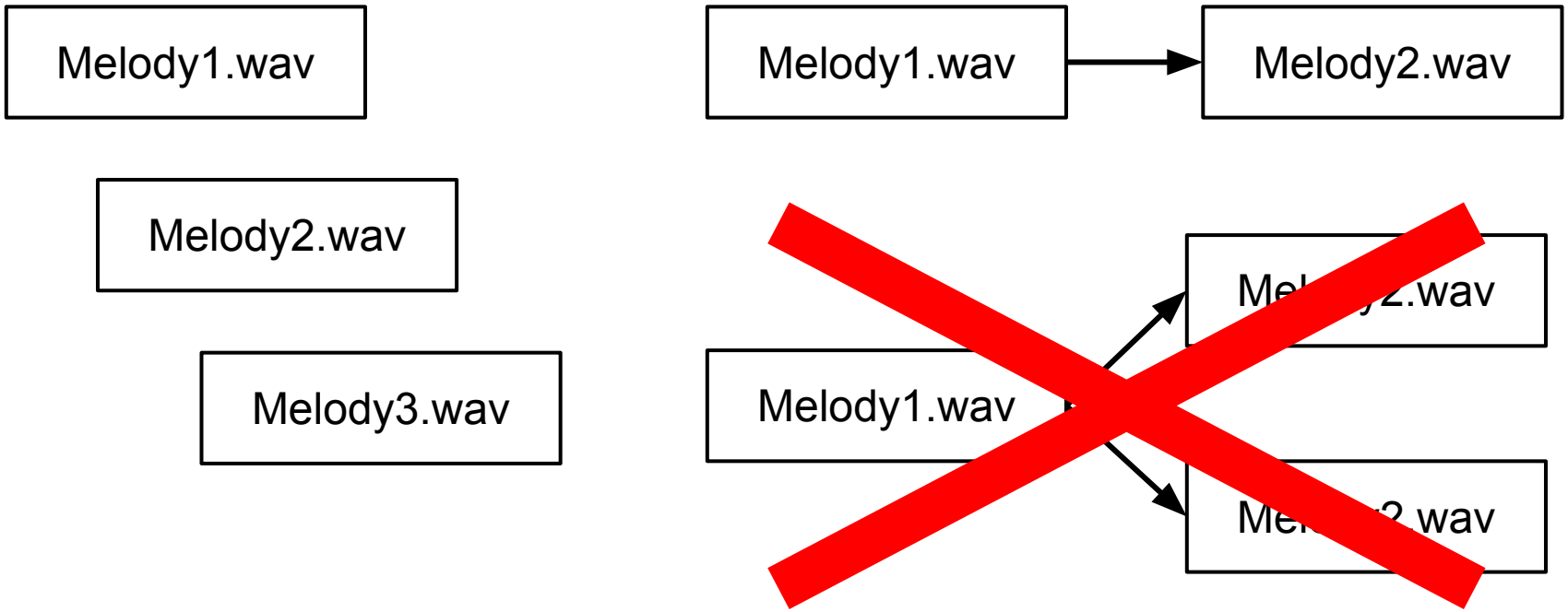
Musically meaningful structuring actions are simple and quick to do;

Both bottom up and top down construction are allowed in any combination;

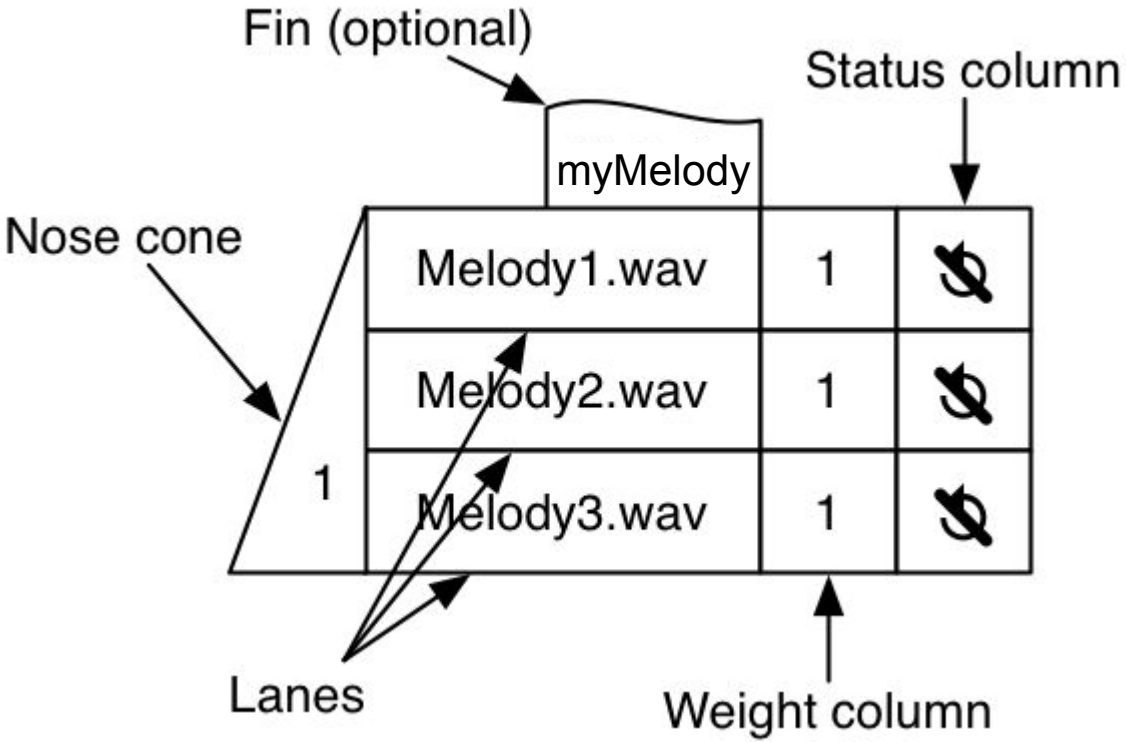
Affordances are designed for a wide range of users — children to experts — via progressive disclosure.

Introduction to the system

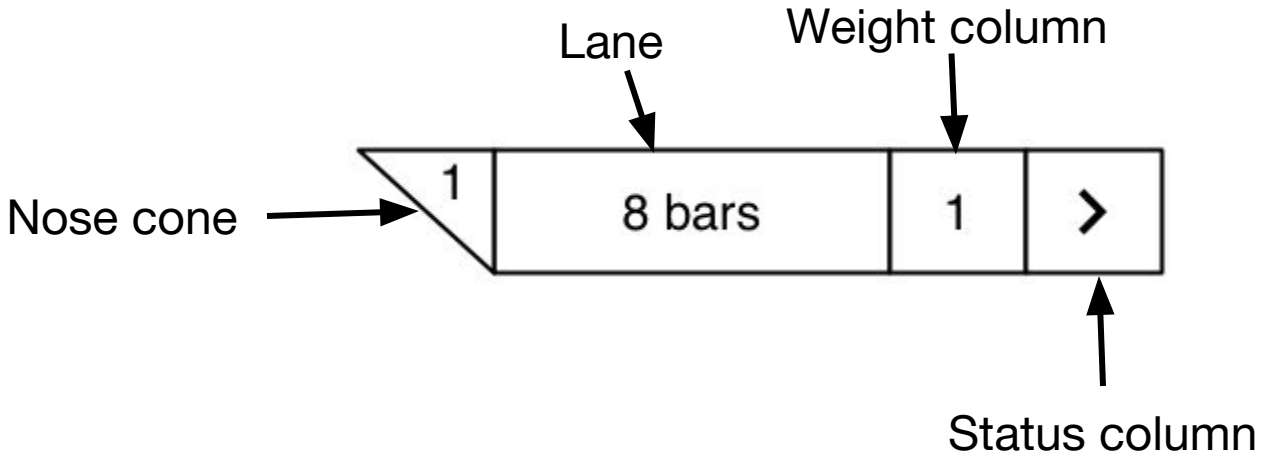
Samples and sequences



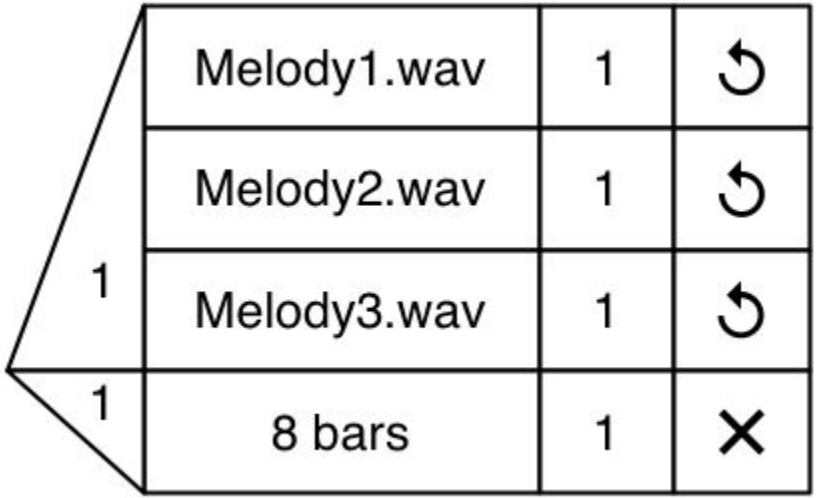
Soundable choosers



Time lanes and time choosers

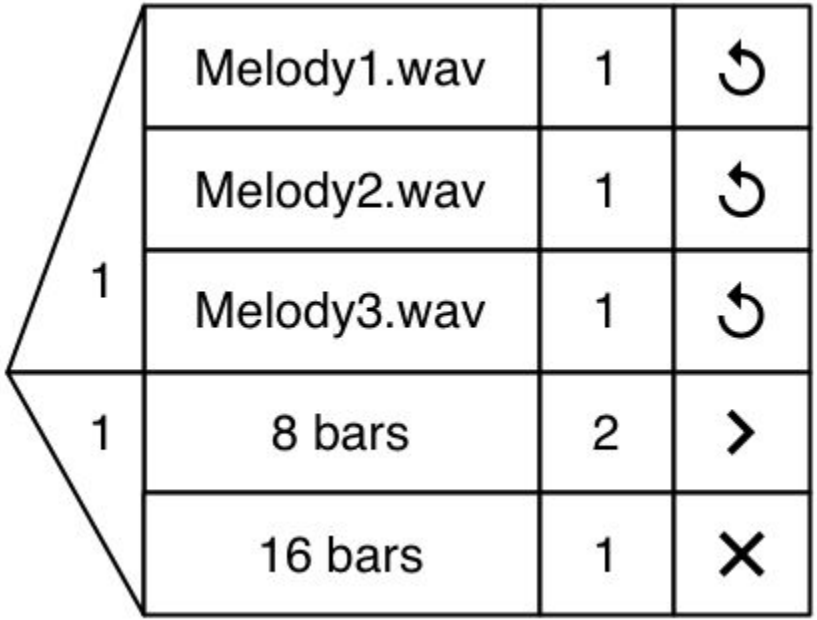


Full chooser



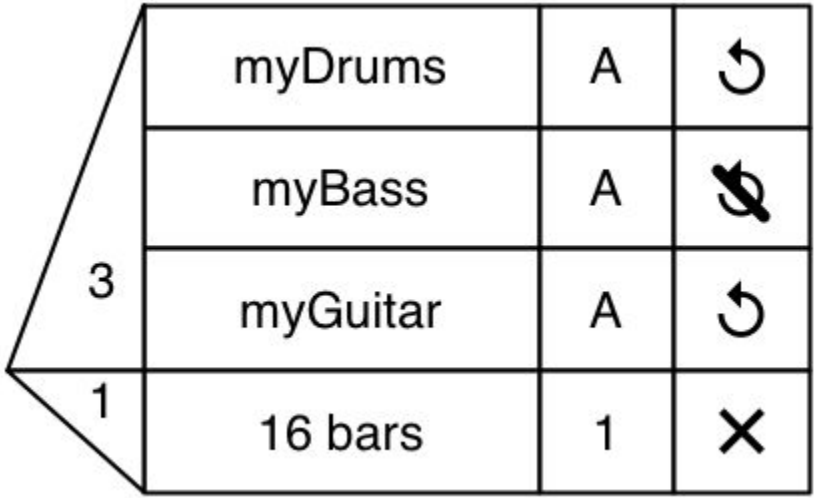
Looping one of three samples for eight bars.

Multiple time lanes



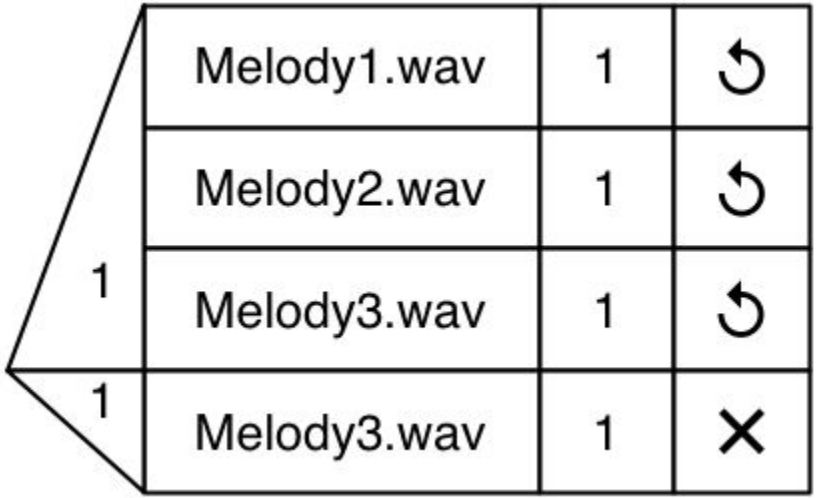
Looping one of three samples for either 8 bars with a soft stop, or for 16 bars with a hard stop.

Always play



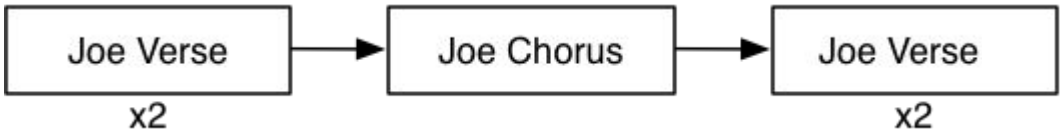
The mandatory lanes mean that the soundable nose cone must be either be three or zero.

Duration set using the length of a sample

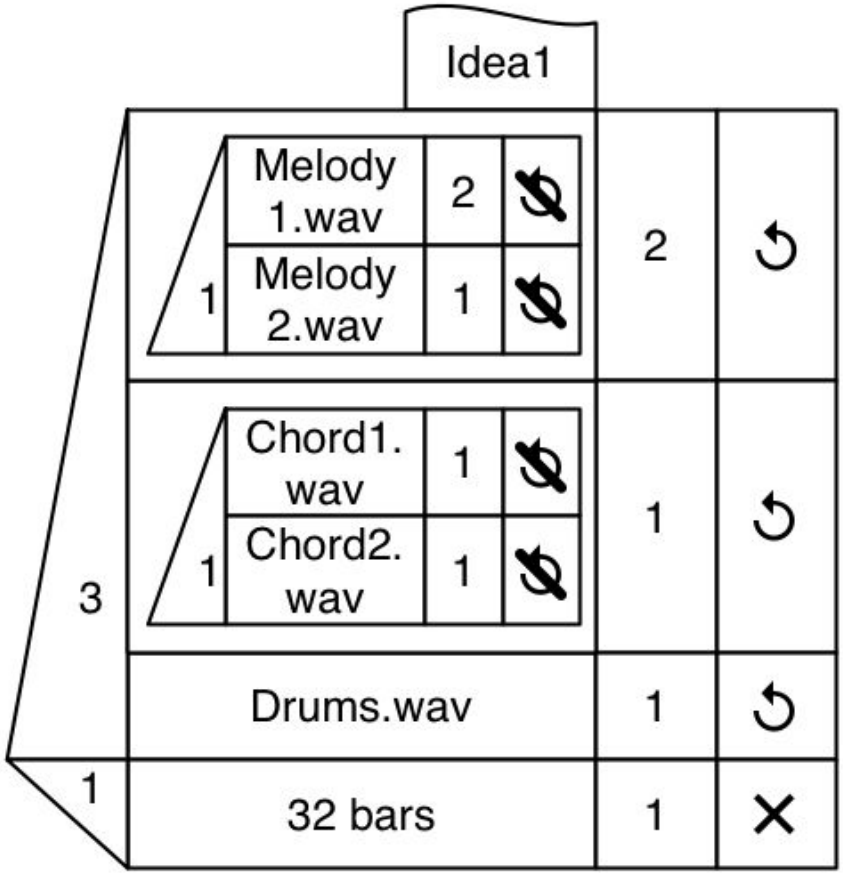


The sample in the time chooser is not audible - it is only used to set the duration of the chooser.

Boxes, repetition, and sequence



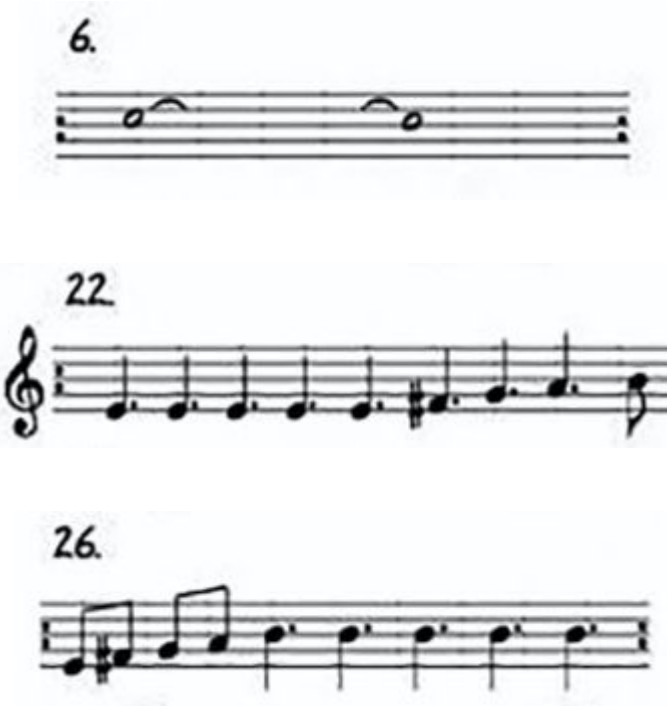
Nesting and recursion



Example 1 - phasing

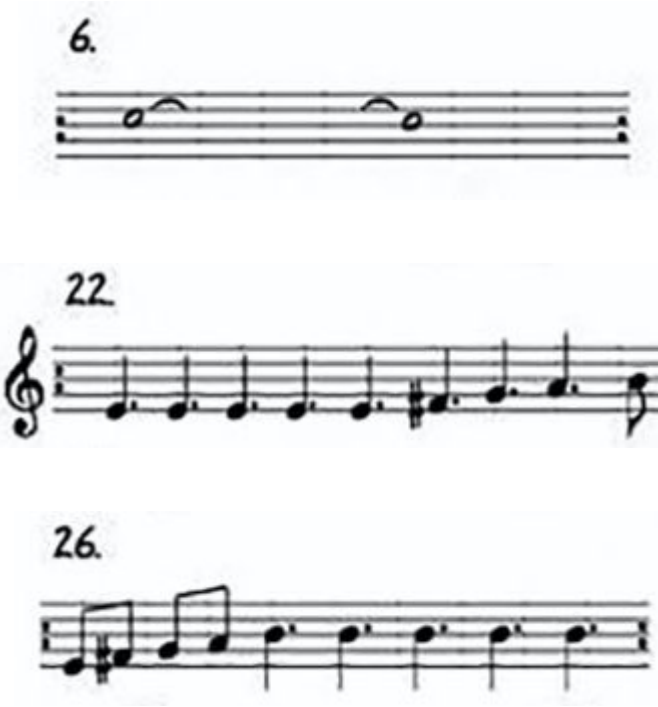
Samples taken from Terry Riley's *In C*

2	0.wav	A	↻
	6.wav	1	↻
	22.wav	1	↻
	26.wav	1	↻



Example 1 - phasing

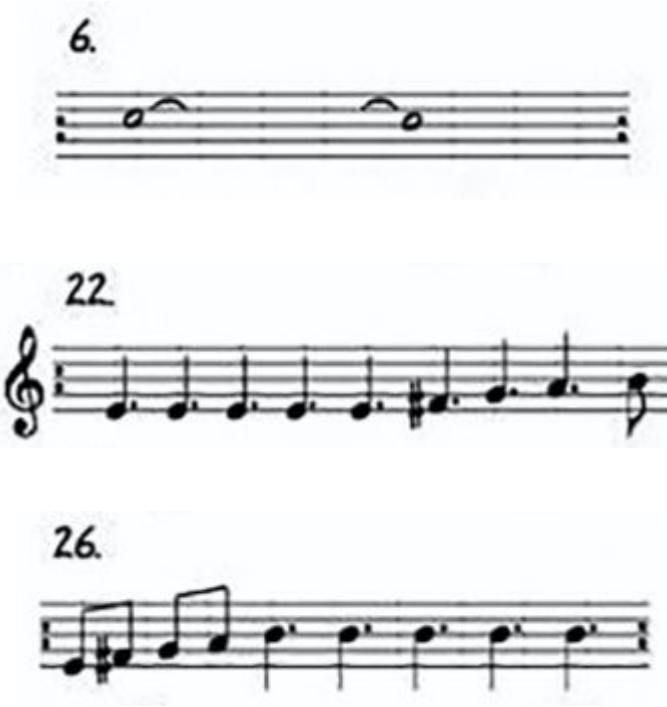
Samples taken from Terry Riley's *In C*



Example 1 - phasing

Samples taken from Terry Riley's *In C*

4	0.wav	A	↻
	6.wav	1	↻
	22.wav	1	↻
	26.wav	1	↻



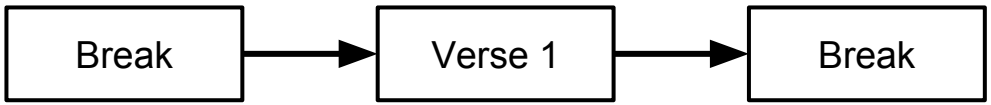
Example 2 - rock

Break

2	drums.wav	A	↻
	bass.wav	1	↻
	marimba.wav	1	↻
1	4 bars	1	X

Verse 1

3	bass.wav	1	↻
	guitar.wav	1	↻
	bv.wav	1	↻
	marimba.wav	1	↻
1	2 bars	1	X
	4 bars	1	X



Evaluation

The aim - an expressive system which is accessible to those with minimal programming skills and little musical training.

Programming walkthrough methodology

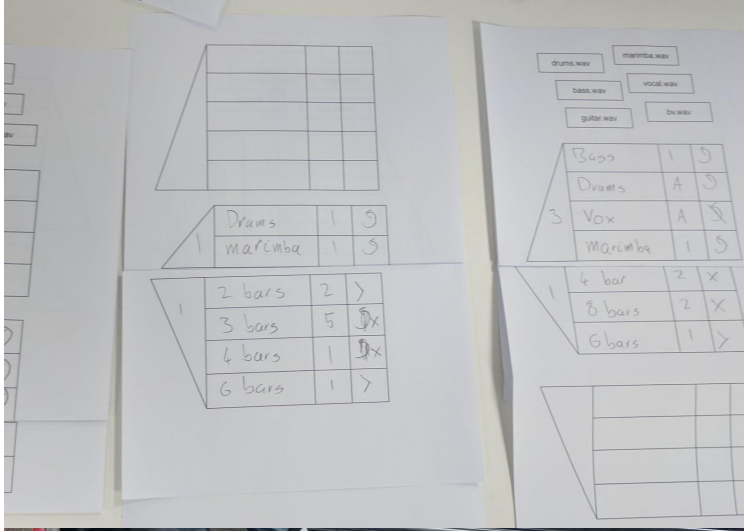
User testing

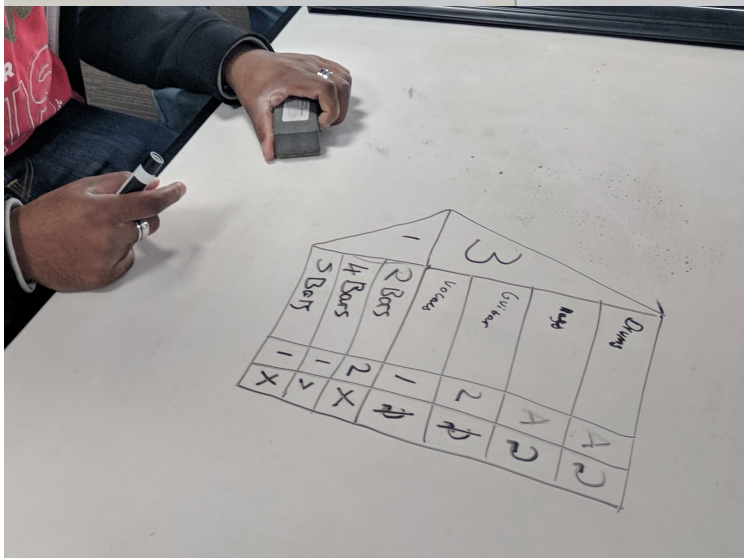
Wizard of Oz with paired participants, 10-12 in total, September/October

Testing of a graphical prototype

Method

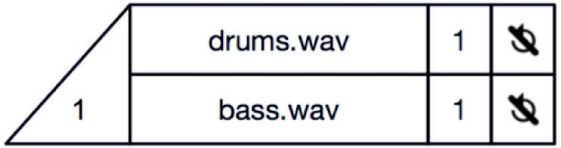
- Seven pairs of users took part in user tests utilising a Wizard of Oz prototyping methodology
- Took a short questionnaire before taking part in the tests — the participants were neither programmers nor traditional musicians
- Users asked to act as active participants — discuss their work, categorise any issues according to the programming walkthrough method (Bell et al., 1992, 1991):
    - Questions;
    - Suggestions;
    - Observations.





Scenarios

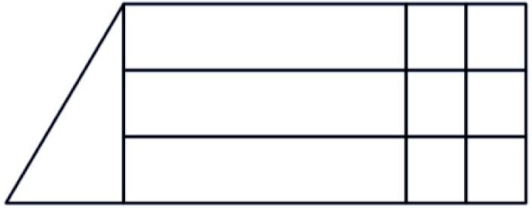
Scenario 1



Here is a Soundable Chooser with two samples.

- If this Chooser is played by itself, how many samples will play?
- How do you know?
- How likely is it that the drums.wav sample will play? How could you make it more likely to play?
- How could you make the Chooser play both samples?
- How would you make it play no samples?

Scenario 2



Using the provided template, make a Soundable Chooser which has three lanes – those lanes should contain looping drums, bass and guitar samples. Make it so that two play at once – the drums always play, and either bass or guitar will be selected with equal probability.

Next, make it so the guitar doesn't play.

Scenario 3

Here is a Time Chooser with two lanes.

- Describe what will happen when this Chooser is run by itself.
- The nosecone is currently set to 1. What else could it be set to? What would happen if it is changed?
- How could you make a duration of 2 bars most likely to be selected?

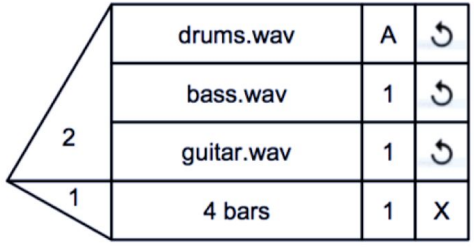
Scenario 4

Keeping the Soundable Chooser you made for scenario 2, create a single-lane Time Chooser using the supplied template.

- Make a four bar rest.
- Now you have made the rest, find two ways to quickly skip it. What do you think the nose cone value could be? What would happen if you gave the nose cone that value?
- What impact would there be if you changed the weight of the lane?

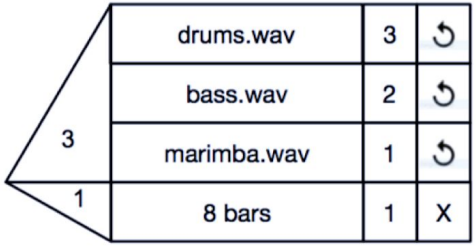
Next, take the Time Chooser and snap it onto the Soundable Chooser created in scenario 2. What is the impact of this?

The result of Scenario 4:



Scenario 5

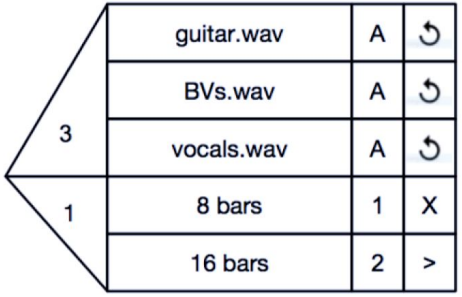
Look at this Chooser and say what will happen when the Chooser is played. The drums and bass samples are four bars long. The marimba sample is two bars long.



- How many samples will play? Will they loop or play once? What effect would changing the loop setting on the drums have?
- How long will the Chooser play for? What happens when the duration elapses?
- What would happen if the Time Chooser was set to a soft stop?
- What would happen if the Time Chooser nose cone stayed at 1 and the Soundable Chooser nose cone was changed to 2? To 1? To zero?
- How could you make it infinite playback? How could it be made into a rest? Skipped entirely?

Scenario 6

Here is a Chooser containing a Time Chooser with multiple lanes.



- What do you expect to happen in the Soundable Chooser?
- What will happen in the Time Chooser? Which lane is more likely to be selected? What are the consequences of the selection of the uppermost Time Chooser lane? What will be different if the lower Time Chooser lane is selected?
- What other values are possible for the nose cone of the Soundable Chooser?
- What other values are possible for the nose cone of the Time Chooser?

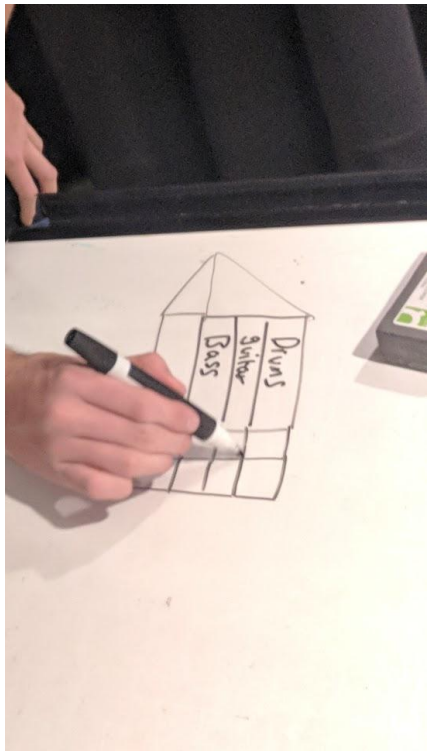
Scenario 7

Using the templates (provided on paper), create a Full Chooser which:

- Has four soundable lanes, three of which will play at any given time. Drums and bass, which always play, and are set to loop: guitar and vocals, where the guitar is twice as likely as vocals to be selected for playback. Neither should loop.
- Has three possible durations, of which one will be selected – 2 bars with a hard stop, 4 bars with a soft stop, and 5 bars with a hard stop. Make the 2 bar duration twice as likely to be selected as the 4 and 5 bar durations.

Scenario 8

Using the templates and samples available, make a piece of music which uses a sequence of three Choosers. The music will be recorded and shared online. The piece should be musically satisfying even if it is run only once. If it is run more than once it should be different in some way.



Reflection on design issues

Musical issues

- Stop behaviour was confusing to four of the seven pairs of users
- Progress bar request - reasonable, but difficult in a nondeterministic system
- Rest functionality confusing for some
- Thematic development is possible via nesting
- While Choosers can be used in linear music making, they are particularly suited to nonlinear music (e.g. game music)
- Some time taken in discussing the desirability of algorithmic music

Programming-related issues

- The nose cone shapes of the two Chooser types seemed effective in communicating their combinatorial usage
- Reuse or re-contextualisation of logic is a design goal and was observed in use, but was sometimes not possible where it didn’t make sense - the rationale behind these requests is instructive as it shows how users understand the tools in the system
- Users required access to metadata

Shared and existing knowledge

- One design motivation is to enable people to understand the system very quickly
- Some users wanted to be able to leverage their existing understanding of DAW software and found new paradigms frustrating — an example of technological framing (Orlikowski and Gash, 1994)
- Technological framing, and the expectations set by the use of commercial DAWs, may be an influence on user requests for a progress bar and the conversations on the desirability of nonlinear vs. linear playback that took place during the user tests

Metaphor

- Interface metaphors are very common and can be useful in communicating the roles of the software and setting realistic expectations when users are familiar with the original interface. However, such metaphors can become problematic if users are not familiar with the original interface
- Now most people are introduced to music production via software, and many do not use hardware, there is an opportunity to revisit some design assumptions
- Hard and soft stops — no clear existing metaphor for a soft stop – both for the function and the icon

Arithmetic

- The use of numbers in the UI was due to their universal familiarity and their ability to concisely represent many relationships. The decision to use numbers for several parameters was motivated by parsimony and consistency. However…
- The use of numbers for multiple parameters was perceived as negative by three participants (‘too many numbers man!’)

Design problems and candidate solutions

- On/off toggle for the Time Chooser nose cone — replaces 0/1
- ‘Always play’ to be replaced by ∞
  - Maximum possible weight - priority boarding
  - In the nose cone — play all available lanes
  - As a duration — play forever
- New stop behaviour – stop when a specified lane stops
- Reorder tutorial materials — introduce Time Choosers in the context of a Full Chooser, with the rest functionality introduced later as a special case

What’s next?

June to August

From the findings of the first round of user testing, develop the specification of the system and consider new evaluation approaches (e.g. new kinds of tasks). Implement the new specification in software in preparation for the second round of user tests.

September to November

Using the newly implemented software, run the second round of user tests in September. Analysis and writing up to be complete by the end of November.

December

Final specification, design, and implementation of the software based on the findings from the second round of user testing.

Conclusions

Choosers were developed to allow non-programmers access to algorithmic composition tools and processes. The design principles were to leverage parsimony 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.

The user tests outlined here show that non-programmers were able to successfully use Choosers to create a number of short pieces of music. Future work will focus on the refinement and re-evaluation of the Chooser notation and supporting materials.