

Open Research Online

The Open University's repository of research publications and other research outputs

Choosers: A Visual Programming Language for Nondeterministic Music Composition by Non-Programmers

Thesis

How to cite:

Bellingham, Matt (2022). Choosers: A Visual Programming Language for Nondeterministic Music Composition by Non-Programmers. PhD thesis The Open University.

For guidance on citations see [FAQs](#).

© 2020 Matthew Francis Bellingham



<https://creativecommons.org/licenses/by-nc-nd/4.0/>

Version: Version of Record

Link(s) to article on publisher's website:

<http://dx.doi.org/doi:10.21954/ou.ro.0001400f>

Copyright and Moral Rights for the articles on this site are retained by the individual authors and/or other copyright owners. For more information on Open Research Online's data [policy](#) on reuse of materials please consult the policies page.

oro.open.ac.uk

Choosers: A Visual Programming Language for Nondeterministic Music Composition by Non-Programmers

Matt Bellingham

A thesis presented for the degree of
Doctor of Philosophy

Supervised by:
Dr. Simon Holland
Dr. Paul Mulholland



Open University, UK
December 2021

Abstract

This thesis focuses on the design of Choosers, a prototype algorithmic programming system centred around a new abstraction (of the same name) designed to allow non-programmers access to non-deterministic music composition methods.

Algorithmic composition typically involves structural elements such as indeterminism, parallelism, choice, multi-choice, nesting, weighting, and looping. There are powerful existing tools for manipulating these and other elements of music. However, while these systems give substantial compositional power to musicians who are also skilled programmers, many musicians who lack programming skills find these tools inaccessible and difficult to understand and use. This thesis presents the design and evaluation of a prototype visual programming language designed to allow structural elements of the kind involved in nondeterministic music composition to be readily visualised and manipulated, while making little or no demand on programming ability.

Initially, a Cognitive Dimensions of Notations review of a representative selection of user interfaces for algorithmic composition software was conducted. The review led to a set of findings used to identify candidate design principles which were then tested via a series of design exercises. The findings from these design exercises led to the development of a new abstraction, the Chooser, via a series of iterative design cycles. Once a candidate design had been finalised it was evaluated with participants via two sets of programming walkthroughs, with the findings from each step used to refine the formalism. The final study used Choosers as a design probe through a series of interviews with domain experts in which manipulable compositions were introduced to prompt discussions on potential future implications for music computing education, music production, and music composition.

Contents

1	Introduction	1
1.1	Personal motivation	1
1.2	Research questions	2
1.3	Useful terminology	3
1.4	Thesis overview	3
2	Literature review	5
2.1	Aural tradition	5
2.2	Common music notation	6
2.3	Phonography	7
2.4	Trackers, sequencers, and DAWs	8
2.4.1	Trackers	8
2.4.2	MIDI sequencers	8
2.4.3	Digital Audio Workstations	9
2.4.4	Visual design of DAWs	9
2.5	Technological frames	10
2.6	Algorithmic composition	13
2.7	Usability and expressivity trade-off	13
2.8	Critical review of representative algorithmic music composition systems	15
2.8.1	<i>WolframTones</i>	15
2.8.2	<i>Noatikl</i>	15
2.8.3	<i>Scratch</i>	17
2.8.4	<i>Sonic Pi</i>	17
2.8.5	<i>Fractal Tune Smithy</i>	18
2.8.6	<i>Manhattan</i>	19
2.8.7	<i>OpenMusic</i>	20
2.8.8	<i>Max</i>	20
2.8.9	<i>Max for Live</i>	22
2.8.10	<i>SuperCollider</i>	23
2.9	Direct Combination: a user interaction design principle	24
2.10	Diagrammatic representations	24
2.10.1	Decision trees (DTs) and influence diagrams (IDs)	25
2.10.2	Grammar-based hierarchical structures	25
2.10.3	Petri Nets	26
2.11	End-user development	27
2.12	Progressive disclosure	29
2.13	Conclusions	30

3	Methodology	31
3.1	Cognitive Dimensions of Notations analysis	31
3.2	Research through Design	32
3.3	Cognitive walkthrough	33
3.4	Programming walkthrough	34
3.5	Walkthrough protocol	34
3.5.1	Analysis	35
3.6	Wizard of Oz	36
3.7	Protocol for interviews with experts	36
3.8	Potential limitations of the proposed methodology	37
3.9	Conclusions	38
4	Cognitive Dimensions of Notations analysis	41
4.1	Introduction	41
4.2	The Cognitive Dimensions of Notations	42
4.2.1	Viscosity	42
4.2.2	Abstraction	43
4.2.3	Hidden dependencies	45
4.2.4	Premature commitment	46
4.2.5	Secondary notation	46
4.2.6	Visibility	47
4.2.7	Closeness of mapping	50
4.2.8	Consistency	52
4.2.9	Diffuseness	52
4.2.10	Error-proneness	56
4.2.11	Hard mental operations	56
4.2.12	Progressive evaluation	57
4.2.13	Provisionality	58
4.2.14	Role-expressiveness	60
4.3	Towards design tools for the compositional representation of time	61
4.4	Conclusions	65
5	Design considerations for an algorithmic composition language	67
5.1	CDN review outcomes	67
5.1.1	Heuristics from the CDN analysis	68
5.2	Candidate frameworks	69
5.2.1	Design exercise using Direct Combination	69
5.2.2	Design exercise using Petri Nets	70
5.2.3	Representing structural musical processes in Petri Nets	73
5.2.4	Design heuristics, observations, and design decisions resulting from the PN design exercise	76
5.3	Conclusions	78
6	Design and development of the preliminary version of Choosers	81
6.1	Implementation	81
6.1.1	Sequence	82
6.1.2	Nondeterminism, choice, parallelism and multi-select	83
6.1.3	Looping	83
6.1.4	Deterministic duration and stop behaviour	84
6.1.5	Arrangement	85
6.1.6	Nesting and referencing Choosers	85
6.1.7	Variables and Variable Choosers	86

6.2	Preliminary sketches in the design space	87
6.2.1	Abstracting from Music Petri Nets	88
6.2.2	Considerations for nondeterministic weighted choice	91
6.2.3	Looping and its implications	94
6.2.4	Hard and soft stops	95
6.2.5	Differentiating Choosers: Soundable and Time Choosers	100
6.2.6	External input	101
6.2.7	Variables	101
6.2.8	Nesting vs. referencing Choosers	102
6.2.9	Comparing Choosers with Petri Nets	103
6.2.10	Reflections on the design of Choosers	107
6.3	Conclusions	110
7	First user study	111
7.1	Problem setting	111
7.2	Method	112
7.2.1	Participants	112
7.3	Walkthrough scenarios	115
7.3.1	Scenario 1: Chooser basics	115
7.3.2	Scenario 2: making a simple Chooser	115
7.3.3	Scenario 3: Time Choosers	117
7.3.4	Scenario 4: Full Choosers	117
7.3.5	Scenario 5: Understanding a musical example	119
7.3.6	Scenario 6: Multiple Time Chooser lanes	120
7.3.7	Scenario 7: creating a specific Full Chooser	120
7.3.8	Scenario 8: Sequence; playground	121
7.3.9	Final formative questions	122
7.4	Identified issues from user testing	123
7.5	Reflection on design issues	124
7.5.1	Musical issues	124
7.5.2	Programming-related issues	125
7.5.3	Shared and existing knowledge	125
7.5.4	Metaphor	126
7.5.5	Arithmetic	126
7.6	Design problems and candidate solutions for the second user study	126
7.7	Conclusions	127
8	Design considerations and inspection evaluation for the second user study	129
8.1	Adopted changes to the notation	129
8.1.1	Infinity	129
8.1.2	Stop behaviour	131
8.1.3	New icons and a column-based layout	133
8.1.4	Management of soundable content in Time Choosers	133
8.2	Changes to the documentation	134
8.2.1	Explaining how infinity is used	134
8.2.2	Improved scenario content and order	135
8.2.3	Enhanced tutorial material for hard and soft stops	135
8.2.4	Clarification on vertical alignment	135
8.2.5	Clarification on Time Chooser nose cone	135
8.2.6	Clarification on order of priority	135
8.3	Conclusions	135

9	Second user study	139
9.1	Method	139
9.1.1	Participants	139
9.1.2	Walkthrough protocol	141
9.2	Walkthrough scenarios—results	141
9.2.1	Scenario 1: simple sequence	141
9.2.2	Scenario 2: Soundable Choosers	145
9.2.3	Scenario 3: infinite weight	146
9.2.4	Scenario 4: Full Chooser, including hard and soft stops	147
9.2.5	Scenario 5: Multi-lane Time Choosers	149
9.2.6	Scenario 6: Playground	152
9.2.7	Final formative questions	152
9.3	Reflection/discussion	153
9.3.1	Surfacing musical moves	153
9.3.2	Identifying and implementing suitable metaphors for non-programmers	154
9.3.3	User expectations	155
9.3.4	Possible improvements for design and implementation	155
9.4	Conclusions from the second user study	156
10	Design exploration in preparation for the final user study	159
10.1	Methodological considerations	159
10.2	Design iterations led by issues identified in user testing	159
10.2.1	Increasing the granularity of the loop control	159
10.2.2	New stop icons	160
10.2.3	Metadata visibility in lanes	160
10.2.4	Visualisation and post-run trace	163
10.3	Speculative design changes	163
10.3.1	Blank lane	163
10.4	Musical examples	165
10.4.1	Piano and violin example	165
10.4.2	Drum machine example	169
10.5	Conclusions	172
11	Third user study	175
11.1	Design of the third user study	175
11.1.1	Introducing the interviewees to Choosers	175
11.1.2	Musical examples for use in the interviews	176
11.1.3	Questions for participants	176
11.2	Participants	181
11.3	Changes made to code by the participants/interviewees	181
11.4	Data analysis	181
11.5	Findings from the expert interviews	181
11.5.1	Reflections on linearity	182
11.5.2	Design consistency	182
11.5.3	Free placement of objects	183
11.5.4	Visibility of metadata and global controls	184
11.5.5	Similarities to other systems	184
11.5.6	Who might want to use this?	186
11.5.7	Who would not want to use it?	189
11.5.8	Would you use it?	190
11.5.9	Enhancements	192
11.5.10	Ecosystem	192

11.5.11 Scope of the software and initial design assumptions	194
11.6 Conclusions	194
12 Conclusions	197
12.1 Answers to the research questions	197
12.2 Limitations	200
12.3 Future work	202
12.4 Concluding remarks	203
References	205
A User guide for the first user study	217
A.1 The play area	217
A.2 Adding material	217
A.2.1 Single-lane Choosers	217
A.2.2 Multi-lane Choosers	218
A.2.3 Exploring the parameters of the Chooser	218
A.2.4 Nesting Choosers	219
A.2.5 Constraining the duration of a Chooser	219
A.2.6 Time Chooser with multiple lanes	225
A.2.7 Alternative to nesting via named Choosers	225
A.2.8 Forcing lanes to always play	225
A.3 Creating a sequence	226
A.3.1 Sequence	226
A.3.2 Top-down or bottom-up	227
A.3.3 Arrangement example - top down	227
A.3.4 Arrangement example - bottom up	228
A.4 Advanced techniques	229
A.4.1 Variables	229
A.4.2 The variable list	230
A.4.3 Example: variable selection	231
A.4.4 Multi-column variable Choosers	232
A.4.5 Zero in the nose cone	233
B Choosers v1 design table	237
C Example user test transcript	239
C.1 Scenario 1: simple sequence	239
C.2 Scenario 2: Soundable Choosers	241
C.3 Scenario 3: infinite weight	246
C.4 Scenario 4: Full Chooser, including hard and soft stops	248
C.5 Scenario 5: Multi-lane Time Choosers	252
C.6 Scenario 6: Playground	256
C.7 Final questions	258
D Design exercises	259
D.1 Design exercises used to inform Choosers v1	259
D.1.1 Lessons from an informal design exercise using Direct Combination . . .	259
D.2 Design exercises used to inform Choosers v2	263
D.2.1 Icon design	263
D.2.2 Testing designs via programming walkthroughs	267
D.2.3 Design sketches for soundable content in Time Choosers	268
D.2.4 Control over lane contents	268

D.3	Design exercises used to inform Choosers v3	271
D.3.1	Shaded/filled weights vs. numbers.	271
D.3.2	Left-to-right sequencing via arrows	273
D.3.3	Design arc—programmatic control of Soundable Chooser lane parameters	274

List of Figures

2.1	The mixer in <i>Pro Tools</i> , showing the use of skeuomorphic faders and rotary pots. .	11
2.2	<i>Ardour's</i> mixer, showing the use of a ribbon controller-style strip rather than a traditional fader.	12
2.3	Signal flow in <i>Tracktion</i> (Rink, 2017).	12
2.4	Four points in the usability/expressiveness space (Repenning and Ioannidou, 1997). .	14
2.5	The main interface of <i>WolframTones</i> (Wolfram Research Labs, 2005); screenshot taken in 2014.	16
2.6	An example of a layered interface: the composition controls in <i>WolframTones</i> (Wolfram Research Labs, 2005); screenshot taken in 2014.	16
2.7	Data connection in <i>Noatikl</i> (Intermorphic, 2015b).	17
2.8	A <i>Scratch</i> patch, showing the use of interlocking block shapes to limit the opportunity for user error.	18
2.9	The <i>Sonic Pi</i> IDE (Aaron and Blackwell, 2013), showing control buttons, workspace tabs, the editor pane, the information pane, and the error pane.	18
2.10	One of several window types in <i>Fractal Tune Smithy</i> (Walker, 2011).	19
2.11	<i>Manhattan</i> (Nash, 2014), showing the use of spreadsheet-style formulae to randomise a cell's pitch and volume parameters.	20
2.12	An example <i>OpenMusic</i> patch (Bresson et al., 2011).	21
2.13	A simple patch in <i>Max</i> (Cycling '74, 2019), showing the interrelationships between objects denoted by virtual patch cables.	21
2.14	Dense connections in <i>Pure Data</i> creating visual 'spaghetti' (Duignan et al., 2004). Image from Farnell (2008).	22
2.15	Auto-complete suggestions in <i>SuperCollider 3.6</i>	23
2.16	<i>SuperCollider</i> help file for the <i>SinOsc</i> sine oscillator.	24
2.17	An Influence Diagram (ID) showing the choices to be made by a diagnosing doctor (Jensen et al., 2006).	25
2.18	Primitive constructs in Petri Nets, showing a series of common structures (Chen, 2003).	27
2.19	Cost-scope trade-offs in End-user development (EUD) tools (Fischer et al., 2004). SDL stands for Specification and Description Language.	28
2.20	The choice of a suitable metaphor to support users of end-user programming systems (Ko et al., 2004). Ideal metaphors would be sufficiently abstract to describe many systems while being concrete enough for analogical reasoning.	29
4.1	Dense connections in <i>Pure Data</i> (Farnell, 2008), creating 'spaghetti code' which increases viscosity.	42
4.2	The transport controls in <i>iTunes</i> with playback stopped, showing the play button. .	44
4.3	The transport controls in <i>iTunes</i> during playback, showing the pause button. . . .	44
4.4	The drum machine interface from the <i>Cylob Music System</i> (Jeffs, 2010).	44
4.5	Lars Kindermann's <i>MusiNum</i> (2006).	45
4.6	Potential layout for structure-aware composition software.	46
4.7	Filter controls in <i>Mixtikl</i> (Intermorphic, 2015a).	47

4.8	An example <i>Max</i> patch taken from Autechre's <i>Confield</i> (Tingen, 2004).	48
4.9	Comments in a <i>SuperCollider</i> patch.	48
4.10	<i>Cylob Music System</i> interface demonstrating both visibility and juxtaposability. . .	49
4.11	<i>SuperCollider</i> help file for the <i>SinOsc</i> sine oscillator.	49
4.12	Tooltip shown in <i>SuperCollider</i> , showing both the method order and default parameter settings.	50
4.13	The form-based interface of <i>Fractal Tune Smithy</i> (Walker, 2011).	50
4.14	Data connection in <i>Noatikl</i> (Intermorphic, 2015b).	51
4.15	Animal breeding metaphor as implemented in <i>Maestro Genesis</i> (Szerlip and Hoover, 2012).	51
4.16	Selection of generations in <i>Maestro Genesis</i> (Szerlip and Hoover, 2012).	52
4.17	Signal flow in <i>Mixtikl</i>	53
4.18	One of several window types in <i>Fractal Tune Smithy</i> (Walker, 2011).	54
4.19	Standard MacOS save dialogue.	54
4.20	<i>SuperCollider</i> save dialogue.	54
4.21	<i>Strasheela</i> (Anders, 2012).	55
4.22	<i>Impromptu</i> (Sorensen, 2010).	55
4.23	A <i>SuperCollider</i> patch made more verbose by commenting individual lines.	55
4.24	The 'ArpeggioSequenceEngine' patch from the <i>SoundHelix</i> documentation (Schürger, 2016), exhibiting poor discriminability and high error-proneness.	56
4.25	An example error message in <i>SuperCollider</i>	56
4.26	Auto-complete suggestions in <i>SuperCollider</i> 3.6.	57
4.27	An example of the => patching syntax in <i>ChuckK</i> (Wang and Cook, 2018).	57
4.28	A simple patch in <i>Max</i> (Cycling '74, 2019), showing the interrelationships between objects denoted by virtual patch cables.	58
4.29	<i>Musical Algorithms</i> (Middleton, 2004).	59
4.30	<i>Wolfram Tones</i> (Wolfram Research Labs, 2005), taken in 2014.	59
4.31	Possible layout allowing common material to be shared between sections.	60
4.32	<i>AthenaCL</i> (Ariza, 2011).	61
4.33	<i>FractMusic</i> (Diaz-Jerez, 2012).	62
4.34	<i>Lexikon-Sonate</i> (Essl, 2010).	62
4.35	The <i>Impro-Visor</i> interface (Keller, 2019) utilising a lead sheet metaphor.	63
4.36	The <i>Improvvisor</i> software (Percussa, 2012).	64
5.1	<i>Peaches en Regalia</i> by Frank Zappa represented as a Music Petri Net (Baratè, 2009).	71
5.2	An original implementation, in <i>PIPE</i> , of the Music Petri Net shown in Figure 5.1.	72
5.3	A simple Petri Net in an early <i>Pure Data</i> (Puckette, 1997) PN system.	72
5.4	Sequencing from Chen (2003) (left) and an implementation in <i>SuperCollider</i> (right).	73
5.5	Concurrency from Chen (2003) (left) and an implementation in <i>SuperCollider</i> (right).	74
5.6	Synchronisation from Chen (2003) (left) and an implementation in <i>SuperCollider</i> (right).	74
5.7	Confusion from Chen (2003) (left) and an implementation in <i>SuperCollider</i> (right).	75
5.8	Merging from Chen (2003) (left) and an implementation in <i>SuperCollider</i> (right).	76
5.9	Four Petri Net tests created in <i>SuperCollider</i> (McCartney, 2002) using the <i>Thr44</i> quark (Cardoso, 2014).	77
6.1	Samples are shown in boxes.	82
6.2	A sequence is assembled via arrows.	82

6.3	A sequence, packaged into a box, and used twice in a sequence. Note that the order of the sequence is indicated by the direction of the arrows. In this instance, the sequence inside the upper box (Melody1.wav then Melody2.wav) will play before the contents of the next box (Melody3.wav) are played. Finally, a duplicate of the first box will play.	82
6.4	An annotated Chooser	83
6.5	An annotated Time Chooser	83
6.6	A Full Chooser	84
6.7	An example of a nested Chooser.	85
6.8	A version of Figure 6.3 which replaces nesting with referencing. The sequence packaged into the box named myBox is referenced twice in the sequence at the bottom of the figure.	85
6.9	An example of referencing a Chooser's name inside the lane of a parent Chooser. This example is functionally identical to the nested example in Figure 6.7.	86
6.10	A Variable Chooser called myVar.	86
6.11	The variable myVar is used to control the number of repeats of two Choosers. The variable selection will re-run each time the Variable Chooser is referenced, which may lead to different values being used for each repeat.	87
6.12	Both references to the Variable Chooser append a user-definable string of text (in this case, @sectiona) to the end of the name, showing that the same selection should be used for both.	88
6.13	<i>Peaches en Regalia</i> by Frank Zappa represented as a Music Petri Net (Baratè, 2009). This is a duplicate of Figure 5.1 reproduced here for the reader's convenience. . . .	89
6.14	A simplified version of the <i>Peaches En Regalia</i> Music Petri Net (Baratè, 2009) shown in Figure 6.13. This version uses a flowchart-style musical arrangement and spreadsheet-style abstractions to represent the transposing sections and the weighting of nondeterministic choices.	89
6.15	A design sketch extending the ideas behind Figure 6.14 to include explicit use of audio samples, nesting, concurrency and weighted nondeterminism.	90
6.16	A test which further explores the sequencing and abstraction elements from Figure 6.14 and Figure 6.15. A sequence of musical sections is shown in the upper left, with the sections populated via named abstractions. The abstractions, in turn, are populated by instruments/samples which are selected via weighted nondeterministic choice.	91
6.17	An abstraction which combines both a nose cone on the left and a weight column on the right.	92
6.18	An early sketch of a Chooser, showcasing concurrency, conflict, and weighted choice. Note that the nose cone is set to 2, meaning that two of the three lanes will be selected to be played. The weight column (far right) shows that the Melody1.wav sample is twice as likely to be selected as the other two lanes.	93
6.19	An early sketch to explore a design for a time lane, which is always at the bottom of a Chooser. In this instance, the time lane contains a nested Chooser to allow for nondeterministic duration. The status column of the time lane uses 'M' for 'timing master'.	93
6.20	An alternative to graphically nesting a Chooser in a lane; this sketch shows how Choosers can be named and referenced from inside other Choosers.	94
6.21	An early annotated Chooser, introducing the status column. All three lanes have looping disabled.	94
6.22	A sketch showing a nested child Chooser in a lane of the parent Chooser. All lanes have weights and a status column showing that all lanes have looping disengaged.	95

6.23	An early design to explore stop behaviour; the triangular ‘warning’ icon represents a hard (or ‘rude’) stop, meaning that any soundable content that is playing when the duration is reached will be stopped immediately.	96
6.24	This sketch is functionally identical to Figure 6.23, but uses named Choosers. . . .	96
6.25	A Chooser with one nose cone and two time lanes.	98
6.26	An early sketch of a Chooser with visually differentiated time lanes using a three-dimensional lane design and a shared nose cone.	98
6.27	A sketch of a Chooser with visually differentiated time lanes. This example uses additional vertical lines in the time lanes, and maintains a shared nose cone for all lanes.	98
6.28	A sketch of a Chooser with visually differentiated time lanes. As with the previous two examples, this design maintains a shared nose cone for all lanes.	98
6.29	Testing a double line between the soundable (upper) and time (bottom) lanes in a Chooser. This design uses a shared nose cone. Note the early test of an alternative icon for a hard stop (×).	99
6.30	A Chooser with a separate nose cone for the soundable lanes. This example has no nose cone for the time lanes, with the assumption that one time lane will be selected. This design was rejected as it removes the ability to remove the time lanes from selection.	99
6.31	A sketch of a Chooser with differentiated soundable and time lanes via the use of two nose cones. This design distinguishes lane types while allowing the nose cone values to be independently set. The final design, shown in Figure 6.32, uses a downwards-sloping nose cone for the Soundable Chooser and and upwards-sloping nose cone for the Time Chooser.	99
6.32	A Full Chooser, consisting of a Soundable Chooser on the top and a Time Chooser on the bottom.	100
6.33	A Full Chooser with an audio sample in a Time Chooser lane. The duration of the sample will set the duration of the Full Chooser; as the lane is set to a hard stop the currently playing sample (either <code>Melody1.wav</code> or <code>Melody2.wav</code>) will stop as soon as the duration has elapsed. <code>Melody3.wav</code> will not be played as it is in a Time Chooser lane.	100
6.34	Using the current semantics, if the user wants the Full Chooser to use <code>Melody3.wav</code> for both duration and soundable output, it must appear both in the Time Chooser and in a selectable lane in the Soundable Chooser.	101
6.35	A sequence with an OSC message connected to a Soundable Chooser. The OSC message acts as a ‘start’ message for the Chooser, allowing for software or hardware control of playback.	101
6.36	A sketch used to explore the functionality of Variable Choosers. The Variable Chooser named ‘Var’ (left) will select either 0 or 1 with equal probability when it is called. ‘Var’ is referenced from the nose cones in both the Soundable and Time Choosers combined into a Full Chooser on the right. Note that the default behaviour of a Variable Chooser is to run afresh each time it is called, leading to four possible outcomes (1 and 1; 1 and 0; 0 and 1; 0 and 0).	102
6.37	A sequence in Choosers (top) and in Music PNs (bottom).	104
6.38	Splitting from one to two simultaneous music objects in Choosers (top) and Music PNs (bottom).	104
6.39	Alternative choice in Choosers (top) and Music PNs (bottom).	104
6.40	Probabilistic weight in Choosers via lane weights (top) and in Music PNs via the probabilistic weight of arcs (bottom).	105
6.41	An extension of Figure 6.40 with infinite looping enabled on three of the four music objects.	105

6.42	Arithmetic 'x2' notation in Choosers (top), and a PN repeat (bottom).	106
6.43	Arithmetic repeat notation, as shown here, means 'play this object twice'. In this case one lane will be selected and played, and then the Chooser will reselect and run again.	106
7.1	An example of a participant sketch made using paper templates and the corresponding <i>SuperCollider</i> template code, edited by the facilitator during the sessions to enable participants to hear the results of their work.	113
7.2	Participant Chooser work using paper templates.	114
7.3	Participant Chooser work using a whiteboard.	114
7.4	The eight scenarios used in the first user study. In each case, italicised text is used to provide context (e.g. the use of tutorial videos). The main text shows the instructions given directly to the participants.	116
7.5	Soundable Chooser used in scenario 1 of the first user study.	117
7.6	Blank paper template used in scenario 2 of the first user study.	117
7.7	Time Chooser used in scenario 3 of the first user study.	117
7.8	Paper template used in scenario 4 of the first user study.	118
7.9	An example expected result of scenario 4 from the first user study.	118
7.10	Full Chooser used in scenario 5 of the first user study.	119
7.11	The Full Chooser used in scenario 6 of the first user study.	120
7.12	Completed paper Chooser templates used in scenarios 7 and 8 as part of the first user study.	121
8.1	An example of the final design of Choosers v2. Note the use of infinity (used here in the Soundable Chooser's nose cone); no internal columns; new icons for hard stop (the upper three Soundable Chooser lanes) and soft stop (the vocal .wav lane); new weight icon; and mute/unmute icon for soundable content in a Time Chooser lane (the marimba.wav lane).	130
8.2	The 'always play' weight used in Choosers v1 (left); the use of infinity as a weight, used in Choosers v2 (right).	130
8.3	Three ways to create a Chooser with infinite playback: a Soundable Chooser with looping lanes and no Time Chooser (left); a Full Chooser with a deselected Time Chooser via a zero in the nose cone (middle); a Full Chooser with ∞ as the duration (right).	131
8.4	Infinity used in the nose cone to play all available lanes. Note that the example on the left has three lanes with non-zero weights, so all three lanes will be selected to play. The example on the right has one lane with a weight of zero, meaning that only two lanes are available and so only two lanes will be selected to play.	131
8.5	A sequence to show the use of Time Choosers and hard stops for exact duration control.	132
8.6	A sketch showing an on/off button replacing a Boolean 0/1 choice in the nose cone of a Time Chooser. Note that a stop icon is still used in the Time Chooser in this early sketch; this was a test of a master stop behaviour control which would set all stops in the Soundable Chooser lanes to either a hard or soft stop in a single action.	133
8.7	A still from the tutorial video showing the aeroplane priority seating/lottery ticket analogy.	134
9.1	An example of a participant sketch and the corresponding <i>SuperCollider</i> template code, edited by the facilitator during the sessions to enable participants to hear the result of their sketched Chooser. Note that the whiteboard pen colours do not have special meaning; the Chooser was drawn collaboratively by two participants, each using a different pen colour.	140

9.2	Scenarios 1–4 from the second user study.	142
9.3	Scenarios 5 and 6 from the second user study, plus the final questions.	143
9.4	The simple sequence example shown to participants in scenario 1 as part of the second user study.	143
9.5	A participant creating a sequence on a whiteboard.	144
9.6	Two participants working collaboratively on a sequence.	144
9.7	An image of a Soundable Chooser, used as part of scenario 2 in the second user study.	145
9.8	The example used in scenario 3 of the second user study, showing the use of infinite weight.	146
9.9	A still from the tutorial video showing the aeroplane priority seating/lottery ticket analogy. This image is a duplicate of Figure 8.7 and is presented here for the reader's convenience.	147
9.10	The Full Chooser used in scenario 4 of the second user study, introducing Time Choosers and both hard and soft stops.	148
9.11	A still from the tutorial video showing a visualisation of hard and soft stop behaviour.	149
9.12	A participant working on a Full Chooser design using a whiteboard.	150
9.13	The example used in scenario 5 of the second user study, showing infinity in the nose cone of the Soundable Chooser, the user of a multi-lane Time Chooser, and soundable content in a Time lane.	150
10.1	An example of the final design of Choosers v3, showing the use of an integer inside the loop icon to specify the number of times the material in a lane should be played, a blank lane to allow for a musical rest for the Time Chooser's selected duration (in this case, 8 beats), and updated hard (top) and soft (bottom) stop icons.	160
10.2	Finite looping control in Choosers v3. The uppermost Chooser shows 'loop off' in the first lane and 'play once' in the second lane; these are functionally identical. The second Chooser shows 'loop on' in the first lane and 'loop infinitely' in the second lane; these too are functionally identical. The third Chooser shows examples of both 'zero weight' (first lane) and 'zero loops' (second lane). The first lane's zero weight means that the lane cannot be selected for playback. While the second lane's positive weight means that it can be selected, the zero loop setting means that the lane contents will not play and the lane will act as a blank lane (to be introduced in Section 10.3.1). Finally, the lowermost Chooser shows an example of the new functionality that the addition allows; if selected, the top lane will play twice before stopping, and the bottom lane will play four times before stopping.	161
10.3	Three alternative visualisations showing the musical duration of each lane in a Soundable Chooser.	162
10.4	Global controls for tempo and time signature.	162
10.5	An alternative visualisation to that shown in Figure 10.3: here the user can choose the unit used to display duration, and can opt to play the entire sample ('percussion'). This design is a development of the panel shown in Figure D.36 of Appendix D.	163
10.6	A single-lane Soundable Chooser with the output of a Variable Chooser used as the nose cone value. This will result in the melody1.wav sample being played if the output of variable var is a 1. If var outputs a 0 then the sample will not play. Note that the nose cone value of a Variable Chooser is always 1 and cannot be changed (see Section 6.1.7 in Chapter 6).	164
10.7	An alternative to Figure 10.6, this time using a blank lane. There is an equal chance of the selection of either lane. The upper lane will result in playback; the lower lane will result in no output and, if used as part of a sequence, the Chooser would be skipped.	164

10.8	This example is identical to Figure 10.7 with the addition of repeats via the ‘x8’ multiplier. This will result in 8 repeats, each of which will result in an equal likelihood of either playback or no playback.	164
10.9	A Full Chooser with a blank lane in the Soundable Chooser.	165
10.10	A Full Chooser with a blank lane in the Time Chooser.	165
10.11	A Full Chooser with blank lanes in both Soundable and Time Choosers.	165
10.12	The musical output resulting from the selection of blank lanes in Soundable and/or Time Choosers.	166
10.13	The four short piano phrases used in the piano and violin piece shown in Figure 10.14 and Figure 10.15. Note that the lefthand phrases are three beats long, and the righthand phrases are four beats long. The righthand phrase starts halfway through a note, and is syncopated when compared to the lefthand phrases.	167
10.14	Piano and violin example 1. 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. Audio of this example is available online (Bellingham, 2020a).	167
10.15	Piano and violin example 2, which takes the piano and violin example shown in Figure 10.14 and adds Time Choosers 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. Audio of this example is available online (Bellingham, 2020a).	168
10.16	Piano and violin example 3. This is the same as the example in Figure 10.15 but with the duration of lefthand increased from 3 to 3.5 beats. Given that the audio samples in the ‘lefthand’ Chooser are both 3 beats long, this introduces a half-beat rest at the end of each phrase which creates an alternating syncopated/synchronised pattern when played alongside the 4-beat ‘righthand’ Chooser patterns. Audio of this example is available online (Bellingham, 2020a).	169
10.17	Piano and violin example 4. This replaces the 3.5 beat duration of the ‘lefthand’ Chooser with a duration of 3.05 beats, creating a slower phase shift between the ‘lefthand’ and ‘righthand’ patterns. Audio of this example is available online (Bellingham, 2020a).	170
10.18	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. Audio of this example, named ‘Drum machine example.wav’, is available online (Bellingham, 2020a).	171
11.1	An example of a pre-prepared musical example. The figure shows the complete Chooser diagram as shown to the participants (top). The section shaded in grey corresponds to the <i>SuperCollider</i> template code (bottom), edited by the facilitator in response to modifications requested by the interviewees.	177
11.2	Example 1—rock music, showing the use of sequencing and named Choosers and offering significant scope for change via the manipulation of lane weights and nose cone values.	178
11.3	Example 2, version 1—piano and violin piece, demonstrating nested Choosers and the interplay between looping parent and child Chooser lanes. This example was described in detail in Section 10.4.1 in Chapter 10 and is duplicated here for the reader’s convenience.	178

11.4	Example 2, version 2—this enhances the piece shown in Figure 11.3 by adding Time Choosers to the lefthand, righthand, and violin Soundable Choosers to create Full Choosers. This change allows for the durations of each Chooser to be overridden, offering access to syncopation, synchronisation, and phase music. The violin Chooser also contains a blank lane which will play silently for the duration of the Chooser if selected. As in Figure 11.3, this example was described in detail in Section 10.4.1 in Chapter 10, and is duplicated here for the reader's convenience. . . .	179
11.5	Example 3—drum machine, electric piano, and glockenspiel. This example is more complex than the previous pieces, and includes double-nested Choosers and the use of beat-duration (rather than bar-duration) Time Chooser lanes. The nesting hierarchy has been visualised in columns and rows for the reader's benefit. This example was described in Section 10.4.2 in Chapter 10 and is duplicated in this chapter for the reader's convenience.	180
11.6	A coda symbol (left), compared to the hard stop (centre) and soft stop (right) icons.	183
11.7	The Akai MPC60, released in 1988 (Exarchos, 2019).	185
11.8	Ableton <i>Live</i> (Ableton, 2021).	185
A.1	Three separate samples shown in the main window: each is a single-lane Chooser.	217
A.2	Two single-lane Choosers—one containing a sample on the left, and one containing the name of a Chooser on the right.	217
A.3	This Chooser contains playable media; in this instance, samples. As such it is a <i>Soundable Chooser</i> and has a nose cone which slopes downwards.	218
A.4	Anatomy of a Chooser.	218
A.5	One lane to be selected.	218
A.6	One lane to be selected, lane 1 has a weight of 2.	219
A.7	Two lanes to play concurrently.	219
A.8	All three lanes will be selected and the weight of the lanes will not be a factor. . . .	219
A.9	One lane with a weight of zero, meaning that the lane's contents will not be selected.	220
A.10	A Chooser with zero in the nose cone. This means the entire Chooser will not run.	220
A.11	A nested Chooser.	220
A.12	A time lane.	221
A.13	A Time Chooser.	221
A.14	A Soundable Chooser containing three samples and a Time Chooser containing a time lane with a duration of eight bars.	221
A.15	In this example one of three samples will be selected for playback. The samples will not loop, meaning that the chosen sample will play once and then stop. The time lane is set to a soft stop (the > symbol in the status column). The Chooser will run for an 8 bar duration, and if it is still playing after 8 bars the sample will be allowed to finish before the Chooser is released.	222
A.16	In this example the chosen sample will loop until the duration of the time lane has completed. The time lane is set to a hard stop (the X icon). Once the duration is complete the sample that is playing will be stopped immediately.	222
A.17	The Chooser will play for 8 bars, after which the playing sample (if it has not already finished) will be stopped using a hard stop.	222
A.18	The time nose cone has been set to zero, meaning that the Soundable Chooser will run as though the Time Chooser is not there. One sample will play for its length and the Chooser will be released.	223
A.19	The Soundable Chooser's nose cone is set to zero, meaning that none of the samples will be selected. The Time Chooser's nose cone is set to 1, meaning that the duration shown in the time lane will run. This Chooser therefore runs silently for 8 bars, creating an 8 bar rest.	223

A.20	In this example both Soundable and Time Choosers have their nose cones set to zero. This means that the entire Chooser is skipped.	223
A.21	Making Melody 3 the time. Either Melody1.wav or Melody2.wav will be selected to play, and will loop until the duration of Melody3.wav has elapsed.	224
A.22	This is the same as the previous example except that Melody3.wav can also be selected by the Soundable Chooser.	224
A.23	A lane containing an external input message in the Time Chooser. The status column in the Soundable Chooser shows that the lanes containing samples will loop until stopped. Once the external input is received the Chooser will stop. Note that the time lane is set to a hard stop, and so the currently playing sample will be stopped immediately.	224
A.24	A single-lane Chooser containing an external input will start the playback of a subsequent Chooser once the external input is received.	225
A.25	In this example there are two lanes in the Time Chooser. If the uppermost lane is selected the Soundable Chooser will run for 8 bars and the sample will be allowed to complete before the Chooser is released (a soft stop). If the lower lane is selected the Soundable Chooser will run for 16 bars before being immediately stopped (a hard stop).	225
A.26	A variable called myLength called inside the Chooser myMelody.	226
A.27	A musical section showing that all three playable lanes will be selected for playback. The section has an associated Time Chooser meaning that the section will be exactly 16 bars long.	226
A.28	Arrow showing sequence.	226
A.29	Macro arrangement.	227
A.30	The macro arrangement with duration information for each section.	227
A.31	A new Chooser, Joe Verse, which is used to populate the existing arrangement.	228
A.32	The myDrums Chooser, which populates the myDrums lane of the Joe Verse Chooser.	228
A.33	A musical section, named 'Idea 1'.	229
A.34	An arrangement that creates a sequence by referencing Choosers.	229
A.35	A variable Chooser called xyz.	230
A.36	The variable 'xyz' is used to control the number of repeats of two Choosers. The variable selection is independent.	230
A.37	Both references to the variable Chooser add '@2' to the end of the name, showing that the same selection should be used for both.	230
A.38	This example is functionally identical to the previous one. This time the user has chosen to give a more meaningful name to the variable Chooser.	231
A.39	Two Choosers are created. They are not yet connected.	231
A.40	A Chooser named 'Var' which will select one of two integers.	231
A.41	A Chooser named 'Top' in which the 'Var', 'myMelody' and 'myLength' Choosers have been nested.	232
A.42	The creation of Choosers named 'myChords' and 'HarmProg'.	232
A.43	'myMelody' and 'HarmProg' are nested inside 'Drift'.	232
A.44	The variable Chooser xyz.	233
A.45	A multi-lane variable Chooser with descriptive column names.	233
A.46	A multi-lane variable Chooser with non-descriptive column names.	233
A.47	In this example the variable is called 'throw' and the columns are named 'a' and 'b'. Therefore, the possible options are 'throw:a' and 'throw:b'. Note that each reference to the 'throw' variable is independent, and represents a new 'throw of the die'.	234
A.48	This is a functionally identical example which uses the more descriptive column names. The format remains the same.	234

A.49	Example using non-descriptive column names and an ID. Note that the result of the ‘throw’ variable will result in a duration of 8 bars <i>and</i> a hard stop <i>or</i> a duration of 16 bars <i>and</i> a soft stop.	235
A.50	Example using descriptive column names and an ID.	235
C.1	The simple sequence example shown to participants in scenario 1 as part of the second user study.	240
C.2	The result of scenario 1, successfully completed on a whiteboard.	241
C.3	An image of a Soundable Chooser, used as part of scenario 2 in the second user study.	242
C.4	Scenario 2 being completed. In this instance, P2 is drawing and P1 is making suggestions.	245
C.5	The example used in scenario 3 of the second user study, showing the use of infinite weight.	245
C.6	The result of scenario 3, collaboratively written by both participants.	247
C.7	The Full Chooser used in scenario 4 of the second user study, introducing Time Choosers and both hard and soft stops.	248
C.8	Scenario 4, with changes being made by P2.	251
C.9	The finished Chooser for scenario 4.	251
C.10	The example used in scenario 5 of the second user study, showing infinity in the nose cone of the Soundable Chooser, the user of a multi-lane Time Chooser, and soundable content in a Time lane.	252
C.11	The result of scenario 5.	256
D.1	A paper sketch showing a branching musical arrangement on the left, audio files on the upper right, and a list of possible actions on the lower right. The Intro section is currently selected, and the audio files that are assigned to that selection are highlighted. One file will be selected per colour, with duration shown in bars (L : 16 means ‘16 bars’) and weighting of each choice shown as a percentage. The Action section in the lower right shows the current available actions given the selection of the Intro section.	260
D.2	In this sketch, the section named ‘Drifting’ has a duration of between 64 and 256 bars. It has been selected alongside the number 8 (upper right), meaning that the Action menu now shows all the actions which combine the section and the number.	261
D.3	This example shows the audio files assigned to the musical section named ‘Drifting’. As with Figure D.1, colours are used to group files together; in this instance, one of Drums1.wav, Drums2.wav, or Silence will be selected.	261
D.4	This example tested a user-controlled layout; the user introduces as many widgets (called depots) as required. In this way the user can create multiple widgets of the same type, such as multiple number widgets, to further constrain selection. Note the down arrows to the right of menus, which are a filtering system.	262
D.5	This is an implementation of the DC design in <i>SuperCollider</i> (McCartney, 2002).	262
D.6	A steel weight to be used as a metaphor for lane weight.	264
D.7	An early icon test for Choosers v2, including a trapezium to represent the steel weight shown in Figure D.6. This is not necessarily a helpful image as it suggests physical weight rather than likelihood. A circular loop icon is used for repeats (see Section D.2.1.2). Hard and soft stops are shown using a play triangle either running through a line (soft stop will continue to play) or being bisected by the line (hard stop will stop immediately)—see Section D.2.1.3.	264
D.8	A variant of the previous design (see Figure D.7) with more abstracted icons. This uses basic geometric shapes, resulting in relative simplicity while retaining the power of the metaphors. The icons are large for reasons of legibility; tests with smaller abstract icons are shown in Figures D.22, D.23.	265

- D.9 An example showing a circular ‘repeat’ icon in the Soundable Chooser. Note that this example tested the use of a ‘repeat’ icon in the time lane, providing consistency while allowing for the duration to be easily changed. This design was rejected as it did not offer additional clarity or functionality when compared with simply changing the duration in the Time Chooser lane. 265
- D.10 Icons for looping behaviour in popular music playback applications. The upper row shows a disengaged loop (YouTube Music, iTunes, Spotify). The lower row shows the icon in the same applications denoting a single song loop. Note the use of a number, as in the prototype design for repeats. The tension here is between *repeats* as a musician would understand them (x1 means ‘play once’) and a *loop* (‘loop once’ means ‘play twice’). The number ‘1’ in the icons on the lower row refers not to the number of repeats or loops, but instead shows that a single song will loop infinitely. Framing, and the expectations it brings, would suggest that a different icon should be used, as the functionality is different. 266
- D.11 A possible notation change to multiplication rather than the circular icon; this sketch shows a Chooser in which the drums.wav audio file will play once and the bass.wav file will play twice. The intention is to avoid the danger of a ‘1’ inside a loop symbol being misinterpreted as meaning ‘play once, and then loop once’. Instead, it was hoped that ‘x1’ might be more clearly understood unambiguously to mean ‘play one time’ and less likely to be misunderstood as meaning ‘play one time, then loop 1 time’. The ‘repeat’ column is not used in the Time Chooser as the duration can be changed by using multiplication notation directly in the time lane, as shown in Figure D.12. 266
- D.12 This figure shows two examples of ‘atomic repeats’, which uses multiplication notation within sample or duration boxes. The drums.wav sample is being repeated twice. This ‘glues’ two copies of the sample together to create a new atomic unit. This atomic unit is then played once (the $\times 1$ in the repeat column). The Time Chooser lane also includes an ‘atomic repeat’ which results in a new duration of 12 bars (4 bars $\times 3$). This is equivalent to changing the duration to 12 bars and is offered as an alternative notation. Note the difference between the two soundable lanes—the drums.wav sample will play twice regardless of when it is stopped by the time lane, whereas the bass.wav sample may only play once if it is longer than 12 bars, or play twice if it is shorter than 12 bars. If the duration was to be changed using multiplication notation it would be done directly in the time lane, as this would also create a new atomic unit. 266
- D.13 Following on from Figure D.12, this example explores a possible minimum/maximum repeat column. Here the drums.wav sample plays for a minimum of 1 and a maximum of 2 times. The bass.wav sample plays at least once, and has an infinite maximum number of repeats. This design provides bounds but does not change the atomic unit. The suggested design in Figure D.12 appears to offer more musically meaningful options. 267
- D.14 The refreshed icons used in the second user study: loop and non-loop; hard and soft stop; weight of 1 and infinite weight; soundable file in a Time Chooser play and mute. 267
- D.15 Following a simple scenario as part of an inspection by programming walkthrough, the user drags a single sample into the play area—this creates a box and adds a repeat control. 268
- D.16 The user snaps on a second lane to the bottom of the first sample box. This creates a Soundable Chooser, with a nose cone on the left, and a weight column on the far right. 268

D.17	Time Chooser test to explore the trade-offs in sharing a consistent design between Soundable and Time Choosers; this is a Time Chooser version of the Soundable Chooser in Figure D.15. Note the lack of a nose cone.	268
D.18	The same design as in Figure D.18 but with a nose cone which clearly differentiates this Time Chooser from a Soundable Chooser. It also allows for the Chooser to be quickly skipped by setting the nose cone to zero.	269
D.19	A Full Chooser. Note the new icons, added stop controls, and the lack of weight and mute for the time chooser.	269
D.20	A Full Chooser with nondeterministic duration, necessitating weights in the Time Chooser.	269
D.21	A Full Chooser with a sample in one of the Time Chooser lanes, requiring the mute control. Note the lack of symmetry caused by the lack of a column in the uppermost time lane.	269
D.22	A series of candidate icons for the management of soundable content in Time Chooser lanes. This series of images shows examples of greyed-out speakers to denote no audible playback of the contents of the time lane. Durations do not contain any audible material. The melody.wav example in the bottom row can be made audible or not audible.	270
D.23	A continuation of the candidate icons for the management of soundable content in Time Chooser lanes shown in Figure D.22. These tests use muted/unmuted speaker icons of the type used in operating systems such as iOS, MacOS, and Windows. . .	271
D.24	Example popup controls associated with each lane in a Soundable Chooser. Volume and pan are shown: additional options include mixing desk-style controls such as filters and reverb level. The trim control allows the user to set the start and end point of the lane contents. The snap control is shown in a box to the right of the trim control—this sets the number of ‘slices’ that the faders will snap to. This can be clicked to disable the snap for freehand control.	272
D.25	Candidate stop icons from the process outlined in Section 10.2.2. The final design adopted in Choosers v3 is shown on the lower right.	272
D.26	An alternative lane weight icon which uses progressive shading to denote relative weights, with ‘zero weight’ shown on the far left and ‘infinite weight’ on the far right.	273
D.27	A sketch showing an alternative to sequencing via arrows; a ‘big arrow’ in which Choosers can be placed to be played sequentially from left to right.	273
D.28	Pulse Choosers v1: three equivalent sketches of a Variable Chooser (upper lane) being repeatedly triggered by a looping Time Chooser (bottom lane). The three Choosers demonstrate alternative equivalent notations.	274
D.29	A sketch showing a Full Chooser being ‘pulled apart’ to show the type of message sent from the Time Chooser to the Soundable Chooser.	275
D.30	A generalisation of the ‘pulled apart’ Chooser which will form the basis of Pulse Choosers v2.	276
D.31	A Pulse Chooser v2 sketch. Each time the looping Time Chooser fires the operation layer will produce a random number (in this case, a random number between 0 and 7) which is then used to set the anchor point. Note the infinite loop setting in the Time Chooser lane, meaning that the Time Chooser will repeatedly trigger the selection of a new random number every 1/8th of a bar.	276
D.32	A modified version of Steve Reich’s <i>Clapping Music</i> (1980) using the Pulse Chooser v2 design. The anchor point of clap2.wav is incremented by 1 each time the Time Chooser’s duration elapses. A new duration is selected each time. Reich’s original piece can be played by replacing the random duration with a fixed value (e.g. 8 bars).	277

- D.33 Rock example using speculative design. As shown, the Soundable Chooser (left) will play all five lanes due to the use of ∞ in the nose cone; all lanes are selectable as their lane weights are greater than 0. The user can optionally use the Soundable Chooser's lane weights to remove ($=0$) and add (>0) lanes from selection; audio of this figure, demonstrating an example in which the user controls lane weights to add and remove samples to create an improvised musical arrangement, is available online (Bellingham, 2020a) – Rock example using speculative design.wav. The anchor points of four of the five Soundable Chooser lanes are programmatically controlled by the Pulse Choosers on the right of the figure. Each Pulse Chooser is rhythmically triggered by a looping Time Chooser; upon firing, a random number will be selected and used to set the anchor point of one of the Soundable Chooser lanes. While capable of surprisingly rich output, the use of multiple UI widgets leads to hidden dependencies and various legibility issues (e.g. the high number of grid points in the drums.wav 2 Pulse Chooser on the top right of the figure). . . . 278
- D.34 Using a Variable Chooser to populate the operation layer of a Pulse Chooser. . . . 279
- D.35 An example showing the use of drawers to access parameters for textual control, and an example of a referenced Variable Chooser—vv23. The Variable Chooser will be repeatedly triggered on every beat and will select one of two numbers (2 or 4). This number is then used to set the anchor point. 280
- D.36 Steve Reich's *Clapping Music* (1980) implemented using manual control. The user can move the anchor point of clap2.wav to a grid point, set using the control to the right of the trim slider. In this instance the user can move the controls to one of 12 grid points. 281
- D.37 A development of Figure D.36 with a Variable Chooser (myV) used to control the anchor point of clap2.wav. myV will be triggered 12 times, with an interval between 4 and 32 bars chosen afresh. Each triggering will increment the variable value by 1. 281
- D.38 A sketch exploring the use of the draft textual language to increment the anchor point of clap2.wav. The textual statement in the lower lane of the Chooser defines myAnchor as incrementing by 1 on each repeat. The variable myAnchor is then used in the trim control as the fourth setting (grid, start point, end point, anchor). . . . 282

List of Tables

3.1	Potential risks that may arise when a designer takes a role in administering walk-throughs or undertaking analysis, and the steps taken to mitigate them.	37
6.1	Mapping user requirements, findings from the CDN review (Chapter 4), and the resulting feature in Choosers.	107
6.2	Mapping user requirements, findings from the DC design exercises (Section 5.2.1), and the resulting feature in Choosers.	108
6.3	Mapping user requirements, findings from the PN design exercises (Section 5.2.2), and the resulting feature in Choosers.	109
8.1	Issues in the design of Choosers v1 and supporting tutorial materials, identified in the first user study (see Section 7.5 of Chapter 7), mapped to proposed changes in the design of Choosers v2 and supporting materials.	136

Previously published papers

Sections of this thesis have been previously published in the following papers:

Chapter 4

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://users.sussex.ac.uk/~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>.

Chapter 6

Bellingham, M., Holland, S. and Mulholland, P. (2016) Designing a Highly Expressive Algorithmic Music Composition System for Non-Programmers, In *DMRN+11: Digital Music Research Network One-Day Workshop 2016*, Digital Music Research Network One-Day Workshop, [online] Available from: <http://oro.open.ac.uk/52732/>.

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 from: <http://hdl.handle.net/2436/621151>.

Chapter 7

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

Chapter 10

Bellingham, M., Holland, S. and Mulholland, P. (2019) Toward meaningful algorithmic music-making for non-programmers, In *Proceedings of the 30th Annual Workshop of the Psychology of Programming Interest Group—PPIG 2019*, Church, L., Mroiu, M. and Marshall, L. (eds.), [online] Available from: <http://hdl.handle.net/2436/623571>.

Note regarding the code used in this thesis

I wrote the original code used in the development of Choosers. One of my supervisors refactored the code to make it more convenient to quickly encode diagrams when running user studies.

Chapter 1

Introduction

This chapter outlines my initial motivation, introduces relevant terminology, lists the research questions, and presents an overview of the thesis.

This thesis will present the design and evaluation of Choosers, a visual programming language designed to allow structural elements of the kind involved in nondeterministic music composition to be readily visualised and manipulated, while making little or no demand on programming ability. The language centres around a novel non-standard programming abstraction (the Chooser) which controls indeterminism, parallelism, choice, multi-choice, recursion, weighting, and looping.

1.1 Personal motivation

While working as a producer on an album project, I found myself recording several takes while the musicians rehearsed, discussed, and changed the material they were to record. My role, as producer, was to act as arbiter; which take was the best? Motivated by my background as a musician, and my profession as an educator, I began to consider processes for identifying a range of aesthetic options, rather than just one.

I considered the impacts of recording on music, on music creators, and on listeners. I was struck by Evan Eisenberg's (2005) comparison of a music recording to a video recording of a stage play, in which he highlighted the distinctions between a transitory shared event and a carefully constructed recording which offered a platonic ideal performance. Phillip Auslander (1999) similarly compared the live performance of a play or a piece of music to the recorded and mediated versions in film or audio recording. In both cases the authors reflected on the differences between a live performance and a recorded one, and the effects these changes have on the work. Regardless of the relative merits of the two forms, the choice is presented as binary; a live performance is necessarily variable and loaded with the possibility of change, whereas the recorded performance is static and unchanging. When considering the differences between visiting a theatre and a cinema, Auslander states that 'we would not want a projectionist to be *creative* in showing a conventional film' (1999). But, why not?

We may consider a play's script and a musical score as in some ways analogous. Both are a set of instructions, or blueprints, from which a final performance is constructed. Consequently, each performance will be different due to the creative interpretation and human variability of the collective in attendance; performers, directors, conductors, technical staff, and audience. Each performance is necessarily unique. When recorded, the performance becomes reified and repeatable, and all of the variations which used to differentiate the different performances become fixed and part of the overall experience (Zak, 2001). The audience member's relationship with the work is changed by the act of recording.

I became interested in testing a system in which a score-like blueprint could be 'performed' by software, rather than be interpreted by performers. Such a blueprint would allow the playback to

be different each time it is run, and would offer a transient experience similar to a live performance. As a first step, I took a recording familiar to many—*Bohemian Rhapsody* by Queen—and introduced variations in playback to seek feedback from people with a passing familiarity with the original recording. The original multitrack recording of the track contains three vocal takes for the first verse; each take adheres relatively strictly to the melody and lyrics, with only minor performance differences between them. The original mix of the track makes use of just one of the three alternative takes. I split the three vocal takes into eleven sequential phrases, resulting in thirty-three short audio files, and created a simple switching system. In this system, a random number generator selected one of three initial phrases and, once this file had finished playing, one of three alternatives of the second vocal phrase was randomly chosen, and so on. The original piano track was added alongside the vocal to provide musical context to the ‘performance’. I named this test the ‘Creative Projectionist’ after Auslander’s phrase, and used it as the basis for informal discussions with musicians, music producers, and Music Technology students.

Some participants were curious about the experiment but did not regard it as desirable; they felt that their emotional bond with certain recordings was due to familiarity with the material, and were concerned that even minor changes could harm their relationship with the work. However, several people were excited by the creative possibilities of the tools and asked how they could make musical adjustments. Commonly, users wanted to freely replace the audio files, control the sequence in which the files were played, control the number of files being played synchronously, and freely change the number of optional takes and the likelihood of each selection. While simple for a programmer, these changes presented significant challenges to non-programmers as they required understanding of the software, both to locate the relevant section (sometimes several layers deep) and to make the relevant changes. Importantly, all those who interacted with the creative projectionist were not programmers, but instead were musicians and/or music producers. These users did not feel capable of using a programming language to make changes to the patch, and they could not think of a platform which would allow them to create a similar tool without specialist support.

I began to consider the potential role of a low-threshold algorithmic music composition tool which would present a suitable entry point for a non-programmer who is already familiar with DAW software; such a tool would enable these users to create algorithmic music autonomously and without the support of experts. The phrase ‘algorithmic *GarageBand*’ was used in initial discussions with the ‘Creative Projectionist’ participants as I was inspired by *GarageBand*’s provision of a limited and targeted level of functionality, and the wide range of users and use cases that this seemed to enable. While *GarageBand* is accessible for novice users, it is also used by experts when more advanced functionality is not required, or by experts in domains other than music or audio recording. The envisioned new algorithmic music composition tool would facilitate nonlinearity in two senses; it would enable composers to easily select between alternatives as well as allowing for automatic selection between alternatives in principle at any level of structure in the piece. The tool would be related to, but distinct from, live coding software such as *Tidal* (McLean and Wiggins, 2010) as the users of the system are not programmers. The tool would also be related to nonlinear music implemented in some video games, but is separated due to a focus on music composition tools rather than interaction with a game engine. In comparison with live coding and game audio, it was anticipated that algorithmic music composition tool for non-programmers are likely to require a different set of design trade-offs. Once developed, such a tool may have an impact on the work of music producers, music composers, and both educators and students in music computing education.

1.2 Research questions

The research questions are as follows; notes on terminology are given below.

How can nondeterministic composition tools be made accessible to non-programmers?

What methods would support the design and evaluation of nondeterministic composition tools for non-programmers?

What are the implications for music production, music computing education, and music composition?

As outlined in the previous section, the term ‘non-programmer’ is taken to mean a user who may have at least some familiarity with DAW software, but who has very little to no experience of writing algorithms in a programming language.

1.3 Useful terminology

The thesis will use Bruce Jacob’s definition of algorithmic music (1996), which is the application of a well-defined algorithm or algorithms to the process of composing music.

The ‘Creative Projectionist’ experiment, outlined above, prompted a number of interesting questions from those taking part. One frequent topic was in categorisation; is this a performance or a recording, and is it a linear or a nonlinear work? The changes created by switching between vocal takes results in relatively minor phrasing changes, the structure of the piece is unchanged, and a casual listener may not even notice the changes. One might argue, therefore, that the resulting musical structure should be termed linear. A similar argument might be made if one altered the structure in simple ways—by changing the number of repeats of some section, for example. However, in this thesis, both in these simple cases as well as in other cases where structure might be radically varied on different runs—or even repeated exactly when desired—we will consider a system and the outputs it produces to be nonlinear provided there is always the capability to produce a different output each time the patch is run.

The term ‘nondeterminacy’ is used to refer to probabilistic selections made as part of an algorithm. Nondeterminism is a key feature in some algorithmic music systems. As will be discussed in Section 2.6, composers such as John Cage and David Tudor (1959) used the term indeterminacy to refer to changes in one or more elements of a composition or a performance either by chance or by human agency. In contrast, and by the definition used in this thesis, nondeterminate selections are made entirely probabilistically and without human agency.

Using the term ‘nondeterministic composition tools’, this thesis considers the design of tools capable of a subset of nondeterministic music composition, namely high-level structural arrangement and musical form. While the principles implemented in Choosers potentially extend down to note level, the primary focus of the forthcoming chapters is on structural arrangement. Such work could alternatively be referred to as ‘nondeterministic music arrangement’; however, music arrangement is typically thought of as an orchestration process, and it was felt that this term could prove confusing.

1.4 Thesis overview

The thesis is structured as follows:

- Chapter 2** considers the impact of manuscript and phonography on music, and the effects of user expectation on software design. It presents algorithmic music as an area, with a brief review of representative software, and considers relevant design trade-offs.
- Chapter 3** introduces the key methodological tools used in this thesis—Cognitive Dimensions of Notations; Research through Design; programming walkthrough; and the Wizard of Oz technique. The protocols for the walkthroughs and interviews is presented, followed by a list of the limitations of the chosen methodology, and the mitigations applied.
- Chapter 4** presents an analysis, using the Cognitive Dimensions of Notations (CDN), of a representative selection of user interfaces for algorithmic composition software. The findings from this CDN review were used to identify candidate design principles.

- Chapter 5** outlines design exercises used to test two of the design principles selected using the findings of the previous chapter's CDN review. Both Direct Combination and Petri Nets were tested, with findings used to inform the development of this thesis' fundamental abstraction.
- Chapter 6** presents the design and development of the preliminary version of Choosers. The design is outlined, showing how it allows for sequencing, indeterminism, choice, parallelism, multi-select, and looping. The chapter then shows how this design developed from the design exercises outlined in previous chapters. A number of design sketches are shown alongside a consideration of trade-offs and alternative implementations.
- Chapter 7** presents the design of, and findings from, a participant programming walkthrough evaluation of the first design of Choosers. While participants were able to create engaging non-deterministic music, a number of design issues were identified concerning conceptual alignment, problems due to user expectations, user understanding of the soft stop, and the tutorial order.
- Chapter 8** describes the process of developing Choosers v2 given the findings from the first user study. This chapter shows the development of infinity as a maximal setting for weight, duration, and nose cone values; new icons for loop, weight, and stop settings; per-lane stop behaviour; a trim control; an improved tutorial order; and supporting documentation changes.
- Chapter 9** outlines the second user study, using Choosers v2. The design changes to support conceptual alignment seemed to have been successful; auditioning was an important tool for musicians; users understood and valued the infinity options; and supporting tutorial materials had been effective. The findings also showed that one user wanted more granular loop control; some icons were not sufficiently clear; and some participants wanted DAW-style execution visualisations.
- Chapter 10** takes the findings from the second user study and uses them to consider a range of options, trade-offs, and alternatives in the preparation of Choosers v3. A variety of loop and repeat options were tested; new stop icons were developed; a 'blank lane' was introduced as an alternative notation; and musical examples were developed.
- Chapter 11** outlines the third and final user study. This study took the form of interviews with domain experts in the fields of music production, music computing education, and music composition. Findings from each interview are presented and summarised.
- Chapter 12** considers each research question in turn before discussing the limitations of the project and directions for future research.

Chapter 2

Literature review

This chapter discusses literature relevant to the development of an algorithmic music composition tool that aims to make minimal demands on programming ability.

For the vast majority of its history music has been *alive*—evolving and changing with each new performance. The revolutionary capability to record music—phonography—allowed music playback to be divorced from performance for the first time, and led to the unprecedented pollination of genres across continents. The reification of music also created an expectation of fixity among both musicians and listeners, and so the software tools that were eventually developed to create recordings were, and are, focussed on linear playback. As a result, and for the past several decades, musicians, record producers, and listeners have been implicitly encouraged to apply a linear frame of reference onto many genres of music.

This chapter will begin by considering how manuscript shaped music-making, the impact of phonography on music, and the software tools used to create linear recordings. The definition of algorithmic music will be revisited before considering the design trade-offs of a small representative subsection of software capable of algorithmic composition. In order to understand these design trade-offs, expressivity and usability will be used as a ‘lens’ through which to consider the ways in which algorithmic composition software design can meet the needs of users.

Simon Frith (1996) has suggested three broad categories which are relevant to this project: a *folk stage* in which music is ‘stored’ in the body rather than externally; an *art stage* in which music is notated; and a *pop stage*¹ in which recorded music becomes a commodity. This chapter will begin with a consideration of Frith’s first two stages; Frith’s third category, the pop stage, is covered with a consideration of the cultural impacts of recording in Section 2.3.

2.1 Aural tradition

Frith’s folk stage refers to an aural tradition in which music knowledge is transferred via listening to, and participating in, social music-making. While they exist all over the world, such aural traditions tend to be geographically localised due to low mobility, and many cultures use music for both social cohesion and communicative purposes. Music-making can be considered a transfer of meaningful social practice, with replicated musical patterns transmitted between generations of musicians and composers (Jan, 2017). In the aural tradition, musical styles develop locally and are transferred slowly via the movement of people. Such practice inevitably leads to regional variations; if considered as a type of media transfer, the fidelity of the aural tradition is poor as detail will be lost, and long-term storage and retrieval is unreliable.

¹Note that Frith is not referring to popular music as a genre.

Music in Frith's folk stage is created by musicians using instruments and/or the voice; the instruments are mediating tools which form a core part of the creative process (Duignan et al., 2010). Marc Leman defines the immediate aural feedback that instruments give musicians to be part of an important feedback loop as the Action-Reaction cycle (2008); a musician can play a note, hear the result of their performance decisions, make a judgement based on what they can hear, and change their performance accordingly before playing another note. Leman notes that this feedback cycle gives the musicians a sense of control, and encourages immersion in their musical undertaking.

In considering a similar issue in a different domain, Tanimoto's concept of 'liveness' in computer systems (1990) concerns the feedback available when an end user edits the notation of a piece of software. Tanimoto proposes four levels of liveness, with the highest level providing feedback which is informative, significant, responsive, and continually active (Birchman and Tanimoto, 1992). As musical instruments offer immediate aural and haptic feedback they can be categorised as exhibiting this highest level of liveness. A high degree of liveness allows musicians to quickly test small ideas as part of the early, exploratory stages of creativity (Nash and Blackwell, 2014; Blackwell and Green, 2000; Duignan et al., 2004; Smith et al., 2009); tools which allow for quick auditioning of ideas can facilitate creativity via greater ideation (Sternberg, 1999).

2.2 Common music notation

Common music notation emerged in the late 9th Century (Apel, 1961) and, as with instruments, it is an intermediary tool which can be used in the musical process (Duignan et al., 2010). The fundamental idea behind common music notation is to represent pitch on the vertical axis (rising pitches are represented as vertically rising notes on the staff) and duration on the horizontal axis. Early music notation was used to notate modal music, with later developments requiring the development of key signatures to denote tonal centres. As common music notation was changed to reflect the requirements of composers and musicians it came to embody the biases and tendencies of these practitioners (Csikszentmihalyi, 2014). For example, a staff encourages the stacking of thirds in a given key, facilitating the creation of tonal music.

Frith's *art stage* refers to the music that was made once common music notation allowed music to travel independently of the composer or performer. For the first time a musician could read and play music that they had not heard before—they retrieve the music from the page (Taruskin, 1995). However, the fidelity of common music notation is not complete, meaning that each musician needs to interpret the intentions of the composer with regard to phrasing, timing, timbre, and other performance parameters (Taruskin, 1995). Indeed, music cannot be played by a human being *without* creative interpretation. This means that each performance is interpretative and unique.

Marc Leman (2008) suggests that notation may introduce a harmful abstraction layer between musician/composer and music. He argues that notation, as a visual layer which does not offer sonic or haptic feedback, degrades the user's interaction with the music they are creating. Nash and Blackwell (2012) argue that, while notation has a lower level of liveness—as defined by Tanimoto (1990)—compared to music-making with instruments as intermediary tools, composers use common music notation for a range of tasks including managing complexity, sketching ideas, and abstracting details.

It is useful to consider what constitutes a piece of music: *what*, and *where*, is the work? Richard Taruskin (1995) outlines three differing approaches to the primacy of the notated work in a performance. In the first, the score is considered to be the work as it represents a plan for the performance. In the second, the performance is a single instance of the work. In the third, the work is a 'mental construct'—a platonic ideal—which performances can hint at, but can never fully represent. Philip Auslander (1999) refers to a score as a 'blueprint' for a performance, and Evan Eisenberg (2005) considers the manuscript to be a set of instructions which are used to create the work. He compares music to works such as books or paintings which take the form of static objects, and observes that the Romantics viewed performance variability as music's most valuable trait—'fixity is death'

(2005). The relative merits of precisely repeatable music playback will be considered in Section 2.3.

Throughout the twentieth century those composers and musicians who wished to break free of the conventions embedded in common music notation developed tools to circumvent the biases inherent in the stave. For example, Arnold Schoenberg's serialist system gave equal weighting to all twelve notes of the chromatic scale and prescribed the order in which they should be used, avoiding the harmonic and melodic patterns of the past (Ball, 2011). A similar motivation led to the use of graphic scores as a development from John Cage's experiments with indeterminacy in the 1950s (Fetterman, 2012). Graphic scores allow the composer to denote specific musical actions which fall outside of common notation, including specific timbral instructions and directions for improvisation and interpretation.

Music-making, in common with other forms of art, can focus on the *process* of creation or on the *product* that is created. A focus on the process places greater emphasis on the composition and/or performance, often giving the musicians space for interpretation. As a result, the outcomes of the process are typically different (to a greater or lesser extent) each time the piece is performed.

In contrast, a focus on the product requires that the output of the performance is consistent with the intentions of an arbiter, who is often the composer of the piece. Milton Babbitt felt that performers should not be at liberty to change any aspect of his work (Frith, 1996). Edgard Varèse was frustrated that his music was 'altered by interpretation' and wished to have a direct and unmediated transmission to his listeners, such as the relationship between author and reader (Eisenberg, 2005). As will be shown in the following section, phonography offered a good match for product-focussed practitioners. Music with a process focus would either have to conform to the limitations of recording (such as jazz) or work outside of the linear confines of phonography.

Algorithmic music, as one approach to process-led music, will be considered in Section 2.6. The next section will explore the cultural impact of recordings, and the software tools used to make them.

2.3 Phonography

The development of the recording of both light and sound—the photograph and the phonograph—both took place in the nineteenth century (Johnson, 2015). Michael Chanan (1995) refers to Thomas Edison's intention that a user would 'phonograph a sound' in the same way that a photographer takes a picture. This resulted, using Evan Eisenberg's phrase, in the 'reification of music' (2005). As Mark Katz (2004) argues, music now became a precisely repeatable object rather than a unique event. As will now be considered, some genres treat recordings and live performances as two distinct works, whereas other genres consider them to be the same. Auslander (1999) argues that rock music treats recording and performance as one work. Similarly, Gracyk (1996) states that, in rock music, the recording is the primary object and that live performances are a secondary object in which the aim is the recreation of the recording. Other genres do not use the recording as the endpoint which completes the artistic process; Horn (2000) considers jazz as an example where the material continues to evolve after recording, although there can be conflict as the live and recorded works bifurcate.

As with common music notation, the recording process is not neutral; the tools and processes used embody biases and tendencies which influence the resulting work. Several genres of music developed as a result of the opportunities and limitations of the recording process; for example, *musique concrète* was the first music to explicitly take advantage of the medium (Chanan, 1995). The sound of popular music recordings started to change when producers such as Phil Spector rejected the perceived need for an artist's live performance to sound like their recordings; Eisenberg (2005) refers to Spector's work as the 'first fully self-conscious phonography in the popular field'. Both Gracyk (1996) and Auslander (1999) argue that rock music is a recording art, not a performing art, with the identity of the recording stemming from both the production and performance of the work (Horn, 2000).

The work of pianist Glenn Gould is of interest in this context. Gould felt that phonography was a new art, distinct from live performance, and that it required performers to approach it in a different way. Eisenberg (2005) draws an analogy with the distinctions between a fixed linear film and an evolving theatrical performance. Music changes when it is fixed; Chanan (1995) argues that familiarity with a recording destroys the aura of the work as the spontaneity of the performance reduces with repeated plays, and Cone (1968) speculated that music that does not change would become ‘intolerable’ to the listener. Middleton (2000) describes records as an ‘instrument of performance’ and it can be argued that recording decisions are just as interpretive as any other type of performance (Frith, 1996); the recording engineer becomes a new type of creator (Chanan, 1995) who is able to edit new performances into existence (Zak, 2001).

Section 2.6 will return to the requirements of non-predetermined music. The next section will consider the tools used to create linear recordings.

2.4 Trackers, sequencers, and DAWs

Since recording became technically achievable there have been a number of developments in record production, starting with a shift from acoustical and mechanical systems to electrical recording onto magnetic tape (Rumsey and McCormick, 2009). Professional digital audio recording became financially viable in the 1980s, and the Digital Audio Workstation (or DAW) has become the primary audio recording tool for both professional and amateur recordings since its development from the 1990s into the 2000s (Izhaki, 2012).

Both the functionality and presentation of music production software have a significant effect on the ways in which they are used. Csikszentmihalyi (2014) outlines the processes by which a user takes their lead from the domain in which they are working, including the requirement that they need to be conversant in the norms and systems of notation which are commonly used.

2.4.1 Trackers

Soundtrackers, commonly called trackers, allow the user to create patterns of notes. The tabular layout typically used in trackers is visually reminiscent of a spreadsheet. Time is normally represented vertically, scrolling from top to bottom. In most tracker designs, columns represent tracks or voices, and rows represent user-controllable subdivisions of a bar (e.g. semiquavers). Each cell contains numeric data which represents a note (e.g. pitch, instrument, volume) and other control data (e.g. pan position, fades, or glides). It is common for trackers to allow efficient expert input using the QWERTY keyboard, leading to comparisons with programmers text editors such as Vim or Emacs (Collins, 2007). Such expert control led to the capability for virtuosic use (Nash and Blackwell, 2011).

Liveness, as defined by Tanimoto (1990) and introduced in Section 2.2, refers to the feedback available when the user makes a change to the notation, and the Action-Reaction cycle (Leman, 2008) considers the positive impact of immediate feedback on the creative musical act. Nash and Blackwell (2014) relate both concepts to trackers, showing that expert tracker users employ the Action-Reaction cycle by frequently auditioning changes via the playback of short sections of their work. They argue that trackers therefore have a high degree of liveness which supports the user’s creative choices.

2.4.2 MIDI sequencers

MIDI sequencers developed in parallel with trackers. While trackers have a tabular text-oriented interface with a top-down vertical representation of time, sequencers such as *Cubase* (Steinberg Media Technologies GmbH, 2021) typically use a GUI with a left-to-right horizontal representation of time. MIDI sequencers commonly use a central view—the arrange window or page—in which

users can organise and manage individual sequences, which take the form of visual blocks. The contents of these sequences can be created and edited using a number of different views, such as a piano roll, a score editor, or a text-oriented edit view. As a result of the use of various editing operations, sequencers can be seen as a container of connected but perceptually separate devices (Duignan et al., 2004). MIDI sequencers were widely adopted in music production while trackers became part of a niche expert user subculture (Nash and Blackwell, 2011). The use of a number of different graphical tools and a horizontal representation of time became the dominant design of most professional music creation software.

2.4.3 Digital Audio Workstations

Developments in computer hardware throughout the 1990s and 2000s led to the addition of audio recording and editing capabilities into multitrack MIDI sequencers such as Steinberg's *Cubase* (Steinberg Media Technologies GmbH, 2021) and *Logic* by Emagic (later owned by Apple Inc., 2021). Concurrently, the same computer hardware improvements allowed for the development of dedicated digital audio workstation (DAW) software such as *Pro Tools* (Avid Technology, Inc, 2021), which offered multitrack audio recording and editing capabilities. Over time, the feature sets of ex-sequencers and dedicated DAWs have aligned, and the majority of DAWs have mature audio and MIDI functionality.

Most DAW software is designed for linear playback, with each musical element placed in an absolute temporal location (Duignan et al., 2005). A timeline, typically presented from left to right, tends to promote the assumption that music should be built by adding music events to a temporal order (Mooney, 2011), resulting in an accumulative compositional process (Spicer, 2004)—such a system encourages the composer to think of their work as a ‘collection of objects’ rather than a ‘stream’ (Zagorski-Thomas, 2014). Software which shows music in the linear order in which it will be heard exhibits what Duignan (2008) refers to as ‘eager linearisation’. In contrast, software which has a focus on loops or patterns, and which offers flexibility and provisionality in the ordering of these sections, is considered by Duignan to be using ‘delayed linearisation’ (2008).

Section 2.2 introduced the Action-Reaction cycle (Leman, 2008) and liveness (Tanimoto, 1990), and considered how musicians benefit from immediate aural feedback. The importance of liveness and feedback is considered by Duignan et al. (2010), who argue that modern DAWs typically create a clear division or barrier between live performance and composition, even if musicians and composers do not benefit from such a distinction. They argue that DAWs typically do not allow for ‘jamming’ as they have a lower level of liveness than tools designed for performance.

An example of the Action-Reaction cycle and liveness in a different domain is Rebecca Fiebrink's *Wekinator* (2015), an interactive machine learning (IML) tool which allows users to build new interactive systems by mapping various kinds of inputs (e.g. sensor data, motion descriptors, audio features) and parameters of sound and music processes. Significantly, *Wekinator* enables this mapping by demonstration, rather than by writing code (Fiebrink, 2011; Bernardo et al., 2020). IML tools allow users to make an assessment of the model's performance in order to determine how to improve it (Fiebrink et al., 2011). This process—creating a model, hearing the result, making a judgement, and changing the model accordingly—relates strongly to Leman's Action-Reaction cycle (2008). Fiebrink et al. (2011) found that the feedback users gained from evaluating the output of *Wekinator* helped them learn how to interact more effectively with the system, and to discover and manage trade-offs between the models that were easy to build and those which were more useful in practice.

2.4.4 Visual design of DAWs

While the emergence of DAWs represented a new stage in music production, the visual design of many DAW interfaces seems to signal their intended use via the visual emulation of past technologies. Duignan et al. (2004) consider various metaphors in *Reason* and *Ableton Live*, including the

emulation of mixing desks and multitrack recorders, and connections between devices via patch cables. While both *Reason* and *Ableton Live* use conventional metaphors (e.g. knobs), and music hardware metaphors (e.g. mixer sends), *Reason* more closely mimics hardware, whereas the design of *Live* is more abstract. *Reason* maintains a hardware metaphor even when it risks harming visibility or interaction; for example, the virtual rack equipment used in *Reason* (such as effects units and mixers) places controls on the front and sockets on rear of the virtual equipment. This requires the user to toggle between ‘front’ and ‘rear’, and it is not possible to show both settings and connections in a single window. By adhering so strongly to the hardware metaphor, the downsides of hardware are ported over into software.

Such visual emulations often make use of skeuomorphism—the use of ornamental ‘informational attributes’ (Gessler, 1998) which were necessary in the original design of an object (Spiliotopoulos et al., 2018). These designs help to reinforce affordances by leveraging the user’s familiarity with existing objects. For example, the knowledge that an object that looks like a button affords pressing is an example of perceived affordance (Norman, 1988).

An example of a skeuomorphic design with perceived affordances is the use of fader caps and rotary potentiometers in *Pro Tools*, shown in Figure 2.1. The fader caps are rendered as concave capacitive metal caps of the type used on high-end consoles. Beyond the fader’s practical role in both displaying the current level and allowing it to be changed, the fader cap shows the user where to click and drag while suggesting that they are using a high-quality hardware device. An alternative approach is to remove the strict fader metaphor while retaining the overall interaction. As an example, *Ardour*’s use of a textured strip instead of a fader—shown in Figure 2.2—is still skeuomorphic as it makes use of a ribbon controller metaphor. As it does not follow the traditional desk layout it has increased the size of the target as the user can click and drag anywhere along its length, thereby improving mouse control. However, the photorealistic presentation of typical mixing desk features may be important, as will now be considered.

Marrington (2016) argues that the skeuomorphic presentation of software emulations signals the intended use case—for example, *Reason*’s emulated hardware samplers and drum machines imply tools typically used in Hip-Hop, the waveforms and mixing desk interface of *Pro Tools* appeal to the concerns of studio engineers, and the virtual manuscript of *Sibelius* speaks to the requirements of classical musicians. According to Zagorski-Thomas (2014), ‘the choice of visuals, of what is represented, when and how, is a very powerful influence on the user’. As an example of an abstracted interface in DAW design, *Tracktion* (Rink, 2017), shown in Figure 2.3, applied a more abstract metaphor to the management of signal flow, showing audio flowing from left to right through user-controlled processing ‘blocks’ rather than emulating any specific hardware design.

2.5 Technological frames

Related to the use of metaphor and embodied cognition is the concept of *technological frames*. Orlikowski and Gash (1994) consider the cognitive science concept of frames of reference—how people act on the basis of their interpretations of the world, enacting social realities and endowing them with meaning—and apply these frames to assumptions and expectations about technology. They argue that, due to the fact that technologies are social artefacts, their creators will instil in them their own objectives, values, interests, and knowledge. The users of technology may have varying interpretations of technology due to incongruent technological frames, leading to unknowingly misaligned expectations, and unanticipated consequences such as resistance to use, skepticism of the usefulness of technology, and inconsistent adoption.

Sections 2.2, 2.3 have so far considered the frames of reference introduced by common music notation (music should conform to specific pitches, harmony, meter) and phonography (music is, and should be, repeatable). Similarly, Section 2.4.3 considered the expectations and limitations set by the emulation in software of recording hardware, which is an example of technological framing.



Figure 2.1: The mixer in *Pro Tools*, showing the use of skeuomorphic faders and rotary pots.

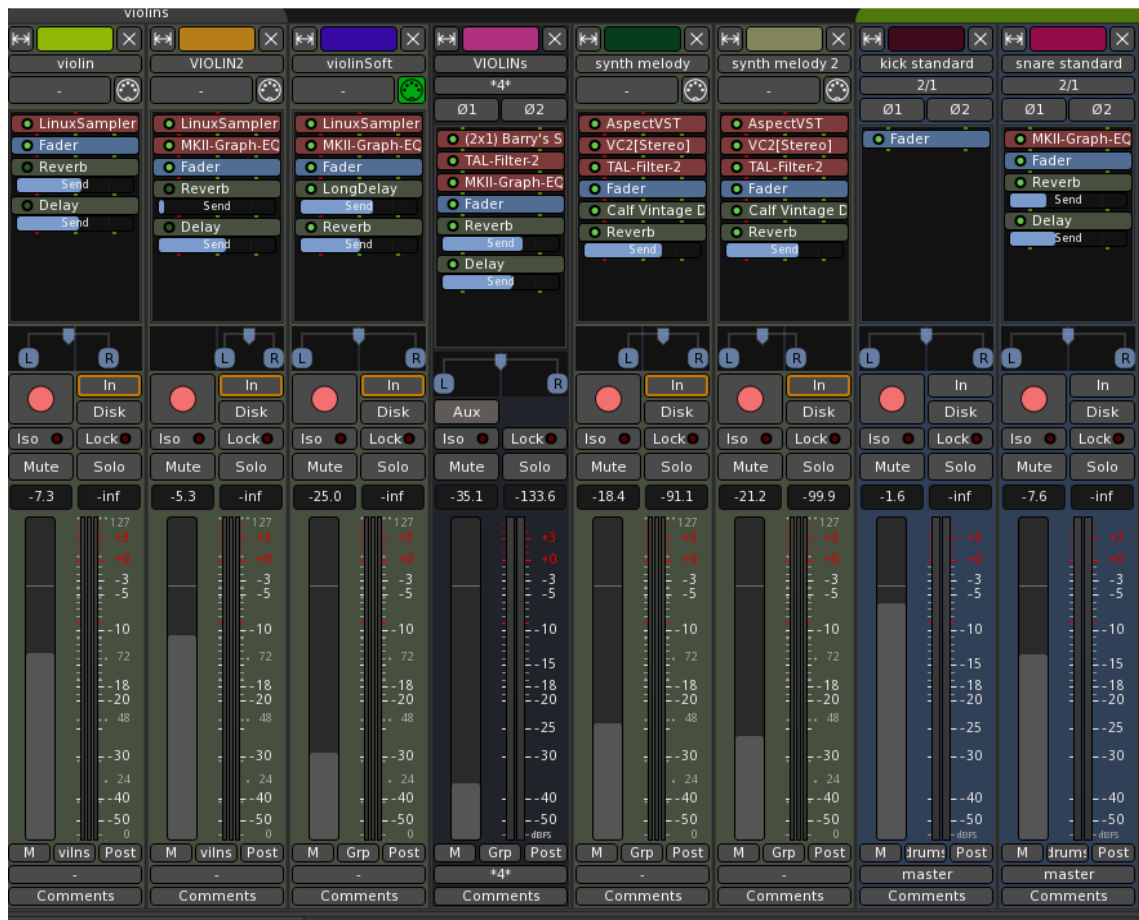


Figure 2.2: Ardour's mixer, showing the use of a ribbon controller-style strip rather than a traditional fader.

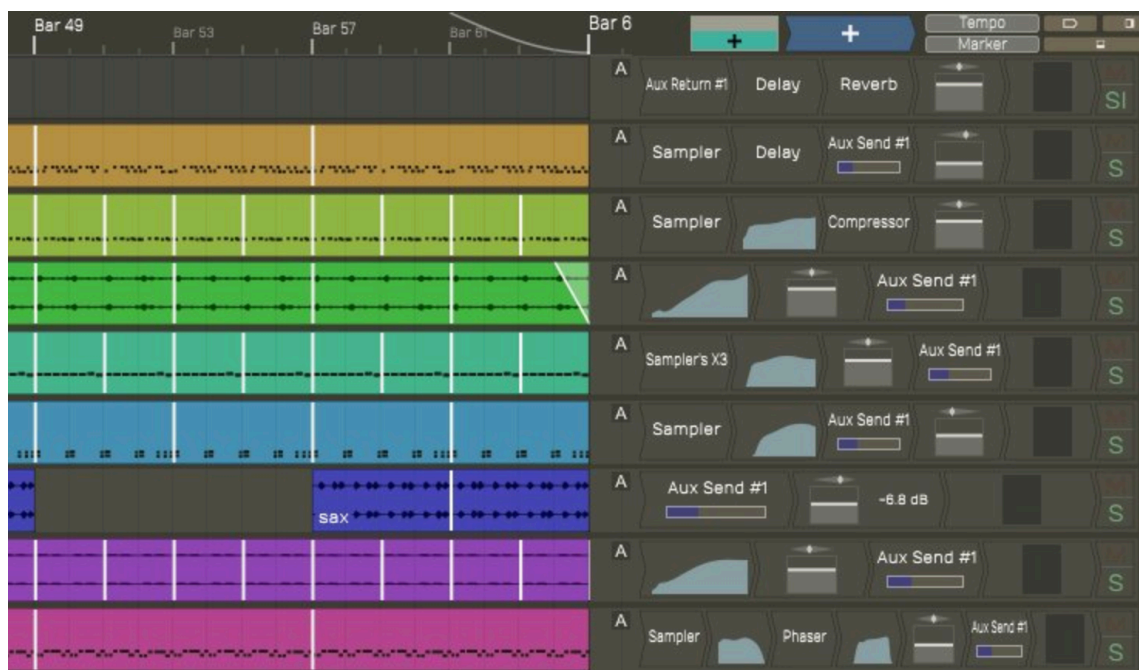


Figure 2.3: Signal flow in *Tracktion* (Rink, 2017).

2.6 Algorithmic composition

Algorithmic music composition is the application of an algorithm or algorithms to the process of composing music. Before computers, algorithmic music left selected compositional or performance decisions to chance, as seen in pre-computer systems such as those involving dice (Hedges, 1978); aleatory literally translates as ‘on the throw of a die’. In the twentieth century indeterminacy in music referred to the process of leaving some elements of composition or performance open to change, either by chance or by human agency, allowing a piece of music to be performed in substantially different ways (Cage and Tudor, 1959). In contrast to improvisation, which commonly refers to decisions made by musicians who are leveraging an understanding of common practice in a genre (Piekut, 2014), indeterminacy typically refers to selections made without a sense of common practice (Reynolds, 1965). The work presented in this thesis may be viewed as being particularly focussed on stochastic music (Xenakis, 1966), which makes use of mathematical probabilities in composition or performance. Probability-driven selection is an important element of several algorithmic music techniques, such as generative grammars and Markov chains (Nierhaus, 2009).

Fernández and Vico’s characterisation of algorithmic composition is the ‘partial or total automation of music composition by formal, computational means’ (2013). While some algorithmic composition systems use formal processes to make music with minimal human intervention (Alpern, 1995), other approaches allow user input. Computer-aided algorithmic composition (CAAC) is commonly defined as the use of software to provide raw materials to the composer in order to support the creative process (Fernández and Vico, 2013). In a paper on the use of algorithmic composition to support the creative act, Jacob (1996) compares the ‘genius’ act of spontaneous creation with the ‘hard work’ of iteratively trying various options before making a final choice, arguing that the ‘hard work’ method of composition is inherently algorithmic. In this context an algorithm can replicate the iterative task of trying multiple options, leaving the composer free to choose between the results. Jacob notes that an algorithm without human intervention is not capable of ‘breaking the rules’, a capability he sees as an essential creative tool (Jacob, 1996). The CAAC model of algorithmic composition requires the system-author, when writing the algorithm, to consider the steps they use to create music and, for example, the types of harmonic movement that they prefer. The algorithm they create can then iterate, outputting multiple variations for the composer to either use or reject. By encouraging human evaluation and feedback the system echoes Leman’s Action-Reaction cycle (2008).

Of the diverse possibilities for the use of algorithms in musical composition, CAAC represents a low degree of automation. Fernández and Vico (2013) consider algorithmic composition to differ from CAAC in the degree of automation: algorithmic composition automates the compositional process to a higher degree than CAAC, to the point where human input can be either minimal or entirely non-existent. In this way of working the user may not be the primary source of creativity, although they may need to direct the process by setting parameters, encoding compositional knowledge, or providing musical examples.

Computer game music is a related area in which music is frequently required to change according to rules and depending on the game state (Collins, 2008). This is often referred to as non-linear music (Buttram, 2003). This type of music differs from algorithmically composed music as it reacts to inputs and changes the output according to predefined rules.

2.7 Usability and expressivity trade-off

The conventional definition of usability, as outlined by Jacob Nielsen (2003), considers usability as an attribute which assesses how easy user interfaces are to use. By his definition, usability is defined by learnability (how easily a new user can accomplish basic tasks), efficiency (how quickly users who are familiar with the design can perform tasks), memorability (how easily users can re-establish proficiency when they return after a period of not using the design), errors (the number

and severity of user errors, and the ease of recovery), and satisfaction (how pleasant the design is to use). Nielsen distinguishes between utility (the provision of features) and usability (how easy and pleasant the features are to use). An alternative definition is given by Rogers et al. (2019), who consider usable interactive products to be easy to learn, effective to use, and provide an enjoyable user experience. They suggest that usable products should be effective, efficient, and safe to use (effectiveness; efficiency; safety), have good utility (utility), be easy to learn (learnability), and be easy to remember how to use (memorability). As can be seen, there is significant overlap between the usability definitions of Nielsen (2003) and Rogers et al. (2019).

However, in the context of my particular focus on novice programmers, a useful alternative way of thinking about usability and expressivity comes from an excellent study by Repenning and Ioannidou (1997) which considers expressivity and usability in the context of novice programmers. In a paper on the design of *AgentSheets*, a computer system designed for educational use (Repenning and Sumner, 1995; Repenning et al., 1998, 2000), Repenning and Ioannidou (1997) discuss the balance between language usability and expressivity. According to their account, in order to be deemed usable for novices the interface must offer a clear solution for a task. In their view, usability is reduced when the user has to assemble their own operations from lower-level operations.

In the same paper, software is described as expressive if it provides tools to the user to solve complex problems. Repenning and Ioannidou argue that the trade-off between usability and expressivity can be plotted with usability (the effort required to achieve a given goal) plotted on the x axis and expressivity plotted on the y axis. An example graph of this type by Repenning and Ioannidou (1997) is shown in Figure 2.4. Given that one of the research goals of this thesis is to consider how nondeterministic composition tools can be made accessible to non-programmers, this characterisation of the usability/expressivity balance is useful when considering the trade-offs that may be required.

An alternative characterisation, suggested by Myers et al. (2000), is to consider how difficult a system is to learn (the threshold) and how much can be done using the system (the ceiling). Many existing systems are either low-threshold and low-ceiling, or high-threshold and high-ceiling. The ideal design would result in a system with a low threshold and a high ceiling. An additional goal for creative support tools, suggested by Resnick et al. (2005), is ‘wide walls’. This refers to a system which supports and suggests a wide range of explorations by providing general primitives that users learn to combine.

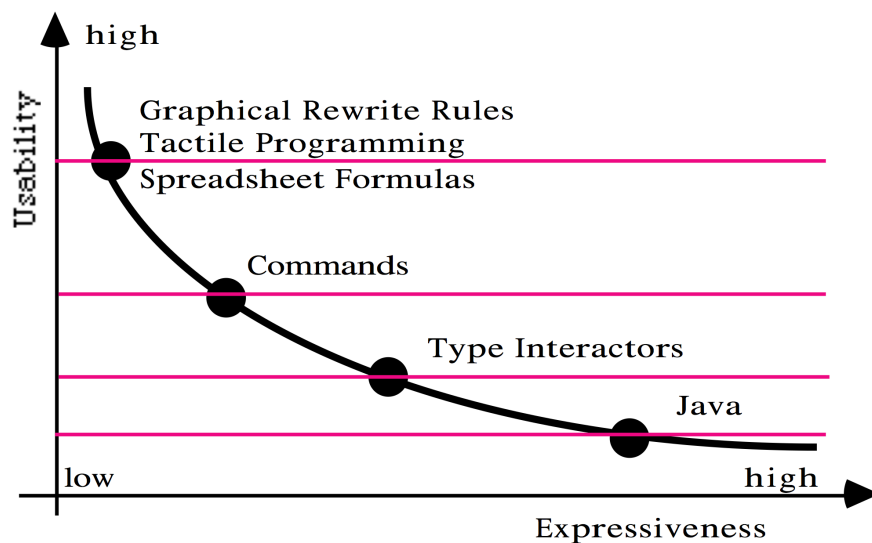


Figure 2.4: Four points in the usability/expressiveness space (Repenning and Ioannidou, 1997).

In order to deal with the trade-off, Repenning and Ioannidou (1997) propose a layered interface which allows users to choose their own point on the usability/expressivity continuum. They claim

that advanced users can make use of additional controls which can be hidden from novice users. For example, applying this notion to music, music novices require an interface which does not presuppose knowledge of score notation or of harmonic, melodic or rhythmic terminology. By contrast, music experts require the ability to leverage their knowledge of these elements within the interface. An example of a layered interface, *WolframTones*, is described in Section 2.8.1.

2.8 Critical review of representative algorithmic music composition systems

Algorithmic composition, like any other musical practice, is deeply influenced by the tools used. Tools for algorithmic composition vary widely. Given the goal of giving non-programmers access to highly expressive tools for algorithmic composition it will be helpful to critically review a representative sample of such existing tools, with a particular view to the trade-off between expressivity and usability. This review will use the characterisations of these two terms introduced by Repenning and Ioannidou (1997) in the context of systems for novice programmers, outlined in Section 2.7. The use of the usability/expressivity continuum as a suitable critical lens will enable us to identify an initial set of finer-grained design issues and trade-offs. In addition to music software it is useful to review some non-musical systems which give non-programmers access to expressive nondeterministic tools. The resulting collection of software is heterogeneous and varies on many dimensions: included are an example of ‘black box’ software with minimal user input; software with a tabular, form-filling UI; and both visual and text-oriented programming languages.

2.8.1 *WolframTones*

A representative example of web-based algorithmic composition software is *WolframTones* (Wolfram Research Labs, 2005). The user is presented with a limited set of controls presented as a ‘black box’ (Rosenberg, 1982). Selecting a genre generates a piece of music, with variations available either by making another selection or by accessing a list in the ‘show variations’ menu. Compositions are shown as rectangular graphics and are differentiated visually by comparing these simple images (see Figure 2.5).

WolframTones makes use of a layered interface, as outlined by Repenning and Ioannidou (1997), allowing users to choose their own point on the usability/expressivity continuum. Novice users can select playback via the genre buttons while more advanced users can make use of the additional parameters in the ‘composition controls’ section. Some of these advanced controls are self-explanatory (instrumentation, for example, allows the selection of standard General MIDI sounds) but others are abstract and not easily understandable (for example, a rule type of ‘15 ($r=3/2$)’, as seen in Figure 2.6, is not easily understood by non-mathematicians). The visualisation of music is abstract and does not represent the content in a meaningful way.

It is easy for users of any skill level to create multiple pieces of music; no musical knowledge is required to achieve a result. Expressivity is low, however, resulting in software that cannot be used to create a specific piece of music. As a result, *WolframTones* may not suit users who need to exert detailed control over the output.

2.8.2 *Noatikl*

Noatikl (Intermorph, 2015b) is a development of *Koan*, the software used by Brian Eno for *Generative Music 1* (Media Art Net, 2006). *Noatikl* makes use of a patching metaphor to enable users to create generative music compositions. Patching allows the visualisation of elements such as signal flow and Boolean logic, and the resulting data flow visibility is high. The software has a reasonably consistent design language, leveraging both hardware synthesisers (the use of photorealistic

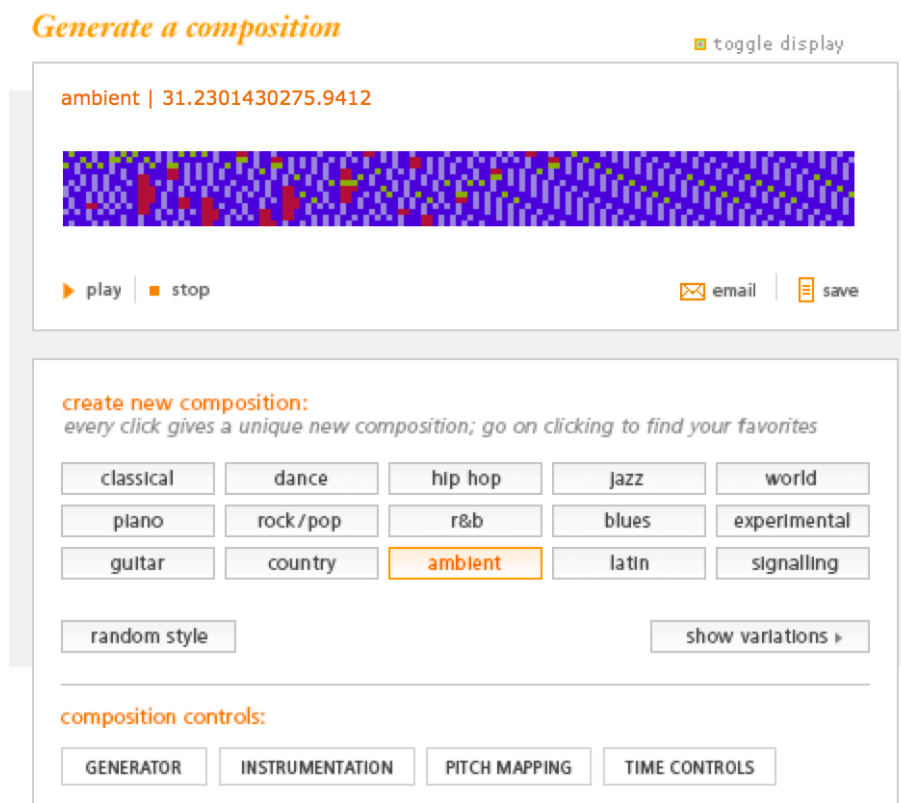


Figure 2.5: The main interface of *WolframTones* (Wolfram Research Labs, 2005); screenshot taken in 2014.

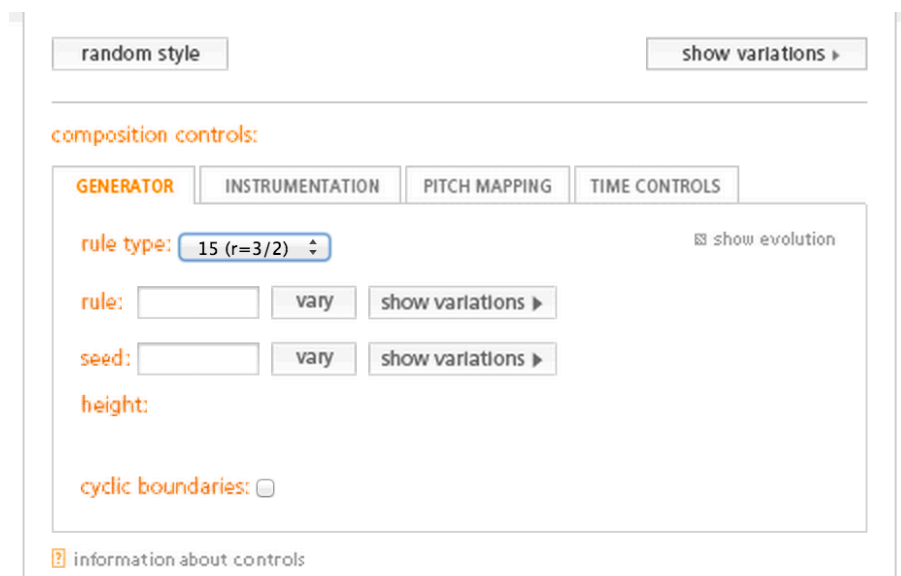


Figure 2.6: An example of a layered interface: the composition controls in *WolframTones* (Wolfram Research Labs, 2005); screenshot taken in 2014.

rotary potentiometers and faders) and patching (patch cables which ‘droop’ as physical cables do; see Figure 2.7).

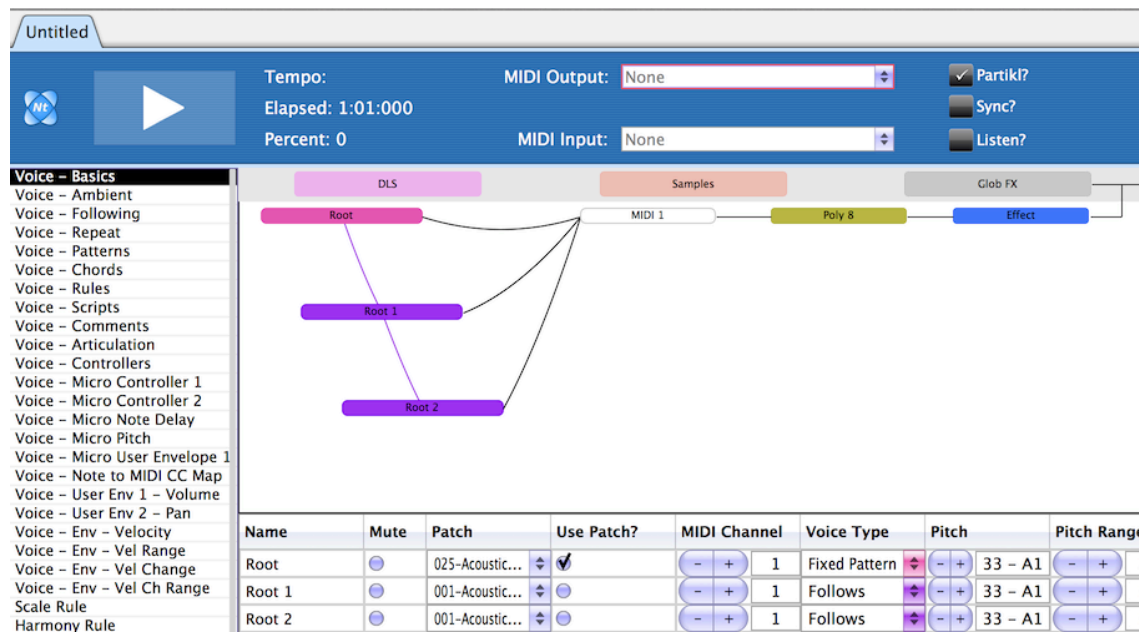


Figure 2.7: Data connection in *Noatikl* (Intermorphic, 2015b).

The structure of a piece of music, depending on the genre, can be a crucial element of the composition. If the genre requires strong structure at the top level, a patching interface such as the one used in *Noatikl* can use the spatial layout of objects to match the user’s conceptual structure.

2.8.3 Scratch

Scratch (Resnick et al., 2009) is a visual programming environment designed to support self-directed learning. *Scratch* was initially designed to introduce programming to novice users, with a specific focus on the needs and interests of young people aged 8 to 16. *Scratch* makes use of a visual blocks language, a single-window user interface, and a minimal command set (Maloney et al., 2010). *Scratch* allows users to make changes to a script while it is running; it therefore exhibits a high level of liveness (see Section 2.2). *Scratch* also allows for what Maloney et al. (2010) refer to as ‘tinkering’; users can experiment with, create, and test small chunks of code before combining them into larger units. *Scratch* uses a white border around scripts to visualise execution. By design, syntax error messages are not required in *Scratch* as the combinatorial possibilities of scripts are limited by how blocks can interlock. The block-based design removes the need for the user to learn syntax while preventing the user from making combinatorial errors. *Scratch* is a general purpose programming language which can be used for music; a musical example is shown in Figure 2.8.

2.8.4 Sonic Pi

Sonic Pi (Aaron and Blackwell, 2013; Aaron et al., 2016) is a language designed for the creation of live-coded music. The language was originally developed for use with the Raspberry Pi computer in an educational context to support the development of computational thinking in students. *Sonic Pi* is designed to trigger and control synthesisers which are generated and manipulated using the *SuperCollider* synthesis server. *Sonic Pi* is implemented as an embedded Ruby domain-specific language (DSL), meaning that novice users can take advantage of Ruby’s forgiving parser to omit some syntactic elements (such as parentheses around method arguments and semicolons between statements). Much of the complexity in the language is hidden from end users, and common musical

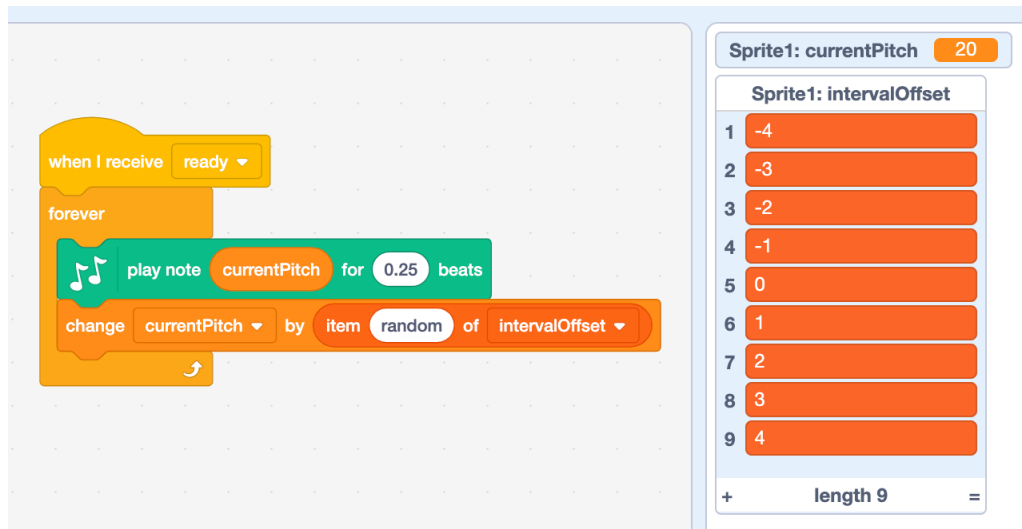


Figure 2.8: A *Scratch* patch, showing the use of interlocking block shapes to limit the opportunity for user error.

operations are presented in an English-like syntax (e.g. `10.times` for repetition) or can be explained using a simple metaphor (e.g. describing `do` and `end` tokens as bookends surrounding code to be repeated). The *Sonic Pi* IDE provides common music transport controls (e.g. run, stop, record), multiple independent workspaces, an editor with syntax highlighting, an information pane for real-time feedback, and an error pane to report syntax and runtime errors to the user. A screenshot of the IDE can be seen in Figure 2.9.

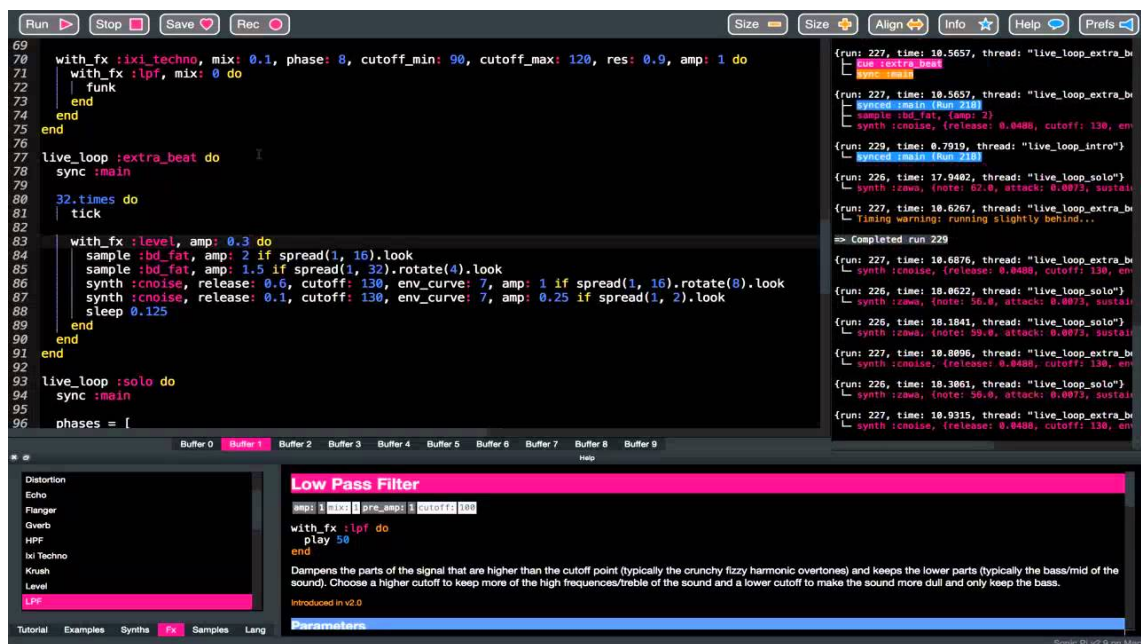


Figure 2.9: The *Sonic Pi* IDE (Aaron and Blackwell, 2013), showing control buttons, workspace tabs, the editor pane, the information pane, and the error pane.

2.8.5 Fractal Tune Smithy

Robert Walker's *Fractal Tune Smithy* (2011) offers significantly more expressivity when compared with *WolframTones* but with a resulting reduction in usability. The software makes use of several

windows with multiple data entry points, allowing the user to review multiple parameters simultaneously.

The software makes use of musical metaphors (such as a rudimentary piano roll) for some elements, while the user defines sections, shapes, masks and note structures which the program's 'generators' use to create new material. As exemplified in Section 2.8.1, there are several pieces of algorithmic music composition software which do not enable, or require, the user to interact with the method of generation. However, *Fractal Tune Smithy*, in common with other software such as Jonathan Middleton's *Musical Algorithms* (2004), requires musical input. This user-provided input is acted on using algorithms which are both hidden from, and inaccessible to, the user.

Fractal Tune Smithy makes use of an eclectic and arguably inconsistent design language which relies on metaphors from both the music and music production spaces. The design uses musical notation, a piano roll, mixing desk-style controls, text-based data entry, various icons such as stars and flags, and window and card metaphors (see Figure 2.10).

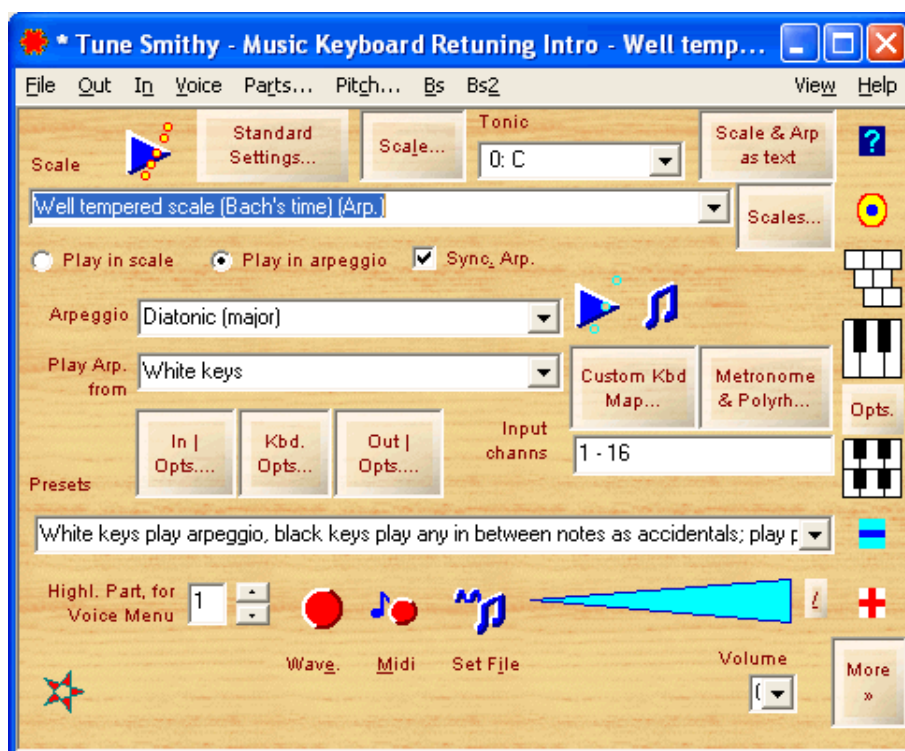


Figure 2.10: One of several window types in *Fractal Tune Smithy* (Walker, 2011).

2.8.6 Manhattan

Manhattan (Nash, 2014) is system which combines the functionality of a tracker with end-user music programming. In common with the research questions listed in Section 1.2, *Manhattan* has a focus on making algorithmic composition tools accessible to non-programmers. In order to reduce the learning threshold of programming, *Manhattan* augments a tracker interface with end-user development capabilities (to be addressed in Section 2.11) via the use of spreadsheet-style formulae. The merging of trackers and spreadsheets—both existing tabular systems—allows the user to programmatically manipulate the contents of a cell (see Figure 2.11 for an example) and to define relationships between cells. In doing so, *Manhattan* augments trackers with end-user control over expressions, built-in functions, variables, pointers and arrays, iteration (for loops), branching (goto), and conditional statements (if-then-else).

When compared to MIDI sequencers, the text-oriented and keyboard-driven interface of trackers may initially seem to have a higher threshold [more difficult to learn; see Myers et al. (2000)]

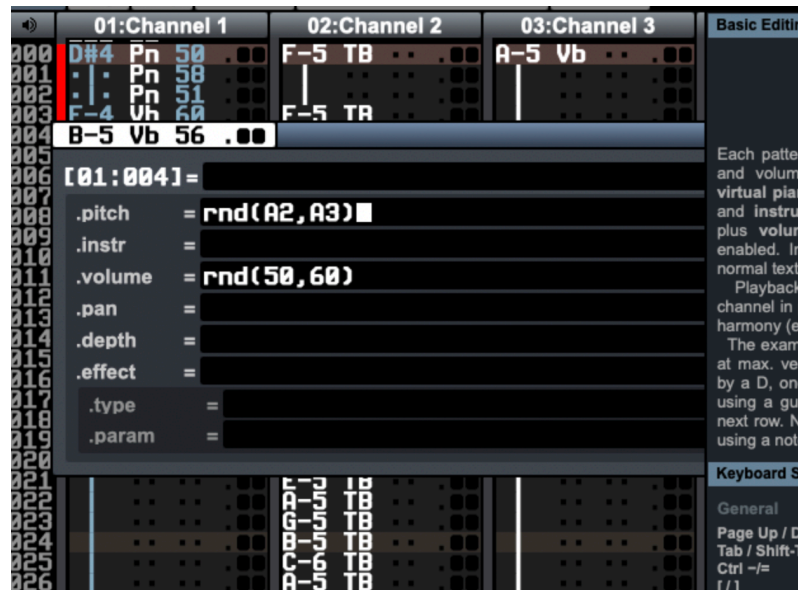


Figure 2.11: *Manhattan* (Nash, 2014), showing the use of spreadsheet-style formulae to randomise a cell's pitch and volume parameters.

and to have a higher ceiling (more capable than sequencers in some contexts, e.g. in the creation of dense rhythmic patterns). However, sequencers may only have a lower threshold than trackers if the user is already familiar with the metaphors typically used in sequencer design (e.g. piano keyboard, mixing console).

2.8.7 *OpenMusic*

OpenMusic (Assayag et al., 1999; Bresson et al., 2011) is a visual programming language which is primarily designed for music composition. Objects can be placed by the user in a patch window; objects are shown as boxes, and include functions, classes, instances, subpatches, and maquettes. Objects can be edited graphically using tools such as score or audio editors. Maquettes are patches with a horizontal time dimension, which allows the musical material to be assembled into time-oriented structures. Objects are then connected together via inlets on the uppermost edge of the object, and outlets on the bottom edge, as shown in Figure 2.12.

2.8.8 *Max*

Max (Cycling '74, 2019) is a visual programming language in which connections between objects are made with virtual patch cables. Depending on the size and complexity of the program, visual languages can be visually expressive as the connections between elements can be clear. For example, it is possible for a user to see an image of a *Max* patch and understand the interrelationships of the objects in a relatively intuitive way²—see Figure 2.13 for an example of a readable patch.

Reflecting the fact that much physical patching utilises a unidirectional (i.e. audio send or return connections) rather than a bidirectional (i.e. MIDI, USB) connection, visual programming languages such as *Max* or *Pure Data* employ typically use a one-way patch cable metaphor. When there are multiple connections the patch can become difficult to read and work with (sometimes referred to as 'spaghetti hell' (Duignan et al., 2004): an example in *Pure Data* can be seen in Figure 2.14). The use of subpatches can improve these visual issues. Subpatches allow the programmer to create objects with inlets and outlets which can abstract and simplify a patch at a given level. Subpatches

²Intuition can be defined as a set of assumptions informed by prior experience in similar situations (Raskin, 1994). 'Intuitive' is used in this thesis to describe interfaces and interactions which, due to their background and previous experiences, users find comfortable to use and easy to learn.

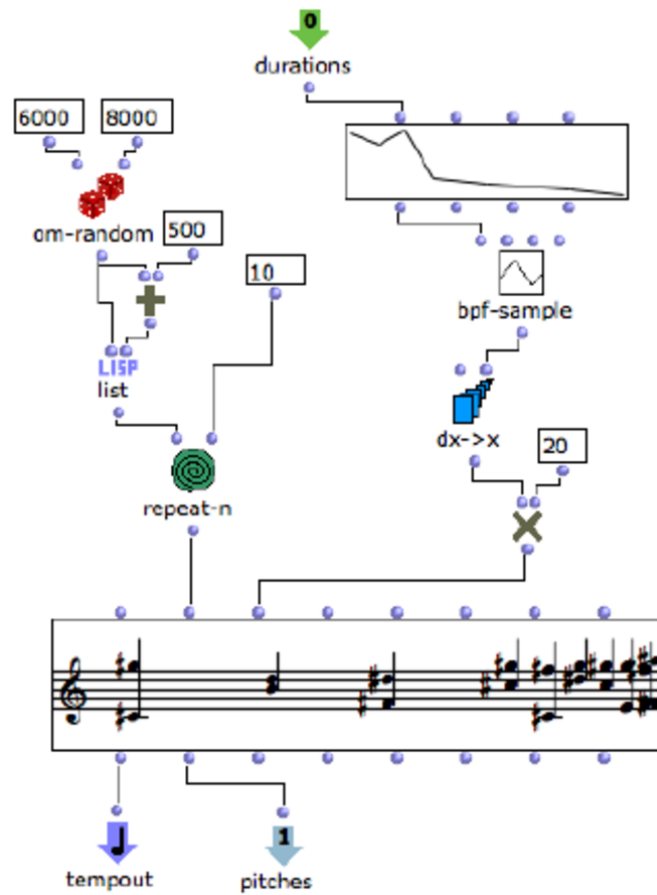


Figure 2.12: An example *OpenMusic* patch (Bresson et al., 2011).

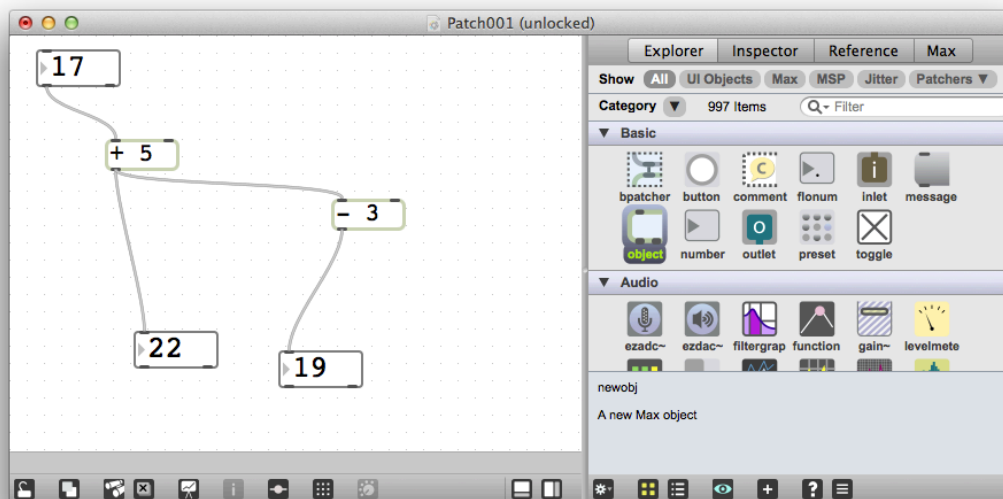


Figure 2.13: A simple patch in *Max* (Cycling '74, 2019), showing the interrelationships between objects denoted by virtual patch cables.

can also be used to easily copy material. Note that there is an associated trade-off; abstracted connections (‘send’ and ‘return’ objects, for example) allow for non-visual connections with a potential reduction in visual readability.

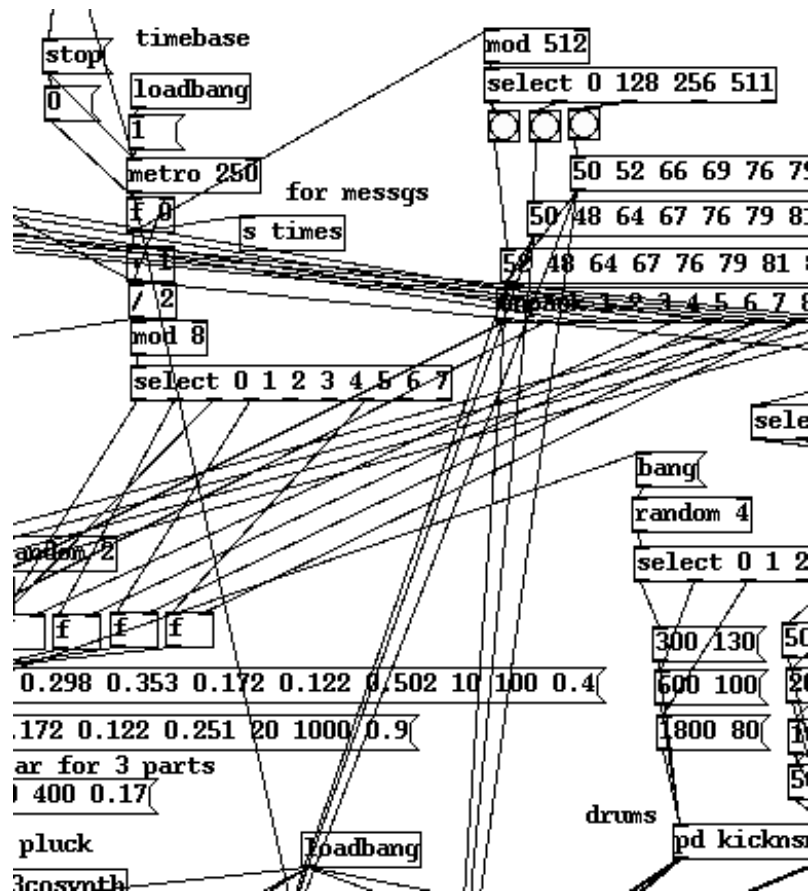


Figure 2.14: Dense connections in *Pure Data* creating visual ‘spaghetti’ (Duignan et al., 2004). Image from Farnell (2008).

Graphical systems such as *Max* allow users to start with one element and then iteratively expand the patch. A simple patch using a patch-cable analogy can be easily readable, but increased complexity can lead to a visually congested patch which then becomes hard to read, understand, and maintain. While graphical systems like *Max* and *Pure Data* can deal with complex tasks, this is only possible if the user knows or learns multiple programming abstractions and memorises many arbitrary conventions. Many of the programming abstractions and conventions are idiosyncratic to the particular piece of software.

2.8.9 *Max for Live*

Ableton *Live* (Ableton, 2021), also colloquially known as ‘Ableton’, is a DAW which offers sequencing, triggering, and processing controls but, as outlined by Blackwell and Collins (2005), *Live* does not provide the deep algorithmic control available via tools such as *Max*. A bridging system for users with both *Max* (see Section 2.8.8) and Ableton *Live* installed, *Max for Live* (Cycling ’74, 2020; Ableton, 2020) allows the two systems to interoperate, significantly increasing the extensibility and programmability of *Live*. *Max for Live* allows the user to create and manipulate a variety of objects including instruments, signal processing, tools for live performance, and visualisations. *Max for Live* can be used to connect Ableton *Live* to a wide range of external devices (e.g. via Arduino or OSC). This extensibility has enabled *Max for Live* to be used as part of new systems; for example, *Gibberwocky* (Roberts and Wakefield, 2016) is a live-coding system which combines *Max for Live*

with a web-based textual environment.

2.8.10 *SuperCollider*

SuperCollider (McCartney, 2021) is a text-oriented music language featuring an object-oriented language similar to Smalltalk with a C-like syntax. Many of the text-oriented languages used for music, such as *SuperCollider*, *Csound* (Vercoe, 1996), and *Impromptu* (Sorensen, 2010), require the user to learn their syntax and abstractions before any meaningful output can be created. This creates a steep barrier to entry for non-programmers, although the flexibility such languages afford results in an environment with high expressivity. *SuperCollider* offers a number of tools that allow for code to be reused or expanded: multichannel expansion, for example, is used to create separate outputs following a user-defined sequence. Such tools expand the expressivity of the language while reducing usability due to the additional burden it places on the user's knowledge of the syntax. *SuperCollider*'s IDE (Integrated Development Environment) includes autocompletion of class and method names (see Figure 2.15). Such a system significantly reduces errors introduced by mistyping. Default settings are not always directly visible. *SuperCollider* has defaults for every object but they are only visible within the help files (see Figure 2.16). The parameters are explicitly visible when the user issues them as methods but the method order (i.e. which parameter is controlled by each command) is again only visible in the help files. An experienced user will memorise the method order but this lack of visibility has an impact on the novice user.

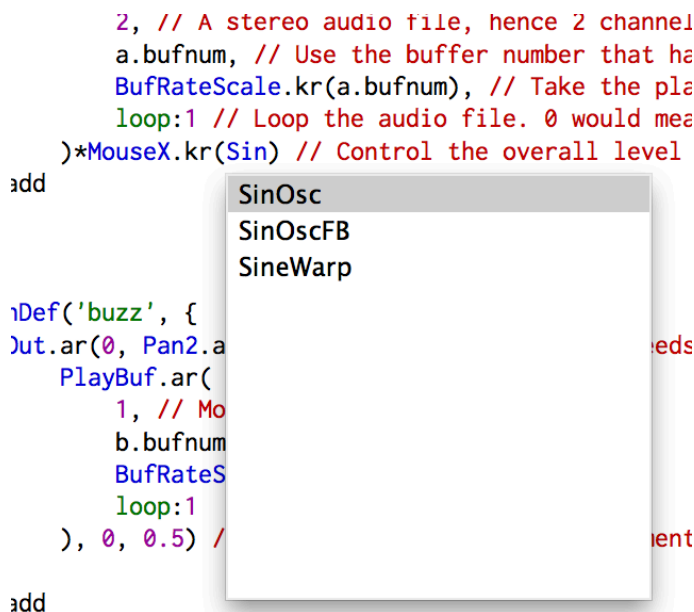


Figure 2.15: Auto-complete suggestions in *SuperCollider* 3.6.

While it is clear that the software reviewed above varies greatly in terms of the trade-off between usability and expressivity, the balance of these trade-offs depends on the purpose, background, and experience of the user, the amount of code involved, and many other factors. Having reviewed a range of software the next section will review one interaction principle and various diagrammatic representation techniques which in turn offer possible insights into exploratory music programming. In particular, they both suggest ways of helping non-programmers to gain fine-grained control over data and behaviour.

SinOsc

Interpolating sine wavetable oscillator.

Source: /Applications/SuperCollider/SuperCollider.app/Contents/Resources/SC
ClassLibrary/Common/Audio/Osc.sc
Inherits from: [PureUGen](#) : [UGen](#) : [AbstractFunction](#) : [Object](#)

See also: [FSinOsc](#), [SinOscFB](#)

Description

This is the same as [Osc](#) except that the table is a sine table of 8192 entries.

Class Methods

```
* ar (freq: 440, phase: 0, mul: 1, add: 0)
* kr (freq: 440, phase: 0, mul: 1, add: 0)
```

Figure 2.16: *SuperCollider* help file for the SinOsc sine oscillator.

2.9 Direct Combination: a user interaction design principle

In investigating how nondeterministic composition tools can be made accessible to non-programmers, it may be useful to consider relevant interaction design principles. Direct Combination is a user interaction technique which focusses on the operations that become possible given a pair of interaction objects. In a Direct Combination system, once the user selects a pair of objects they are presented with a range of operations which are well suited to the combination of the chosen objects (Holland and Oppenheim, 1999). An example, given by Holland et al. (2002), is a scenario in which a user selects a word processing file and a remote display, triggering the option to display the file on the screen. Direct Combination (DC) has the merit that one of its key properties is to reduce the search space in user interactions (Holland and Oppenheim, 1999). In other words, the number of steps users need to take in order to complete a given task can typically be reduced compared with conventional interfaces for any given total number of possible options, thus potentially maximising usability for any given expressivity using the chosen principal characterisation of usability.

A precursor to DC, Slappability, was developed by Daniel Oppenheim as an interaction framework for his DMIX music composition software (1994). The design allows any two arbitrary materials (such as durations, pitches, timbres, or pitch lists) to be combined by dragging the source onto the target object. Once the mechanism is initiated, any of the source object's characteristics (e.g. durations, pitches, timbres, pitch lists) can be applied to any of the target object's parameters.

2.10 Diagrammatic representations

Having considered an interaction technique offering possible insights into exploratory music programming, this section will turn to diagrammatic representations which could be similarly useful.

In order to locate suitable graphical systems capable of representing hierarchical musical information it is instructive to consider existing information flow modelling tools outside of music. Various such tools are well suited to solving problems in complex systems due to the promotion of clusters of information which match the domain and/or perceptual inferences of the users, making them easy to use (Hungerford et al., 2004). Examples of relevant diagrammatic representations include data flow diagrams (DeMarco, 1979), input-process-output (IPO) diagrams (Stay, 1976; Katzan, 1976; Martin and McClure, 1985), design structure matrices (Eppinger, 1991; Browning, 2001; Helo, 2006), and the three systems identified as candidate systems for the purposes of the work presented in this thesis, described below: decision trees (DTs), influence diagrams (IDs), and Petri Nets (PNs).

2.10.1 Decision trees (DTs) and influence diagrams (IDs)

Decision trees are the basis of many decision support tools which show possible decisions and their consequences in a tree diagram. Decision trees are highly expressive but the specification load—the size of the graph—increases exponentially with the number of decisions (Jensen et al., 2006).

Influence diagrams are an alternative graphical representation of a decision situation. IDs have a linear increase in the specification load but have limited expressivity when compared with DTs. Both systems make use of arrows to show both connection and order. The direction of the arrow, rather than the placement of the objects, signifies order. An example is shown in Figure 2.17.

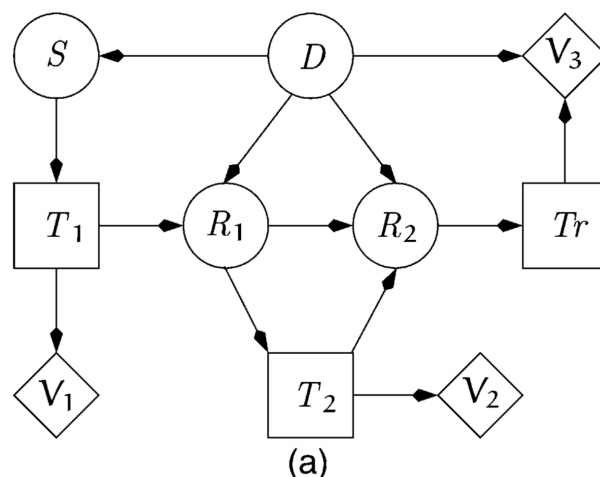


Figure 2.17: An Influence Diagram (ID) showing the choices to be made by a diagnosing doctor (Jensen et al., 2006).

2.10.2 Grammar-based hierarchical structures

Musical structure can be represented via hierarchical structures such as those by Schenker and Oster (1979) and Lerdahl and Jackendoff (1983). The latter's focus on note-level grammar (1983) is interesting for analysis but is less suitable for music generation, and is tightly tied specifically to tonal music. By contrast, *Humdrum* by David Huron (2002) is an example of a musical analysis tool which allows for hierarchical structural analysis. The structural representation tools provided by Alan Marsden (2005) analyse and reveal hierarchical structure at the micro level, and they propose methods (such as an 'elaboration tree') to make a Schenkerian hierarchical analysis system generative. However, the work presented in this thesis is focussed on tools for authoring, rather than analysing, musical structure.

Wiggins et al. (1993) proposed a framework to evaluate music representation systems which included systems for recording, analysis, and generation/composition. They define two orthogonal dimensions; *expressive completeness* as the amount of raw musical data which can be represented, and *structural generality* as the amount of information about musical structure which can be encoded explicitly. For example, a performance shown as a frequency spectrum has very high expressive completeness but low structural generality as high-level structures (such as musical structure, tonal centres, or metric information) have not been represented and cannot be manipulated. Wiggins et al. (1993) note that MIDI, as a communication protocol, was designed to emphasise expressive completeness rather than structural generality. MIDI represents the timing of events with reasonable accuracy but it abstracts away significant pitch information due to the approximation of pitches to the piano keyboard, and no tuning system information is included. As a result, the expressive completeness of MIDI is significantly lower than that of a frequency spectrum. While MIDI's structural generality is superior to that of a frequency spectrum (due mostly to the concept of a MIDI

note) it remains relatively poor as there is no ability for structural annotation.

There are a number of music analysis visualisation notations, such as the mathematical analysis of song structures (Gündüz and Gündüz, 2005), melodic analysis (Ignelzi and Rosato, 1998), gestural representation (Mazzola and Andreatta, 2007), arc diagrams to reveal structure (Wattenberg, 2002), and structural analysis (Stell, 2004). Chan et al. (2010) present an interesting analysis tool which makes use of colour and shade to denote instrumental families, pitches, and functional roles of instruments over time. Their visualisation is not appropriate to the work presented in this thesis given the lack of structural information and the focus on individual instrument roles rather than musical structure.

Blackwell et al. (2001) consider diagrammatic reasoning with relation to visual programming. They suggest a ‘foundational metaphor’ in each diagram type, typically corresponding to the underlying language; for example, expressing memory locations in BASIC as metaphorical pigeon holes, or showing data as ‘flowing’ along wires. The paper explains that diagrams can clarify certain issues while obscuring others, with diagrams used to support reasoning in problem solving contexts. The authors argue that diagrams are not universally superior to text; each has benefits while simultaneously obscuring other information. While Blackwell et al. (2001) acknowledge popular support for the use of diagrams for algorithmic visualisation, they do not find evidence of any clear advantages. One example that they cite of a choice of visualisation paradigm—between control flow and data flow—is interesting from a musical perspective. Given the central importance of time in music a musician is used to commands being issued in execution order: this suggests a match with visualisations that make use of control flow. The paper also suggests that additional or supplementary representations may be useful given that differing representations can highlight different information.

Petre (1995) argues that graphical languages are not universally superior to text, noting that graphical representations require the user to adopt an inspection strategy, as well as linking the interface to specific domain knowledge. Secondary notation requires the emphasis of some information at the expense of others, leading to design trade-offs. Petre notes that experts use tools in a way which is attuned to the underlying structure being represented whereas novice users, working without this context, benefit from a more constrained environment.

2.10.3 Petri Nets

Petri Nets, developed from the work of Carl Adam Petri (1962), allow for the graphical modelling of concurrent, asynchronous, distributed, parallel, nondeterministic, and/or stochastic systems (Longo, 2000)—see Figure 2.18 for examples of simple primitive structures (Chen, 2003). Petri Nets have been used by a number of authors (such as Haus and Sametti, 1991; Baratè, 2008; Baggi and Haus, 2013) for the formal representation and structural description of music. Baratè et al. (2005) extended the standard Petri Net to better meet the requirements of music analysis and music making; the enhanced system is known as Music Petri Nets (Music PNs). The enhancements proposed by Baratè et al. (2005) allow for nondeterminism via conflict, in which one transition prevents a second from occurring; the ability to replace complex Petri Nets by a simple term, allowing for recursion; temporisation to allow for concurrent processes and synchronisation; and probabilistic weight to exert control over nondeterminism. In Music Petri Nets, places can contain *music objects* which can contain any type of musically-meaningful content. Transitions can be used to transfer tokens between objects, but can additionally transform the contents of a music object; such transformations can include transposition, inversion, or retrogradation. Section 5.2.2.1 presents an overview of Petri Nets and Music Petri Nets.

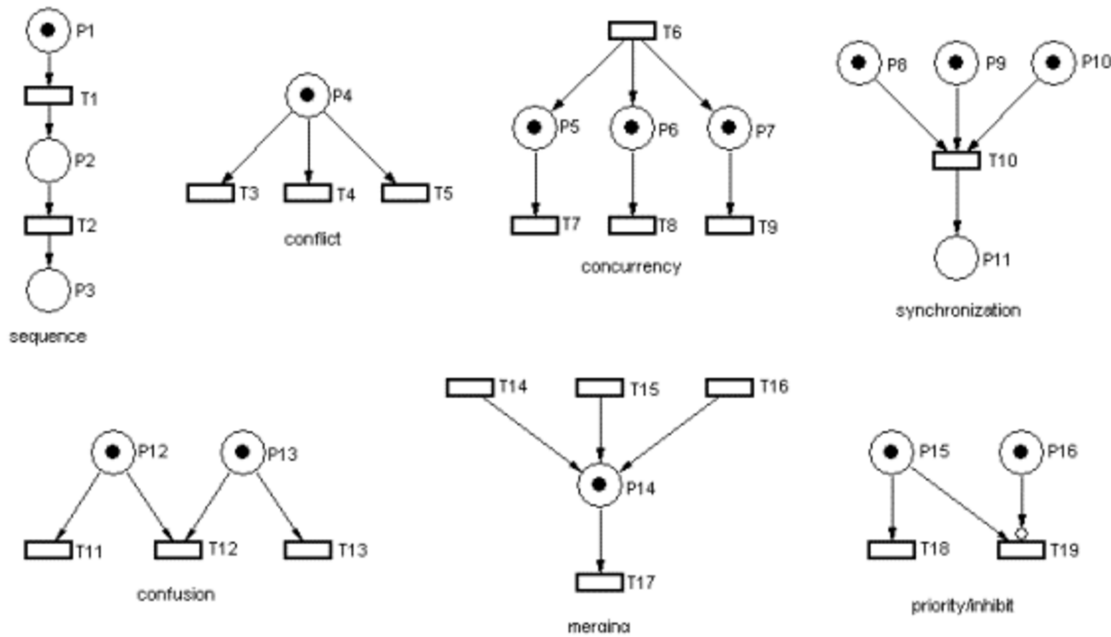


Figure 2.18: Primitive constructs in Petri Nets, showing a series of common structures (Chen, 2003).

2.11 End-user development

The research questions in Section 1.2 focus on the design of tools for novices that allow for a range of musical activities. Section 2.7 considered the trade-off between expressivity and usability, and Section 2.8 considered this trade-off in the context of a range of algorithmic composition tools. The findings suggested that interfaces with a limited set of controls can be highly usable but that this typically limits expressivity (Section 2.8.1). They also found that programming languages may allow for significant flexibility but at the expense of usability for novices (Section 2.8.10). It is sensible, therefore, to briefly consider end-user software which is designed to be both flexible and customisable.

End-user development (EUD) is a process by which software is written to allow non-programmers to create, modify, or extend software to better meet their own needs (Lieberman et al., 2006). Such users need to act like programmers even though they do not have comparable experience or knowledge (Blackwell et al., 2019), leading to a shift in the requirements of software from easy-to-use to easy-to-develop (Paternò and Santoro, 2019). Such software offers flexible configuration for users who are not software developers but who are experts in other domains and whose work is supported by computation (Paternò, 2013). The most successful use of EUD software seems to require a balance between the user's motivation in modifying the tools; the effective design of appropriate tools; and management support (Fischer et al., 2004).

Meta-design (Fischer et al., 2004) suggests a framework for EUD work in which open, 'under-designed' software is provided to end users. The users, as the 'owners' of specific problems, act as co-designers of the software which provides the solution. In an interesting parallel to the usability/expressivity continuum by Repenning and Ioannidou (1997), shown in Figure 2.4, Fischer et al. (2004) map the scope of an application against the learning costs (see Figure 2.19).

In a related paper, Ko et al. (2004) suggest six learning barriers that users of end-user programming systems typically face: design ('I don't know what I want the computer to do'), selection ('I think I know what I want the computer to do but I don't know what to use'), coordination ('I think I know what things to use but I don't know how to make them work together'), use ('I think I know what to use but I don't know how to use it'), understanding ('I thought I knew how to use this but it didn't do what I expected'), and information ('I think I know why it didn't do what I ex-

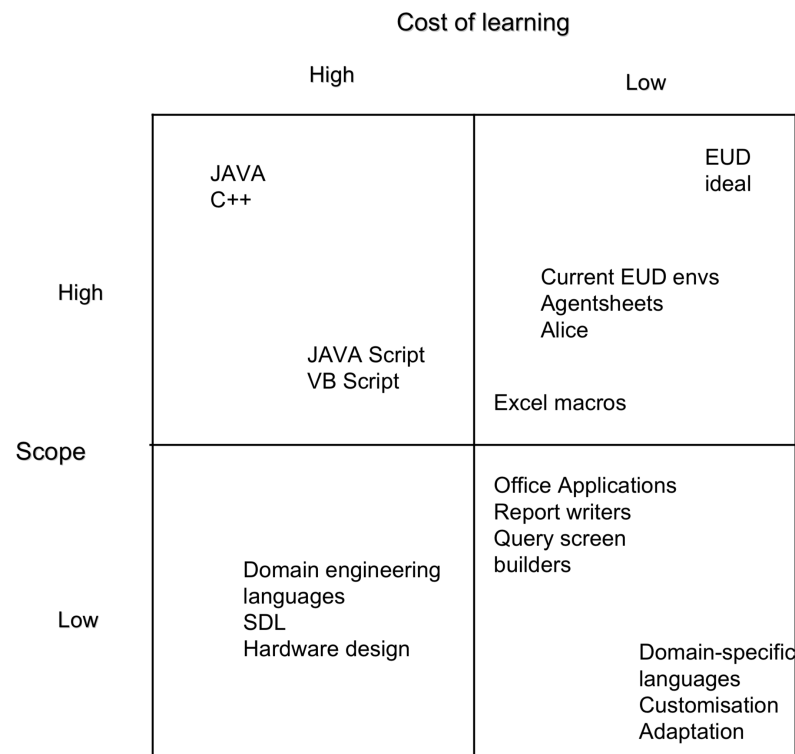


Figure 2.19: Cost-scope trade-offs in End-user development (EUD) tools (Fischer et al., 2004). SDL stands for Specification and Description Language.

pected but I don't know how to check'). To address these barriers they propose the use of a design metaphor to allow the end user to apply analogical reasoning from a source domain to the target domain. Figure 2.20, taken from Ko et al. (2004), shows a range of computational metaphors plotted against concreteness and human-centricity. Note that none of the listed metaphors relate to human involvement. The dashed box represents the suggested ideal point at which human-centric metaphors would be abstract enough to describe a variety of systems while being sufficiently concrete to support analogical reasoning.

EUD is similar to, but distinct from, software appropriation. Appropriation refers to the ways in which users adopt and adapt software in practice in order to 'make it theirs' (Dourish, 2003). As such, appropriation may result in end-users finding surprising new ways to use software—uncovering 'latent affordances for use in new contexts' (Anacleto and Fels, 2013). Tchounikine (2017) describes users as 'not simply passive recipients', describing how they adapt both their behaviour and the technology. In this way a loose parallel can be drawn between appropriation and the interplay between musician and notation (see Section 2.2).

Spreadsheets were one of the first successful EUD environments, showing program 'objects' as occupying cells in a grid. The spreadsheet objects interact with objects in neighbouring cells (Paternò, 2013). The tabular nature of spreadsheets allows for the generation of a template; this provides cues for the user to show what should be entered, thus reducing the mental load on the user when compared with the use of textual languages. Textual programming languages require the programmer to work in a given syntax, whereas spreadsheets allow the end user to enter data into cells without having to remember the order in which information should be entered, or the delimiting syntax to be used. A particularly relevant example is *Manhattan* (Nash, 2014), outlined in 2.8.6, an algorithmic composition tool which merges spreadsheet-style EUD capabilities (via the programmatic manipulation of relationships between cells via formula editing) with tracker-style functionality.

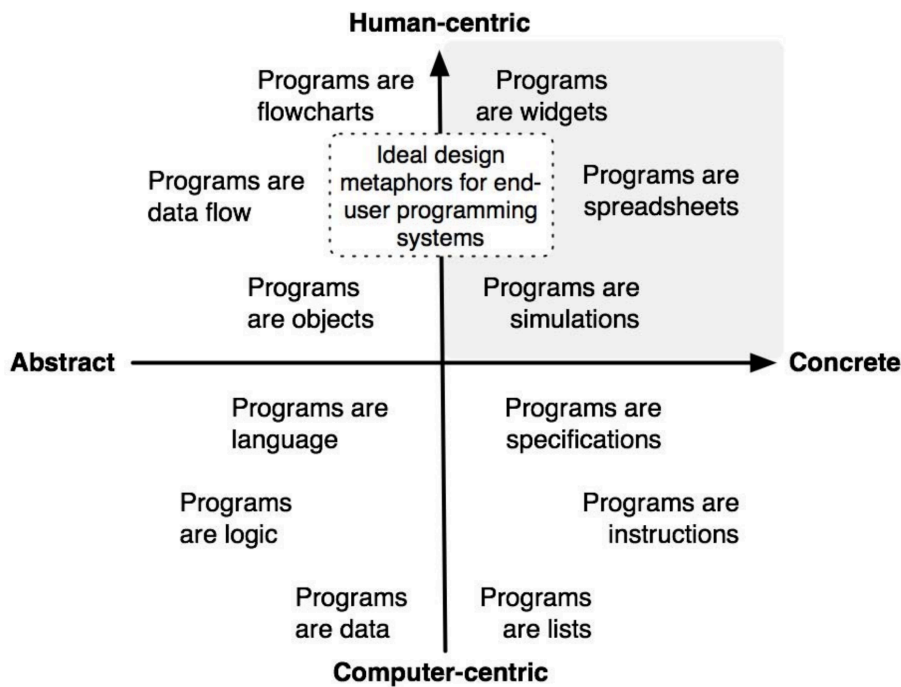


Figure 2.20: The choice of a suitable metaphor to support users of end-user programming systems (Ko et al., 2004). Ideal metaphors would be sufficiently abstract to describe many systems while being concrete enough for analogical reasoning.

2.12 Progressive disclosure

Given that the focus of the thesis is designing for novices, it may be suitable to present a simple interface to users while allowing them to access more powerful tools when appropriate. A useful principle to consider, whatever the details of the approach ultimately chosen, is progressive disclosure.

In the context of reflecting on the design of the Xerox *Star* interface, Kimball and Harslem (1982) describe progressive disclosure as showing only those properties which relate to the currently selected object. Controls which allow for greater complexity are hidden until they are needed (1982). Similarly, Nakatani and Rohrich (1983) describe progressive disclosure in software as displaying only those controls relevant to the current task. They suggest a parallel with hardware upon which infrequently used controls are sometimes hidden behind panels—examples may include thermostats, boilers, or video recorders. In contrast, software controls can appear and disappear in order to avoid overwhelming the user.

Progressive disclosure offers benefits for all users, but novice users in particular may benefit from minimising the choices required when they begin to use a piece of software. The progressive disclosure of options helps to prioritise the user's attention (Nielsen, 2006). Limiting the number of options can allow an interface to tersely express its meaning, and may reduce the error-proneness of the software. By displaying only the most relevant elements the design may reduce the hard mental operations required of the user. Visibility and progressive evaluation can be improved, and viscosity reduced, by limiting the information presented.

Nielsen (2006) outlines the need for initial and secondary features to be identified as part of the design process, and for the method by which the user will progress from one to the other to be made clear.

2.13 Conclusions

This chapter has considered the effects on music of common music notation and linear recording, before highlighting the distinction between reified linear recordings and the output from non-deterministic algorithmic composition systems. The chapter has considered software designed for the creation of both linear recordings and algorithmic music and has explored the literature on the trade-off between usability and expressivity, Direct Combination, diagrammatic representations, end-user development, and progressive disclosure. An analysis of a representative selection of algorithmic composition software is presented in Chapter 4 and the findings are used to inform the design work outlined in Chapter 5. Direct Combination and Petri Nets are used in the design exercises outlined in Chapter 5, informing the development of the preliminary version of Choosers outlined in Chapter 6.

Chapter 3

Methodology

This chapter introduces the key methodological approaches used in this thesis, principally Research through Design and programming walkthrough, an adaptation of cognitive walkthrough. The chapter also outlines the Cognitive Dimensions of Notations, used as an initial step to identify design issues.

Given the focus on the needs of non-programmers and the desire to offer levels of control not normally available to users without programming experience (see Section 2.7), an appropriate methodological approach is one in which the identification of issues, and the learning of practical and theoretical lessons from iterative design, is the focus. The purpose of the work presented in this thesis is not to develop a definitive interface but is instead to explore the issues and problems surrounding several possible solutions. An appropriate framework for helping to shape such an approach is Research through Design (Gaver, 2012), to be outlined in Section 3.2. Under the umbrella of Research through Design I use an adaptation of cognitive walkthroughs (Section 3.3), termed programming walkthroughs (Section 3.4), as a more specific methodology to identify problems in a rigorous way. These walkthroughs are carried out using a Wizard of Oz (WOz; see Section 3.6) simulated front-end and a fully implemented back-end, using the walkthrough protocol outlined in Section 3.5. As an initial step, the Cognitive Dimensions of Notations (Section 3.1) are used to identify design issues and trade-offs in a representative selection of software capable of algorithmic music composition.

Section 2.8 in the previous chapter showed that there are a number of inspiring related projects which make use of a variety of abstractions such as trackers, spreadsheets, blocks, piano rolls, and patch cables, but these are clearly not the only ways to address the research questions discussed in Chapter 1. Given that the target user is not a programmer it is particularly useful to investigate approaches that give the power of programming without the need to engage with complex textual syntax. The nature of the target users, as shown in Chapter 7 and Chapter 9, is that they are unlikely to have knowingly engaged with algorithms in the context of music creation. These users may not have previously encountered any algorithmic music tools and so do not, as yet, know anything about the topic. The notation presented in this thesis aims to help such users engage with a narrow set of algorithmic music processes while minimising the problems and issues that may otherwise occur. The users are likely to be using the notation, at least in part, to learn experientially about nondeterministic music-making. In this situation, methodologies such as measuring the time taken to write a particular piece of code will not offer useful insight. Instead, a talk-aloud walkthrough is a suitable methodology as it is more likely to draw out relevant insight from the participants.

3.1 Cognitive Dimensions of Notations analysis

Given the research aim of enhancing the usability and musical expressivity of nondeterministic composition software for non-programmers, the trade-offs inherent in interface design need to be

understood. The Cognitive Dimensions of Notations (CDN) framework is useful to us as it offers a vocabulary to explore the design space and tools to critique the software. It is important to get some understanding of the existing design space of nondeterministic and algorithmic composition software. This software is heterogeneous and varies on many dimensions. However, it is useful to identify some of the design issues and design trade-offs that arise in the different approaches to this problem, and to get a broad idea of the different categories of approach.

Cognitive Dimensions of Notations (Green, 1989; Green and Petre, 1996) are design principles for notations, user interfaces and programming language design, or from another viewpoint ‘discussion tools’ for designers. The authors stress that the CDN framework is not a comprehensive analysis tool but instead can be used as a ‘lens’ through which the usability of software can be predicted. The dimensions are useful in considering the trade-offs between various elements (Church and Green, 2008). The CDN framework considers the ways in which people can interact with information artefacts (such as software or hardware)—interactions include searching, incrementing, modifying structure, transcribing, exploratory design, exploratory comprehension, and comparison (Church and Green, 2008). The various dimensions allow for a specific aspect relating to usability to be considered. In a work related to this thesis, Nash (2015) uses and adapts the CDN for the purpose of designing and analysing notations in digital and traditional music study and practice. The paper applies the CDN in the analysis of both formal and informal common music notation (scores and sketches), a visual programming language (Max), and sequencer/DAW software. Nash (2015) formed the basis of a paper by Hunt et al. (2018) which used the CDN framework to consider an ideal usability profile for a prototype system which integrates generative music elements into a score editing workflow.

In order to bring some kind of discipline to such an analysis but not to prejudice what kinds of issues might be important, and to guide the development of the first design, a CDN analysis of a representative sample of systems was undertaken. The analysis resulted in a long paper published as a Technical Report (Bellingham et al., 2014b) and as a short paper presented at PPIG (Bellingham et al., 2014a). The complete CDN analysis is included in Chapter 4, with the resulting initial set of heuristics listed in Section 5.1.1 in Chapter 5.

3.2 Research through Design

The range of tools shown in the previous chapter (see Section 2.8) and in the CDN analysis in the next chapter suggests that the design of nondeterministic music software is what Rittel and Webber (1973) term a ‘wicked’ problem; that is, a problem for which there is no *a priori* answer or solution, and which cannot be understood without knowing about its context. In contrast to the natural sciences, which typically converge on a single concept or paradigm (Kuhn, 1970; Collins, 1994), Zimmerman et al. (2010) suggests that ‘wicked’ problems benefit from a process in which possible solutions gradually emerge. One approach to such open-ended problems is to create a tree of options in order to quickly explore a range of approaches. My judgement, based on my experience as a musician, composer, music producer, and educator, can be used as a heuristic guide while quickly iterating through a number of draft sketches before producing candidate designs for participant review and discussion. A suitable methodological framework for this is Research through Design.

Research through Design (Frayling, 1993) refers to the application of design practice in a given field, with the resulting designs embodying the designer’s judgements and solutions to the specific issues in the space. The designer’s work is not intended to be definitive; rather, it may represent one of many possible solutions. The output of the work can variously take the form of the designed artefacts and systems, methods, and/or theoretical and conceptual insights and frameworks. These outputs, or design objects, are knowledge producers both for the users who encounter them and the designers of the objects (Bardzell et al., 2015). The design artefacts and systems are sometimes, but not always, linked with accounts of field tests (Gaver, 2012). Theory emerges from the design process; William Gaver (2012) describes theory as an ‘annotation’ of the design: an object exists in

the design space and theory can explain the decisions that are embodied in the work. An annotated portfolio represents an area in the design space and may therefore make the design domain and the potential dimensions clearer.

Gaver (2012) considers the potential benefits that Research through Design, as a generative discipline, offers. He suggests that design can lead to ‘multiple new worlds’, rather than describing a single existing one. The designer’s work is not intended to be definitive; rather, it represents one of many possible solutions which may be incompatible with each other. In Gaver’s words, the goal of Research through Design ‘is not to develop theories that are never wrong, it is to create theories that are *sometimes right*’ (2012).

The methodology presented here includes an extended iterative design phase which is informed by participant programming walkthrough evaluations (to be outlined in Section 3.5). There is a trade-off between spreading finite development efforts between multiple design and evaluation iterations or investing all efforts into fully developing and formally evaluating a single design, and both approaches have merits. The latter approach is suited to a design space which is well understood. However, given this design space (nondeterministic composition tools for non-programmers) is relatively unexplored, it is arguably more valuable to create and quickly evaluate multiple iterations in order to improve insight. In order to facilitate this approach, participants will interact with a fully implemented back-end via a simulated front-end (see Section 3.5; Section 3.6), allowing for the collection of user feedback. Insight into the real-world applicability of Choosers was sought via interviews with domain specialists, presented in Chapter 11.

Feedback from participants can be collected once an initial design of the notation has been finalised. Given that the identification of the most suitable abstractions and representations for the target user group was prioritised, it was decided that more iterations could be completed in a given timeframe by using a fully implemented back-end and a fully deterministic hand-simulated user interface rather than building a prototype for each iteration. A suitable methodology for this is a programming walkthrough, which is an adaptation of a cognitive walkthrough; both walkthrough types will now be explored and contextualised.

3.3 Cognitive walkthrough

A cognitive walkthrough is an adaptation of the design walkthrough, which is a so-called ‘hand simulation’ in a particular domain: examples include architecture walkthroughs, document walkthroughs, and code walkthroughs (Yourdon, 1989; Fagan, 2001). In contrast to a design walkthrough, a cognitive walkthrough is designed to be applied to software interfaces for end users, with a focus on ease of learning. It is a ‘hand simulation of the cognitive abilities of the user’ (Polson et al., 1992), designed to consider how easily the user can learn to perform the tasks that a piece of software has been designed to support.

A cognitive walkthrough is accomplished by performing tasks indicative of those the end user will undertake (Lewis et al., 1990). The steps required, and the match between user expectation and software performance, are noted (Jeffries et al., 1991). In this way, a cognitive walkthrough allows the designer to make informed inferences about an interface’s usability by better understanding the roles of the user’s mental processes (Lewis et al., 1990; Polson et al., 1994). The process can be administered quickly, allowing it to form part of a fast iteration cycle. Importantly, the cognitive walkthrough methodology is based on cognitive theory surrounding learning by exploration (Polson and Lewis, 1990; Norman, 1986, 1988), and evaluates the ability of a user to perform a task with little or no formal instruction (Polson et al., 1992).

Given that the focus is not on user interfaces in general but is instead on designing nondeterministic composition tools for non-programmers, there is a need to understand how writable the notation is for this class of user: how easily can the participants create music using the notation? The next section will introduce the programming walkthrough, an adaptation of the cognitive walkthrough,

which focusses on the ease of programming rather than the ease of learning.

3.4 Programming walkthrough

A programming walkthrough is an adaptation of the cognitive walkthrough with a specific focus on the *writability* of a programming language (Bell et al., 1992). This is done by requiring a description of the steps required to write a program, and of the guiding knowledge required to select these steps. The design can then be examined by considering the number of steps (e.g. can this be reduced?) and the knowledge required for each step (e.g. does a step require unspecified or extensive knowledge?).

Recall that we are interested in users who are using a novel notation while learning experientially about a subset of algorithmic music composition techniques. Rather than focus on speed or efficiency, the participant studies will ideally have an explicit focus on the mental, rather than physical (e.g. typing), steps required to create music using the notation. Programming walkthroughs are an excellent methodological match as they can, in part, be used to identify the knowledge required to guide a user's decisions (Bell et al., 1992). By working in pairs, the self-reporting of the participants is improved as it is articulated by two people who are able to collaborate with the walkthrough facilitator on the identification of questions, problems, suggestions, and other observations. These responses will then be used to guide further iterative development of the notation.

3.5 Walkthrough protocol

The participants were selected from a pool of self-taught music producers with experience of making music electronically using music sequencers and DAWs (see Sections 2.4.2, 2.4.3). These practitioners were targeted as they make music using software while typically being non-programmers; as such they are part of the target demographic identified in the research questions listed in Section 1.2. This class of participant are often not traditional musicians (i.e. they are unlikely to read manuscript, they may not play a traditional musical instrument), although they may be conversant with some elements of music theory.

All participants were asked to complete a short questionnaire before taking part in the user tests. The questions, listed below, are focussed on the participant's experience in playing musical instruments, reading music notation, using music software (e.g. DAWs), performing in a collaborative environment, programming, and writing algorithmic music.

- Do you play any instruments? If so, please tell us which instrument(s) you play.
- Can you read any music notation?
- Are you familiar with any Digital Audio Workstation, sequencing, or synthesis software? If so, what?
- Do you have experience making music using other technology?
- Do you perform with other people? If so, please tell us about it.
- Have you done any computer programming? If so, please list the languages you have used.
- Have you written algorithmic music in the past? If so, please outline your experience in creating algorithmic music.

Pairs of participants were asked to take part in a series of practical tasks based around clearly defined scenarios; these scenarios are detailed in Chapter 7 and Chapter 9. The participants worked on paper or on a whiteboard, and their outputs were played by the facilitator using a set of *SuperCollider* (McCartney, 2002) classes written to implement the musical abstractions behind the system. In order for the facilitator to provide near-instant playback as part of the Wizard of Oz technique (see Section 3.6), a series of templates had been previously prepared and were edited during the sessions to match the participant sketches.

At the end of each walkthrough, the pairs of participants were asked the following formative questions:

- Can you see anything this would be useful for?
- Can you see any ways in which this is similar to other tools you have used?
- Is there anything that is made easier by this system? Anything which was not possible made possible/hard and made easier?

3.5.1 Analysis

The participants in each pair were free to discuss the work and to ask for clarification with the facilitator of the test. Users were asked to act as active participants in the research, and to help in categorising any issues that were raised. As a result, each walkthrough had multiple participants who reached a consensus on how each of their contributions should be categorised. The categorisations that the participants were asked to use—taken from the programming walkthrough method (Bell et al., 1991, 1992) outlined in Section 3.4—were as follows:

- Questions (e.g. why does the loop do that?);
- Problems (e.g. I don't understand what these lanes are for);
- Suggestions (e.g. maybe the cone should be a different shape);
- Other observations (e.g. I like the fins).

In addition, participants were asked if they could think of any other ways in which each scenario could be completed. This prompted a discussion on alternative routes in order to test understanding and to capture user expectations.

The user tests were videoed and exhaustively transcribed to assist in analysis. Each transcript was then annotated in line with the programming walkthrough methodology to show the questions, problems, suggestions, and observations of the participants; the categorisation of each point was agreed with the participants during the walkthroughs. An example user test transcript, showing analytical markings, can be seen in Appendix C. Recurring themes were extracted and form the basis of detailed reflections on the design issues presented in Section 7.5 of Chapter 7 and Section 9.3 of Chapter 9. These themes then lead into the changes made in the next design iteration. Drawing on Research through Design, each set of iterative changes offer one of many possible solutions to a complex problem; they are not intended to arrive at a single definitive solution.

A programming walkthrough assists the the designer in identifying both the sequence of necessary steps and the knowledge required to solve a given problem. Analysis can consider the programming language characteristics which can enhance the ease of programming, as identified by Bell et al. (1992, 1994). These include the adoption of proven representations, such as algebraic formulae; judgement from experience with previous language designs, in which the language designer provides features from outside of the domain; consideration of the benefits of abstract characteristics (such as simplicity and orthogonality) to make the program easier to write; and attention to the process of writing programs by considering ways in which the language could best benefit those who will use it.

Bell et al. (1994) notes that there are several ways of performing a programming walkthrough. One method is when a designer works alone, which can produce useful insights. However, the end-user studies presented in Chapter 7 and Chapter 9 make use of an enhanced programming walkthrough in which groups of 'outsiders' work with the designed artefact, with the designer acting as both observer and facilitator. Bell et al. (1994) found that 'outsiders' can perform a more complete analysis than the designer working alone as they may not skip steps that the designer may consider obvious or uninteresting, and they may detect alternative solution paths that the designer may have overlooked.

Programming walkthroughs offer a clear methodology to follow which can help to mitigate some potential limitations; see Section 3.5 for the walkthrough protocol and Section 3.8 for a full discussion regarding potential methodological limitations. A range of rich data will be captured, including full video, audio, and exhaustive annotated transcripts of all participant walkthroughs

and interviews, all user-defined Chooser parameters added to the back-end code and run during the participant sessions, and all participant-created designs and resulting audio.

3.6 Wizard of Oz

As outlined in Section 3.4, the proposed methodology makes use of a series of programming walkthroughs to gather input from participants. The decision was made to run the walkthroughs using a Wizard of Oz simulated front-end with a fully implemented back-end. This decision was made for a two reasons. Firstly, given that the focus is to explore the design space and develop suitable abstractions and representations for the target users, limited development resources could be focussed on the collection of rich data which could be captured without a fully-implemented front-end. Secondly, more iterations could be made quickly while the notation's design remained fluid and likely to change.

In general, the Wizard of Oz technique can conceal potential problems (such as missing the impact of a particular interaction which has not been simulated) and can make incorrect assumptions regarding the actual implementation of the tasks being simulated (Kelley, 1983). In order to negate this risk, the translation of the paper or whiteboard sketches to parameters on the back-end had no room for interpretation. The facilitator had zero degrees of control, and all user input was played according to a strict algorithm with a 1:1 mapping between the user's drawings and fully-written code. Examples of participant-generated input and the corresponding *SuperCollider* code, edited during the user studies to enable participants to hear their work, can be seen in Figure 7.1 in Chapter 7, and in Figure 9.1 in Chapter 9. Importantly, while some authors reserve the term for situations where the participants think they are interacting with fully functional system and are unaware of the deception until after the test has completed (Green and Wei-Haas, 1985), Wizard of Oz is widely used in the literature to also refer to studies which are administered with the full knowledge of the participants (Li and Dey, 2013).

The Wizard of Oz technique, run with the participant's knowledge, was used in the two user studies outlined in Chapter 7 and Chapter 9. These experiments were carried out using a Wizard of Oz interface supported by a fully implemented back-end written in *SuperCollider*. Users were provided with clearly unfinished and 'messy' non-computational input methods—paper and a whiteboard—in order to highlight the malleable nature of the design at the user test stage. This emphasised the possibility of change to encourage greater and deeper user critique than may have been received with a more polished and 'finished' or 'closed' interface.

3.7 Protocol for interviews with experts

In order to address the third research question (what are the implications for music production, music computing education, and music composition?), a series of ethnographically-inspired, semi-structured interviews will be held with domain specialists in which the notation will be introduced, interrogated via manipulable compositions, and used as a prompt for discussion. The interviews are ethnographically-inspired in the sense that they seek to understand how Choosers may be perceived and interpreted by practitioners in distinct but interlinked communities of practice (Nurani, 2008). Where possible the interviews will take place in the interviewee's natural setting. The participants will be asked to give feedback on the potential uses for the language. The interviews are designed to explore potential uses and barriers in music composition, in music production, and in music computing education. The interviewees will be verbally introduced to the notation before being able to read, play, and modify example pieces of music; the participants will be free to ask clarifying questions throughout the interview. As with the earlier programming walkthroughs, these interviews will again make use of a Wizard of Oz simulated front-end with a fully implemented back-end, with the designer acting as facilitator. The fully implemented *SuperCollider* back-end

will ensure that the participant's input is performed algorithmically with no additional intervention. At the end of each interview the participants will be asked the following questions which have been designed to solicit feedback on the potential users, uses, and barriers for the notation.

- What kind of user would the notation work for?
- What can't the notation do? What is the notation not useful for?
- Are there interesting or useful comparisons with other tools that you use? Would the notation replace or augment them?
- Can you see uses for the notation? How could you use the notation as part of your research/education/performance practice?
- What barriers would you need to be overcome for that to happen?

As with the user studies outlined above, each interview was videoed and exhaustively transcribed. Each transcript was then analysed, with recurring broad themes identified and presented in Chapter 11.

3.8 Potential limitations of the proposed methodology

There are some potential risks when the designer takes a role in administering walkthroughs or undertaking analysis. These risks, and the steps taken to mitigate them, are shown in Table 3.1.

Table 3.1: Potential risks that may arise when a designer takes a role in administering walkthroughs or undertaking analysis, and the steps taken to mitigate them.

Risks	Mitigations
Any existing relationship between participant and designer/facilitator may risk biasing the results. Participants in the walkthroughs may modify or soften any criticism in their feedback if they are aware that the facilitator is also the designer (Hollingsed and Novick, 2007).	All participants in the first and second round of user studies were taken from a pool of new first-year students at the University of Wolverhampton who I had not worked with at the time of the walkthroughs. Despite working in the same or similar industries as I have in my career so far, the interview participants were not current or past colleagues. The walkthroughs were performed with strict adherence to the programming walkthrough protocols (see Section 3.4). All user test interactions were videoed and exhaustively transcribed to allow for a full review; an example is shown in Appendix C.

Risks	Mitigations
The Wizard of Oz technique may offer an opportunity for the designer/facilitator to knowingly or unknowingly intervene in the walkthrough process, potentially allowing for a modification of participant input to produce ‘best case’ results.	The back-end used in the user studies and interviews implemented a strict algorithm which does not offer any degrees of freedom; the participant work on paper or whiteboard is exactly mirrored in software. In order to accurately represent the intentions of the participants during the Wizard of Oz sections of the walkthroughs, each participant-created Chooser was photographed (see examples in Appendix C), with the <i>SuperCollider</i> code created in the walkthrough saved alongside the photographs. The audio from the walkthrough videos provides evidence of the accuracy of playback during each walkthrough.
Walkthroughs are vulnerable to poor control of the walkthrough procedure, such as facilitators forgetting steps in a given path, making assumptions about goals or steps, or failing to notice a link between a goal and a related action (Polson et al., 1992; Bell et al., 1994).	Clear scenarios and scripts were developed and adhered to during the walkthroughs as part of the methodology outlined in this chapter; the scenarios and scripts are outlined in full in Chapter 7 and Chapter 9. Exhaustive transcripts of all walkthroughs and interviews were created and analysed to surface links between goals and actions—see example Appendix C for an example annotated transcript.
A focus on a set of narrowly-specified tasks can be difficult to articulate and administer, and may not allow sufficient user agency, in the context of an open and creative environment such as music arrangement.	Scenarios were designed to move through basic use and arrangement, and to finish with a relatively open-ended compositional task which was constrained in scope by the use of a small pool of audio samples.
Participants in programming walkthroughs may have insufficient access to the facts required to allow them to understand the behaviour of the language being tested (Bell et al., 1991).	The walkthroughs made use of carefully structured tutorial videos, questions to check understanding, and simple implementation tasks to both introduce and embed key concepts and terms.
The participant’s real-world knowledge may interfere with the tasks they are being asked to undertake as part of the walkthrough (Bell et al., 1991).	Participants unlikely to have undertaken algorithmic music work before. Capturing information on technological framing clashes is useful in this context.

3.9 Conclusions

This chapter has presented the key methods to be used in the development and testing of a novel nondeterministic composition notation for novices. Research through Design was identified as a methodology which will enable the consideration of alternative solutions by presenting an annotated portfolio of multiple iterative designs. As an initial step, Cognitive Dimension of Notations are to be used to consider the usability trade-offs in existing software capable of algorithmic composition (see Chapter 4). Once developed, candidate designs will be interrogated via programming walkthroughs, administered via a Wizard of Oz technique, to better understand the knowledge and steps required to complete specific tasks (Chapters 7, 9) while allowing a focus on the design of the

language rather than on the cosmetic user interface. The findings from these tasks will be analysed and used as the basis for modification and further development.

Chapter 4

Cognitive Dimensions of Notations analysis

This chapter presents an analysis, using the Cognitive Dimensions of Notations (CDN), of a representative selection of user interfaces for algorithmic composition software. The findings from this CDN review were used to identify candidate design principles for this project.

This chapter presents an analysis, using Cognitive Dimensions of Notations (Green, 1989; Green and Petre, 1996), of a representative selection of user interfaces for algorithmic composition software. Cognitive Dimensions are design principles for notations, user interfaces and programming language design, or from another viewpoint ‘discussion tools’ for designers (Green and Blackwell, 1998). Expanding on the characterisations introduced in Chapter 1 and Chapter 2, for the purposes of this thesis algorithmic composition software is considered to be software which generates music using computer algorithms where the algorithms may be controlled by end users (who may variously be considered as composers or performers). For example, the algorithms may be created by the end user, or the user may provide data or parameter settings to pre-existing algorithms. Other kinds of end-user manipulation are also possible. A wide variety of algorithmic composition software is considered, including visual programming languages, text-oriented programming languages, and software which requires or allows data entry by the user. The chapter considers a representative, rather than comprehensive, selection of software. The analysis also draws, where appropriate, on related discussion perspectives and tools drawn from Crampton Smith (Moggridge, 2006), Cooper et al. (2014) and Rogers et al. (2019). Finally, the chapter reflects on the compositional representation of time as a critical dimension of composition software that is not satisfactorily addressed by Cognitive Dimensions.

The CDN has been used to review music systems in a number of previous studies. For example, Blackwell et al. (2000) make use of the CDN in a review of a range of music notation, including common music notation and typical sequencer editors (such as a piano roll, event list, and staff notation). In a paper related to the work presented here, Nash (2015) uses the CDN to assess the usability of three notations/interfaces previously addressed in Chapter 2; common music notation, including formal scores and informal sketches (Section 2.2), music programming languages such as Max/MSP (Section 2.8.8), and user interfaces offered by mainstream sequencer/DAW software (see Section 2.4.2; Section 2.4.3).

4.1 Introduction

The chapter considers fourteen Cognitive Dimensions of Notations in turn, using each dimension to illuminate design issues in a wide variety of user interfaces for algorithmic composition. Finally, issues in interaction design for algorithmic composition that are arguably not well addressed by the CDN, such as the compositional representation of time, are considered.

4.2 The Cognitive Dimensions of Notations

4.2.1 Viscosity

Viscosity is a measure of the software's resistance to change, or how easily the user can change a patch or specification once it has been written. *Repetition viscosity* is caused when the software requires several actions to achieve a single goal. *Knock-on viscosity* is created when a user makes a change that requires further action to restore consistency (Green and Blackwell, 1998).

Algorithms encoded in visual patching languages such as *Max* (Cycling '74, 2019) and *Pure Data* (Puckette, 1997) can become highly viscous. Connections are made with virtual patch cables, and when there are many connections the patch can become difficult to read and work with (sometimes referred to as 'spaghetti code'; an example is shown in Figure 4.1).

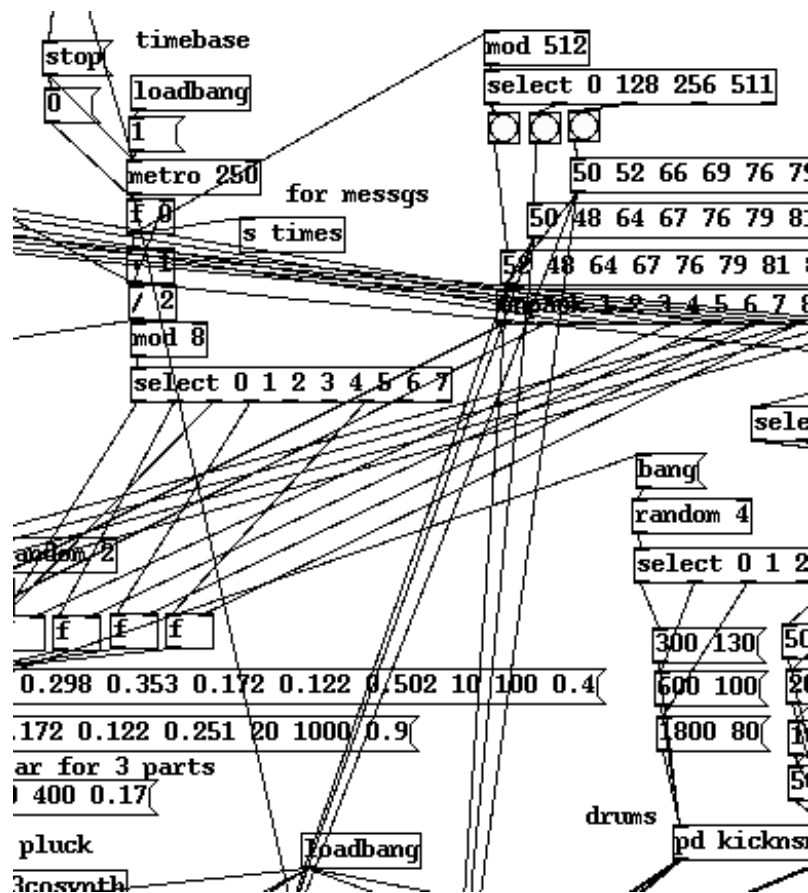


Figure 4.1: Dense connections in *Pure Data* (Farnell, 2008), creating 'spaghetti code' which increases viscosity.

The use of subpatches can ameliorate some viscosity issues. Subpatches allow the programmer to create objects with inlets and outlets which can abstract and simplify an entire patch to a single object. A subpatch need only be defined once, after which an instance can be used wherever the same material recurs, further reducing repetition viscosity. Abstracted connections ('send' and 'return' objects, for example) allow for non-visual connections, potentially reducing visual 'spaghetti', but at the expense of increased visual search cost (see 'hidden dependencies' in Section 4.2.3).

The structure of a piece of music can be a crucial element of the composition, potentially at several levels. If the genre requires a strongly structured piece there may be a conceptual structure at a given level (ABA, for example). An interface which allows the user's conceptual structure to be mirrored will reduce both repetition and knock-on viscosity. More specifically, Cooper et al. (2014) suggest three separate models for the perception of software; the *implementation model* of

the software (how it works), the user's *mental model* (how the user imagines the software to work) and the *representational model* of the software (what the software shows the user). Repetition viscosity could be significantly reduced by better matching the mental model to the representational or implementation models. For example, if the implementation model implemented repeats by repeated references to the same object, then a single change to the object would be reflected in each repetition. If the representational model in turn showed repetition as repeats of a single visual block, the user could input the desired changes once and they would be propagated to each repetition (see *provisionality* in Section 4.2.13).

Some text-oriented programming languages such as *Csound* (Vercoe, 1996) and *SuperCollider* (McCartney, 2002) allow the user to create elements that can be easily reused and redefined. This can reduce repetition viscosity as changes in the variable's parameters will cascade to all instances. The 'rules' implemented in *Noatikl* (Intermorph, 2015b) allow for a reduced viscosity via cascading changes in a similar way. On the other hand, knock-on viscosity can be unintentionally created by introducing wide-ranging changes in this way.

There are links between viscosity and another of the dimensions, *premature commitment* (Section 4.2.4). Premature commitment refers to situations in which the user has to make a decision before they have access to all relevant information. A piece of software with high viscosity makes it hard to amend work once it has been started. It is reasonable to assume that many compositions start with sketches that develop; high viscosity, such as is found in visual patching software, makes such development difficult.

Software with low expressivity could be viewed as exhibiting high viscosity, but this is not necessarily a negative attribute. Highly viscous software can in effect present a user with a single, stable, well defined use-case. An example of this is *WolframTones* (Wolfram Research Labs, 2005) which presents the user with a limited control set. Another example is *Improviser for Audiocubes* (Percussa, 2012), in which the complexity of the performance is generated by the physical layout of the Audiocubes (Percussa, 2013).

Live coding (editing code while it generates sound) requires software with a low viscosity. Elements need to be individually controllable with an efficient syntax, low repetition viscosity and minimal knock-on viscosity. An example of such a design is *Tidal* (McLean and Wiggins, 2010; McLean, 2014), a computer language for encoding and manipulating musical patterns during improvised live coding performances. Similarly, *Impromptu* (Sorensen, 2010) is a Lisp-based environment which allows for the real-time creation and control of objects. *Usine* (Sens, 2013) has a GUI which allows for mouse and keyboard control, with a graphical signal flow based on a modular synthesis concept.

4.2.2 Abstraction

Abstraction can be used to present the user with representations which are closely aligned with the semantic meaning of the corresponding entities. The implementation of an entity is hidden, or *abstracted*. The *abstraction barrier* (Green and Blackwell, 1998) describes the number of abstractions the user must master before using the software. Green and Blackwell describe three classes of software; *abstraction-hungry* systems which require user-defined abstractions, *abstraction-tolerant* systems which permit them, and *abstraction-hating* systems which do not allow them (1998).

Many of the text-oriented languages used in the space, such as *Impromptu* (Sorensen, 2010) and *SuperCollider* (McCartney, 2002), are abstraction-hungry. They frequently also have a high abstraction barrier in the sense that they require the user to learn the syntax and abstractions used. An example of abstraction-tolerant software is the *Algorithmic Composition Toolbox* by Paul Berg (2014), which presents objects to the user and allows the creation of new abstractions. The software makes use of musical metaphors (such as a rudimentary piano roll) for some elements. The user defines objects such as sections, shapes, masks and note structures which the program's 'generators' use to create new material. Some algorithmic composition systems do not require the user to interact with the method of generation; these are abstraction-hating systems. Robert Walker's *Fractal*

Tune Smithy (2011) and Jonathan Middleton's *Musical Algorithms* (2004) both require musical input which is then acted on using algorithms which are both hidden from, and inaccessible to, the user.

There are diverse ways in which abstraction can be used to make software more effectively match the user's mental model (Cooper et al., 2014). For example, multiple steps can be combined to make the software conform to the user's expectations. One powerful type of abstraction makes use of metaphor, for example a metaphor based on the hardware controls of a tape machine, as seen in Figures 4.2, 4.3.

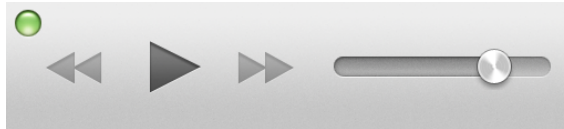


Figure 4.2: The transport controls in *iTunes* with playback stopped, showing the play button.

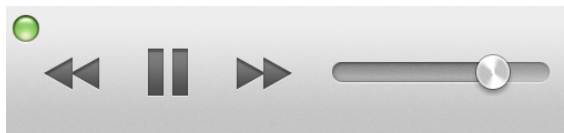


Figure 4.3: The transport controls in *iTunes* during playback, showing the pause button.

Other hardware metaphors are used in current algorithmic composition software. The *Cylob Music System* by Chris Jeffs (2010), shown in Figure 4.4, makes use of step-sequencer and drum machine designs, among others.

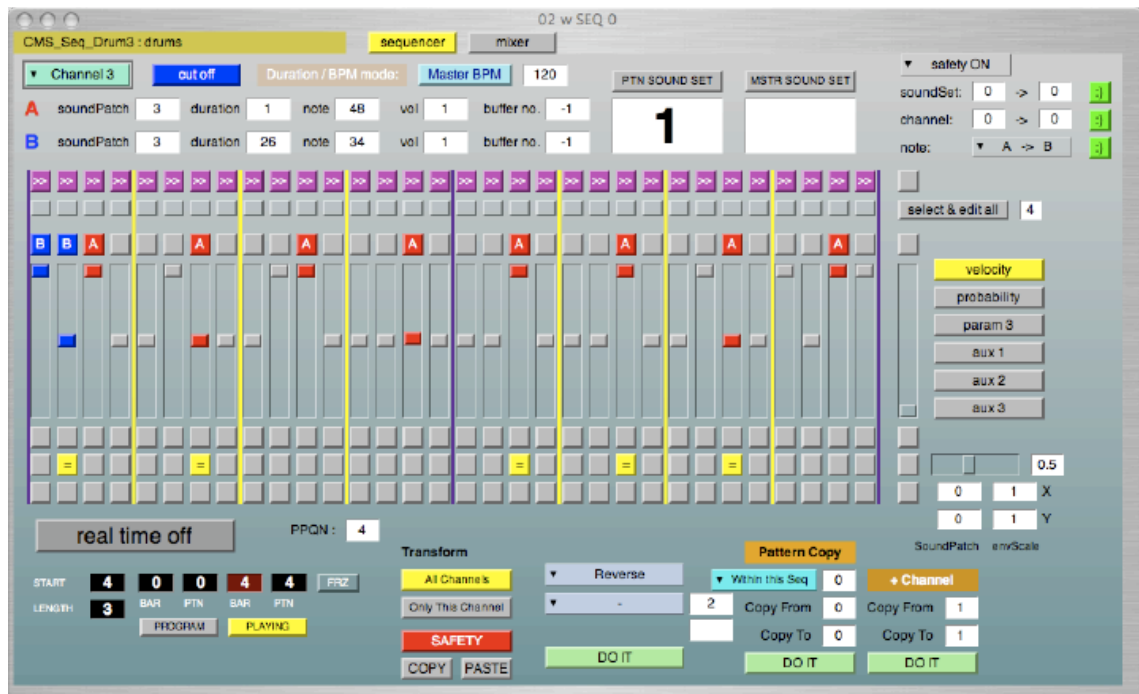


Figure 4.4: The drum machine interface from the *Cylob Music System* (Jeffs, 2010).

There are several music metaphors used in the software in the field which require the user to be conversant with music theory. *Harmony Improvisator* (Synleor, 2013) requires input in the form of scales, chords and inversions. Lars Kindermann's *MusiNum* (2006)—shown in Figure 4.5—requires the user to specify harmonic parameters. *Noatikl* (Intermorphic, 2015b) uses abstractions to cre-

ate what it refers to as ‘Rule Objects’ (‘Scale Rule’, ‘Harmony Rule’, ‘Next Note Rule’ and ‘Rhythm Rule’) to control how the software generates patterns. The *Algorithmic Composition Toolbox* (Berg, 2014) makes reference to note patterns and structures. Roger Dannenberg has explained how staff notation is rich in abstractions (1993); software which uses elements of staff notation is building abstractions on top of abstractions.

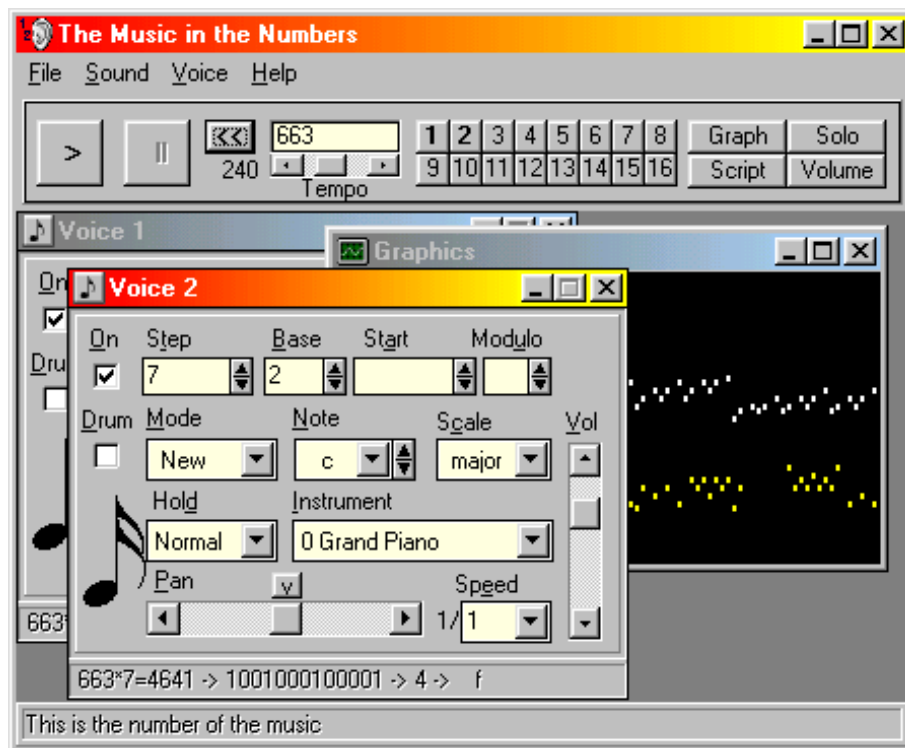


Figure 4.5: Lars Kindermann’s *MusiNum* (2006).

Having reviewed the software listed above, note that one of the design goals will be to keep abstraction barriers as low as possible relative to the required expressivity. Given the target users, metaphors which require the user to be conversant with music theory will be avoided.

Abstraction has a link to *visibility* (Section 4.2.6), another of the Cognitive Dimensions. A high level of abstraction can result in low visibility by increasing hidden dependencies. A design goal for the project is for the software to have a low abstraction barrier but be abstraction-tolerant. Such a design would allow new users to work with the language without writing new abstractions, while more advanced users could write abstractions when appropriate.

4.2.3 Hidden dependencies

Hidden dependencies occur when important links between entities are not fully visible. There is a *search cost* which reflects the effort required to locate the dependency (Green and Blackwell, 1998).

As seen in the case of *Max* in Section 2.8.8, in some systems objects can be connected via either one-way or symmetric links. One-way links can only send, whereas symmetric links can both send and receive. The patch-cable metaphor used in visual programming languages makes one-way dependencies explicit and reduces the potential for hidden dependencies. Visual audio programming systems typically use a patch cable metaphor and, perhaps as physical patching is generally unidirectional (i.e. audio send or return) rather than bidirectional (i.e. MIDI, USB) connection, software such as *Max* and *Pure Data* has opted for a one-way connectivity metaphor. Visual patching systems allow users to see links at the potential expense of increased premature commitment.

Both graphical and text-oriented languages can make use of variables and hidden sends and returns. If users are required to check dependencies before they make changes to the software

the search cost is increased. This in turn can lead to higher error rates (via knock-on viscosity). Abstractions can impose additional hidden dependencies; users may not be able to see how changes will affect other elements in the patch.

4.2.4 Premature commitment

Premature commitment refers to constraints on the order of operation, which leads to the user making a decision before all relevant information is available. *Enforced lookahead* describes how the user is forced to decide on implementation detail before they would otherwise be ready to (Green and Blackwell, 1998).

While experienced users can leverage their understanding of a piece of software to minimise premature commitment, less experienced users may have to rewrite a patch as it develops. Alan Cooper refers to ‘survivors’ (Cooper, 2004); those who manage to use software but find the process of composition is made more difficult by the limitations of the tool. He describes the cognitive dissonance caused by tension between the user’s mental model and the implementation and/or representational models (Cooper et al., 2014). The software can dictate the way the composer writes.

Much software in the algorithmic space allow links to be made only between pre-existing entities. In these cases the user is unable to say ‘I don’t know what is going here’, which can be a useful option when composing. One possible solution to this problem would be to decouple the design of the patch/composition from the actualisation. This could take the form of a graphical sketching tool which would allow the user to test the structure and basic design of the patch, as shown in Figure 4.6.

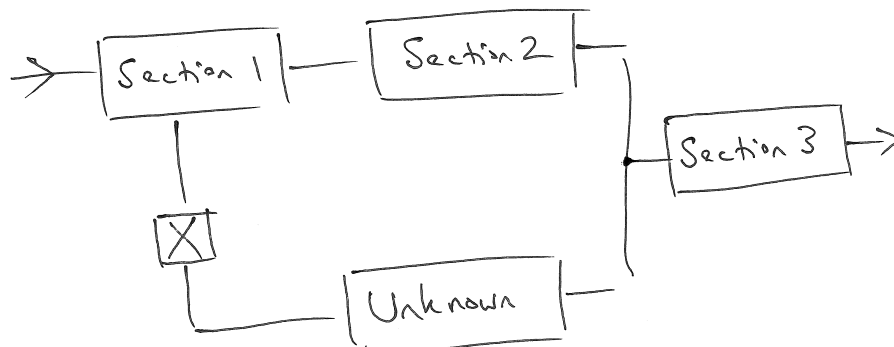


Figure 4.6: Potential layout for structure-aware composition software.

There is a link between premature commitment and the viscosity of the programme (Section 4.2.1); if viscosity is low then the user can redesign their patch with relative ease. Graphical systems such as *Noatikl* show the order of actions using a signal flow analogy which offers an easy way to reorder, thereby reducing viscosity.

Green and Petre (1996) introduce two subsets of premature commitment; *commitment to layout* and *commitment to connections*. Graphical systems such as *Max* encourage users to start with one element and then expand the patch. This forces the user to commit to the layout, and as the patch grows in complexity the viscosity of the software can limit subsequent changes. Graphical systems can be associated with a commitment to connections as significant planning is needed to design a flexible patch. A simple patch using a patch-cable analogy can be easily readable, but increased complexity can lead to a visually congested patch which is hard to maintain. This leads to higher viscosity.

4.2.5 Secondary notation

Secondary notation refers to extra information conveyed to the user in means other than the formal syntax (Green and Blackwell, 1998). Examples of secondary notation are in the collection of con-

trols in *Mixtikl* (Intermorphic, 2015a) and in the design of the *Cylob Music System*. Information conveyed by placement is known as *escape from formalism* (Green and Blackwell, 1998). The spatial placement of the controls adds to the information available, making the device easier to learn. One example is the size and placement of the filter controls in *Mixtikl* (Intermorphic, 2015a), shown in Figure 4.7.



Figure 4.7: Filter controls in *Mixtikl* (Intermorphic, 2015a).

Both *Max* and *Pure Data* allow for graphical elements (such as colour, fonts and canvas objects) to be added to patches. *Max* is particularly flexible when implementing secondary notation, as seen in the patches created by *Autechre*’s Rob Brown and Sean Booth (Tingen, 2004). An example is shown in Figure 4.8.

Another example of secondary notation is code indentation; Green refers to this as *redundant recoding* (1998). Indentation helps to improve legibility and comprehension when reading and writing code. Indentation is used in all of the text-oriented languages under review. Object-oriented languages (such as *SuperCollider*) and XML-based syntax—such as that used in *SoundHelix* by Thomas Schurger (2016)—make significant use of the placement or context of commands for both the readability and functionality of the code. Adding comments to code is another example of secondary notation, and all of the musical programming languages under consideration allow for commenting—Figure 4.9 shows comments in a *SuperCollider* patch.

There is a link between secondary notation and viscosity (Section 4.2.1). If a patch’s structure is changed the secondary notation (such as the placement of controls) can also be affected. The design of the patch therefore needs to include the required secondary notation which will lead to a higher viscosity due to the lack of flexibility in future changes.

4.2.6 Visibility

Visibility is the ability to view components easily. *Juxtaposability* is the ability to view two components simultaneously; this can be useful when comparing two elements (Green and Blackwell, 1998). There is a balance to be struck in composition software between having too much information and not having enough to complete a given task. Key parameters must be made visible without

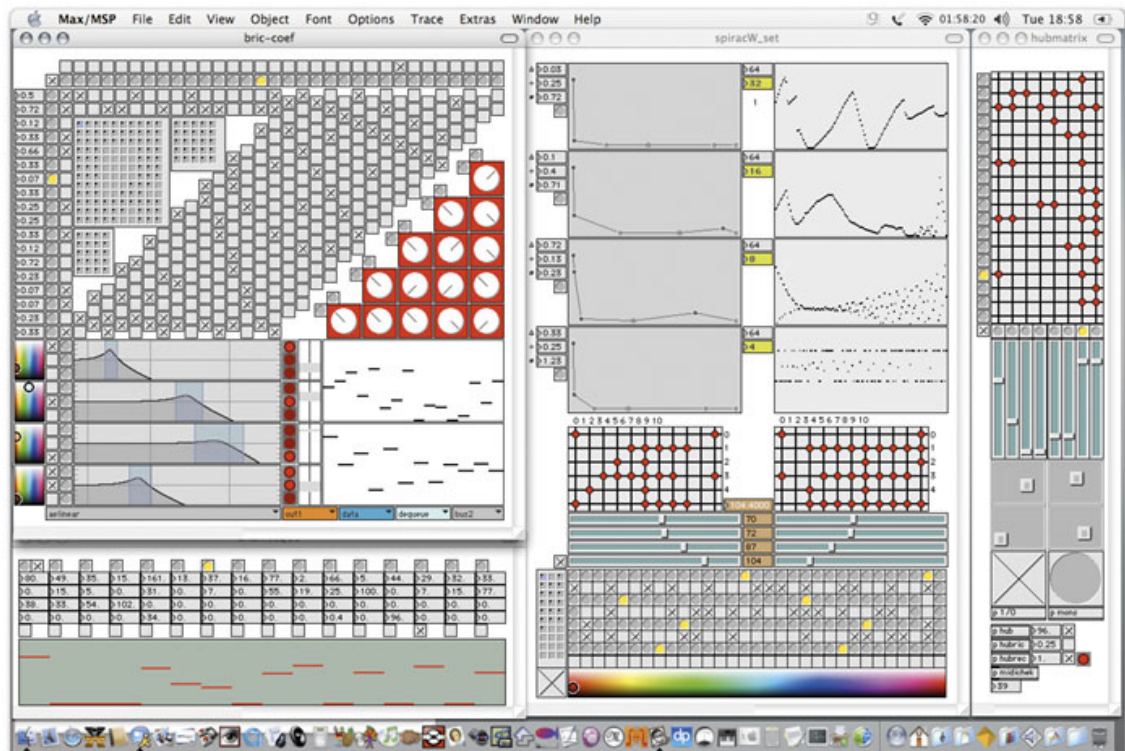


Figure 4.8: An example *Max* patch taken from Autechre's *Confield* (Tingen, 2004).

```
(
  SynthDef('plink',
  {
    arg pitch=200, decay=1, trigger=1, pan=0;

    Out.ar(0, // The bus to be used for the output
    Pan2.ar( // The pan object
    SinOsc.ar( // The sound source
      pitch, // The pitch of the sine wave taken from the argument
      0, // Phase; not used
      EnvGen.kr( // An envelope to control the mul
        Env.perc( // An AR envelope
          0.01, // Attack time
          decay, // Release time, taken from the argument
          0.5), // The mul (i.e. 50% volume)
        Impulse.kr( // The gate for the envelope; makes the envelope open
          trigger)), // The frequency of the impulse, taken from the argument
        pan)) // The pan position, taken from the argument
    ).add
  })

  // Once we have defined the synth we can play it.

  // This will play the synth with the default values.
  n = Synth('plink');

  // This will free the synth (turn it off).
  n.free;
)
```

Figure 4.9: Comments in a *SuperCollider* patch.

introducing clutter to the design. An example of a high level of both visibility and juxtaposability is the layout of the *Cylob Music System* shown in Figure 4.10; several ‘layers’ of the software allow for multiple parameters to be compared and controlled.



Figure 4.10: *Cylob Music System* interface demonstrating both visibility and juxtaposability.

Default settings are not always directly visible, but an interface can surface defaults when they are required. *SuperCollider*, for example, has defaults for every unit generator which are visible within the help files (see Figure 4.11) and shown via tooltips when a unit generator is added (see Figure 4.12). Each unit generator’s arguments can be set either positionally or explicitly via the use of keyword arguments (e.g. `mul: 0.5`); keyword arguments override any positional argument value. While an experienced user may memorise the positional argument order, the information shown in tooltips is likely to be valuable to the novice user.

SinOsc

Interpolating sine wavetable oscillator.

Source: `/Applications/SuperCollider/SuperCollider.app/Contents/Resources/SC ClassLibrary/Common/Audio/Osc.sc`
Inherits from: `PureUGen` : `UGen` : `AbstractFunction` : `Object`

See also: `FSinOsc`, `SinOscFB`

Description

This is the same as `Osc` except that the table is a sine table of 8192 entries.

Class Methods

```
* ar (freq: 440, phase: 0, mul: 1, add: 0)
```

```
* kr (freq: 440, phase: 0, mul: 1, add: 0)
```

Figure 4.11: *SuperCollider* help file for the `SinOsc` sine oscillator.

The user needs to be informed of their current position when navigating the interface. Software such as *Mixtikl* provides this information by having a main view and using windows to access specific elements. Animation and placement denote the layer of the interface that is currently open.

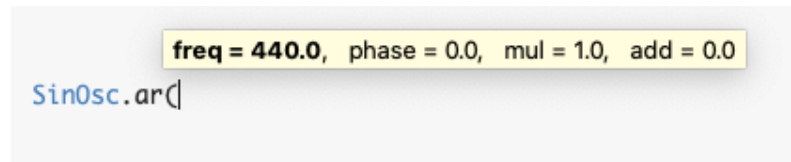


Figure 4.12: Tooltip shown in *SuperCollider*, showing both the method order and default parameter settings.

Both *Max* and *Pure Data* allow users to create subpatches that open in successive layers on the screen; the user can use the ‘Window’ menu to view the open windows and to move to a specific place but the interfaces lacks a clear ‘breadcrumb’-like structure (Adkisson, 2005).

Form-based data entry, as used in software such as *Tune Smithy* (see Figure 4.13) and the *Algorithmic Composition Toolbox*, allows the user to review multiple parameters simultaneously. Such designs do not allow the user to see older entries as they are replaced, which has a negative impact on the juxtaposability of the software. The user is asked to remember the old settings, increasing the work required of the user (Cooper et al., 2014).

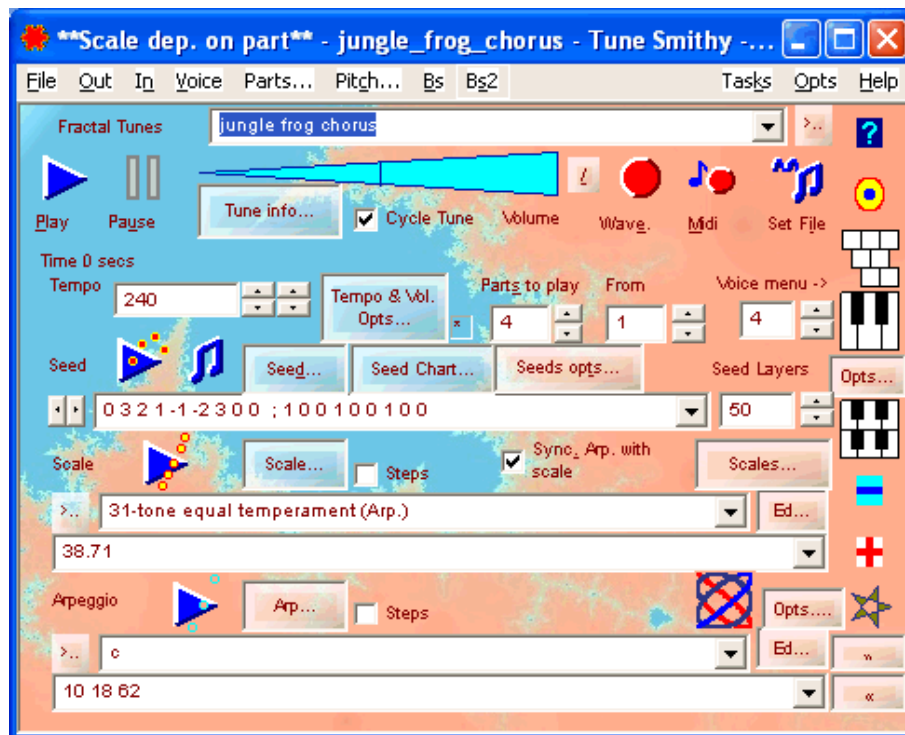


Figure 4.13: The form-based interface of *Fractal Tune Smithy* (Walker, 2011).

Data flow visibility is variable in the software under consideration. Graphical languages such as *Max* and *Pure Data* can exhibit excellent data visibility within single patches, although the use of send and receive objects can impact on this. *Noatikl* and *Mixtikl* show data flow very clearly—Figure 4.14 shows an example of data connection in *Noatikl*. Visibility in text-oriented systems is lower and the user may have to wireframe the patch separately before creating the code.

4.2.7 Closeness of mapping

Mapping refers to the correlation between the interface and the domain. A close mapping (Green and Blackwell, 1998) refers to a situation in which the user can map ‘problem world’ entities directly onto task-specific ‘program world’ entities (Green and Petre, 1996). A close mapping may therefore reduce the number of new concepts the user needs to learn in order to use the interface.

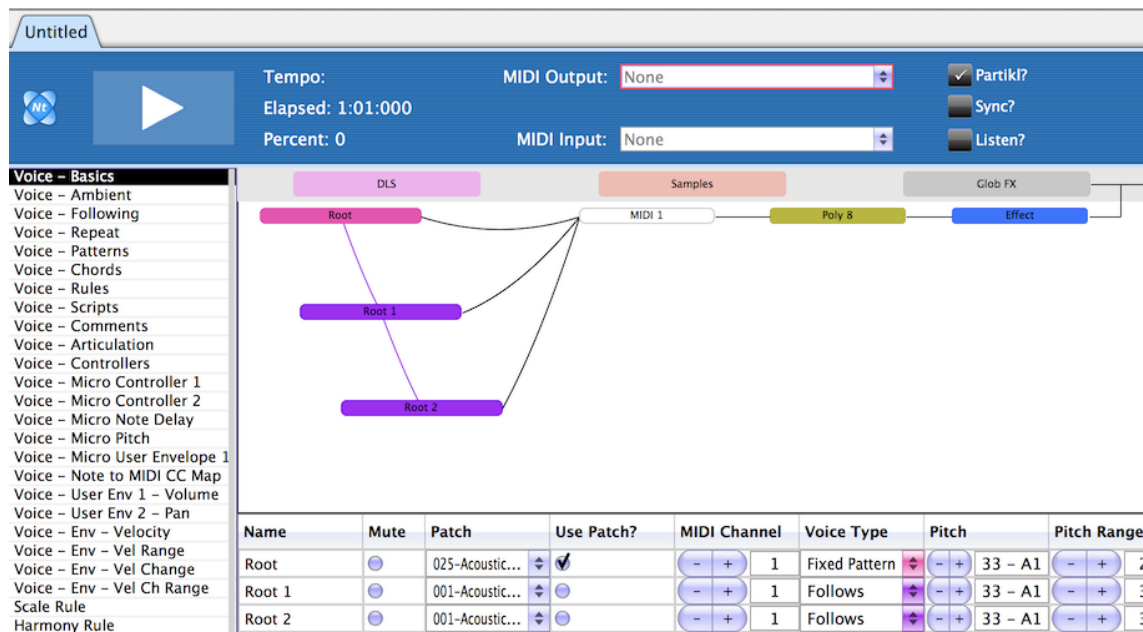


Figure 4.14: Data connection in *Noatikl* (Intermorphic, 2015b).

A distant mapping (Green and Blackwell, 1998) presents a representation model to the user which is significantly different to the domain (Cooper et al., 2014), potentially requiring the user to learn new concepts.

An interesting example, showing how two different domains can match in a useful way, is *Maestro Genesis* (Szerlip and Hoover (2012); see Figure 4.15). *Maestro Genesis* uses the metaphor of animal breeding to allow the user to control the characteristics of 'generations' of music. The software abstracts the generation algorithms behind a 'DNA' button with an icon of a DNA double helix; the user can select from the resulting 'generations', shown in Figure 4.16.

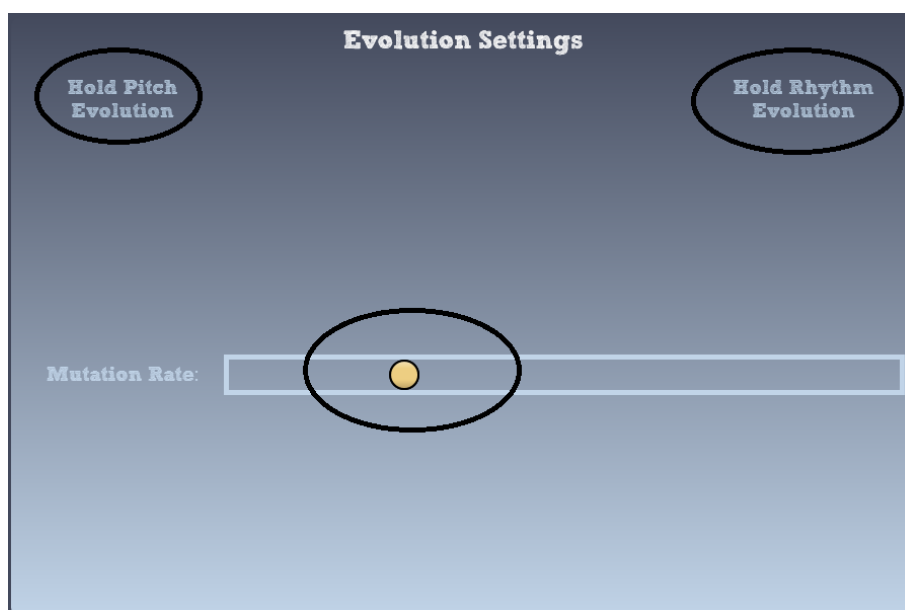


Figure 4.15: Animal breeding metaphor as implemented in *Maestro Genesis* (Szerlip and Hoover, 2012).

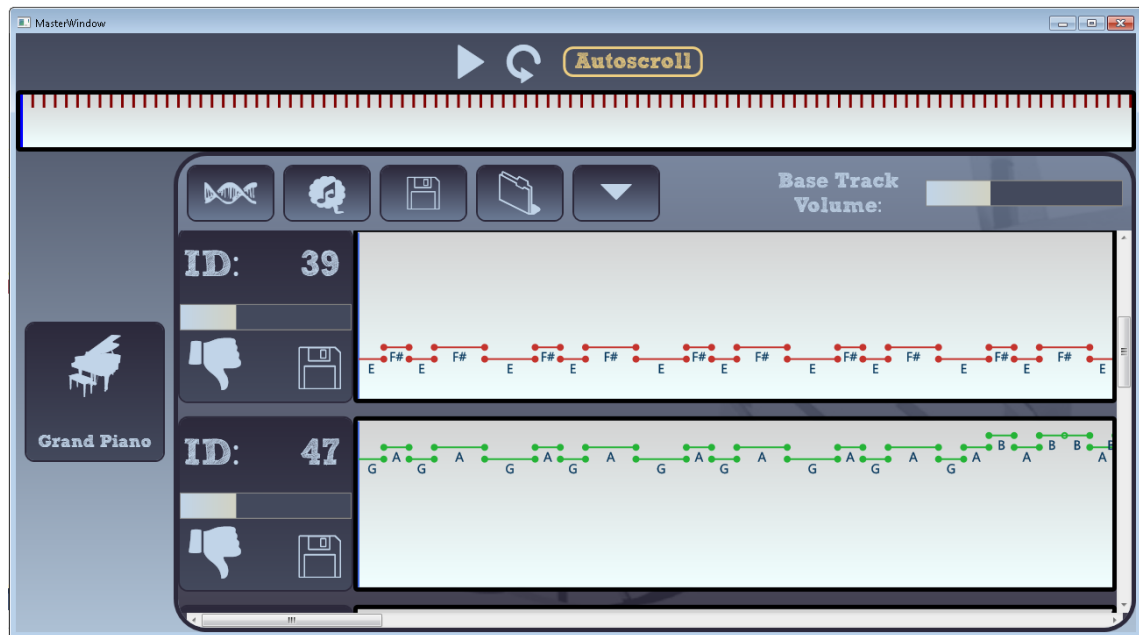


Figure 4.16: Selection of generations in *Maestro Genesis* (Szerlip and Hoover, 2012).

4.2.8 Consistency

Consistency refers to the way in which similar semantics are used in the user interface design (Green and Blackwell, 1998). If an interface is consistent it can positively affect the learnability and usability of the software.

An example of a consistent design is *Mixikl*. The design language leverages both hardware synthesisers (the use of photorealistic rotary potentiometers and faders) and patching (patch cables which ‘droop’ as physical cables do—see Figure 4.17).

Fractal Tune Smithy (see Figure 4.18) makes use of a less consistent design language. The design makes use of notation, piano roll, hardware-style controls, text-based data entry and window and card metaphors. The software is, as a result, highly capable of a wide variety of tasks but potentially at the expense of usability.

There can also be consistency issues when software does not use standard operating system dialogue boxes. An example is *SuperCollider*’s save dialogue, in which the ‘Save’ button is moved from the far right (the OS standard, shown in Figure 4.19) to the far left (Figure 4.20). This is a clear example of poor consistency which could lead to unintended user error (i.e. error-proneness).

4.2.9 Diffuseness

Diffuseness measures the verbosity of language used in the software (Green and Blackwell, 1998, p.39). Shorter names and descriptions can reduce the memory work required of the user (Cooper et al., 2014, p.151). Examples of the use of short names can be seen in graphical programming languages such as *Strasheela* by Torsten Anders (2012)—see Figure 4.21—and Andrew Sorensen’s *Impromptu* (2010)—see Figure 4.22. Inappropriate terseness can conversely lead to user error as there might be too little information for the user. As an example, an *Impromptu* patch can exhibit a high degree of diffuseness and can therefore be difficult to read. Diffuseness can be increased by making both variable names and comments more verbose—an example in *SuperCollider* is shown in Figure 4.23.

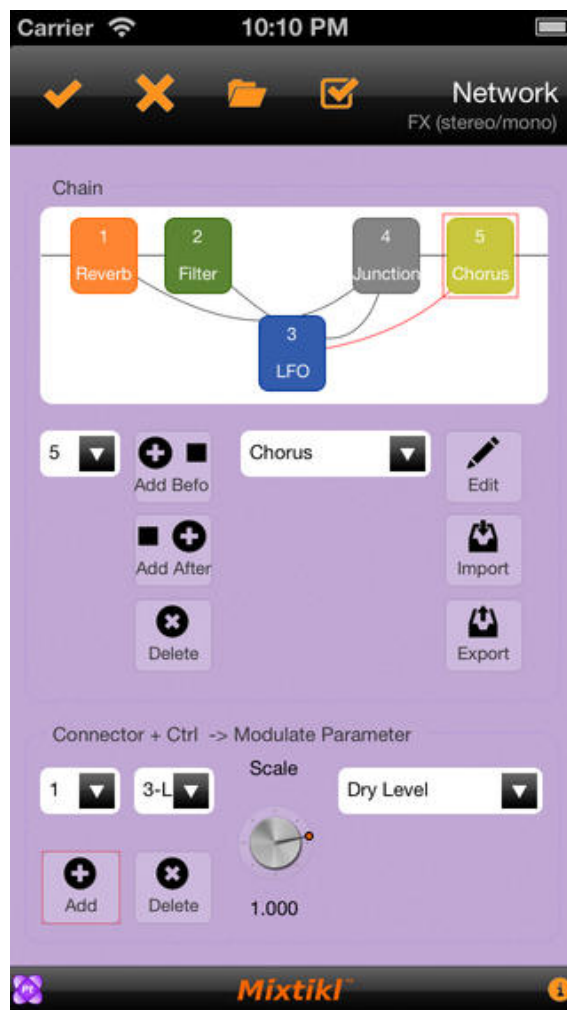


Figure 4.17: Signal flow in *Mixtiki*.

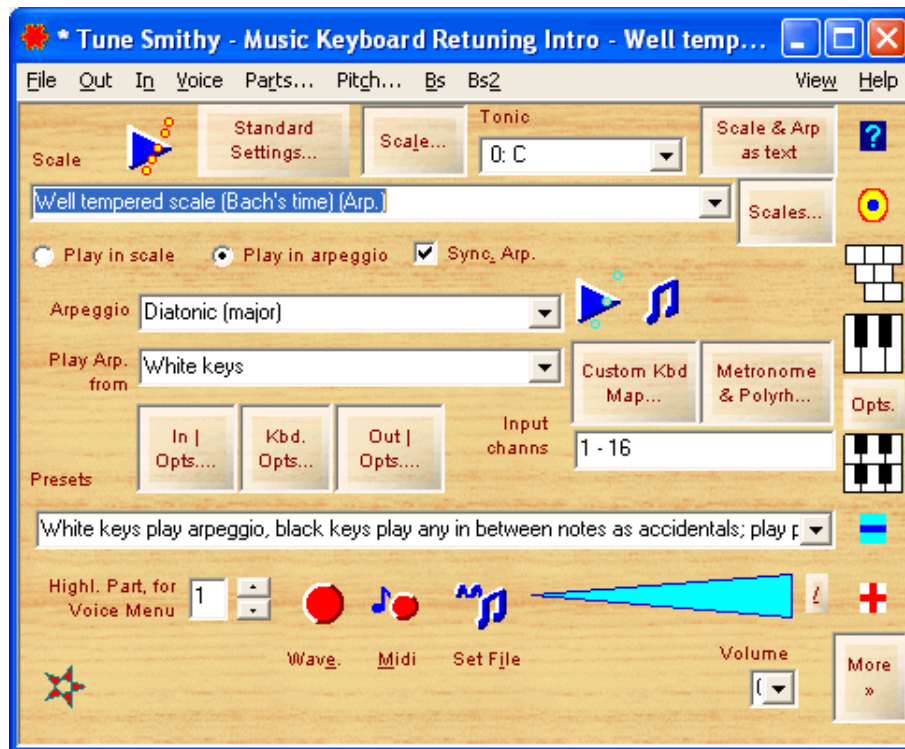


Figure 4.18: One of several window types in *Fractal Tune Smithy* (Walker, 2011).



Figure 4.19: Standard MacOS save dialogue.

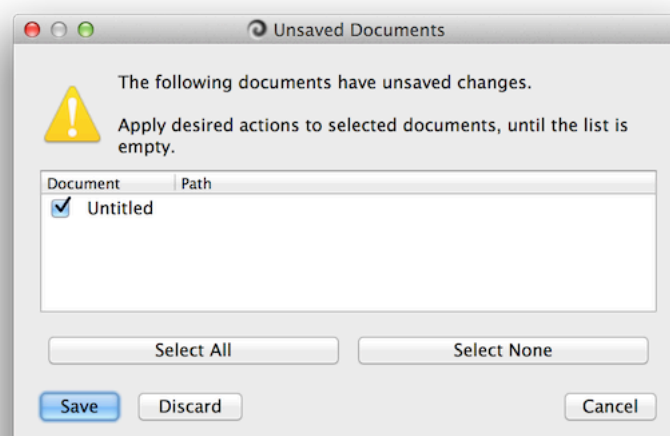


Figure 4.20: *SuperCollider* save dialogue.

```

local
  TextualScore = sim(items:[note(offsetTime:0
                                duration:1000
                                pitch:60
                                amplitude:64)
                           note(offsetTime:500
                                duration:1000
                                pitch:62
                                amplitude:64)
                           note(offsetTime:1500
                                duration:1000
                                pitch:64
                                amplitude:64)])
                                startTime:0 timeUnit:milliseconds)
  MyScore = {Score.makeScore TextualScore unit}
in
  {MyScore wait}
  {Out.outputCsoundScore MyScore
   unit(scoDir:"/tmp/"
        file:"test")}}
  % UNIX: show content of resulting test.sco at terminal (uncomment next line)
  % {Out.exec 'cat' ["/tmp/test.sco"]}
end

```

Figure 4.21: *Strasheela* (Anders, 2012).

```

; make sure that everything is disconnected
(au:clear-graph)

; setup simple au graph
; piano -> output
(define piano (au:make-node "aumu" "dls " "appl"))
(au:connect-node piano 0 *au:output-node* 0)
(au:update-graph)

; play note on piano
(play-note (now) piano (random 60 80) 80 (* 1.0 *second*))

```

Figure 4.22: *Impromptu* (Sorensen, 2010).

```

(
  SynthDef(\fat_PWM,
  {
    arg res = 0.2;

    Out.ar(0,
      Pan2.ar
        (RLPF.ar(
          Pulse.ar
            (MouseX.kr(20, 200, 1),      // pitch on X axis
              SinOsc.kr                  // PWM via SinOsc; 0.2 mul
                (1, 0, 0.2, 0.5),
                0.6                      // Pulse mul
            ),
          MouseY.kr(10000, 200, 1),      // Filter cutoff on Y axis
          res),                          // Resonance
          SinOsc.kr(2, 0, 0.2, 0.5), 0.8) // Panning mod
        )}).send(s)
  )

  x = Synth(\fat_PWM);

```

Figure 4.23: A *SuperCollider* patch made more verbose by commenting individual lines.

4.2.10 Error-proneness

The error-proneness of the system relates to whether the notation used invites mistakes (Green and Blackwell, 1998). The text-oriented systems under review exhibit poor discriminability due to easily confused syntax, which invites error (Blackwell and Green, 2003). For example, *SoundHelix* by Thomas Schürger (2016), shown in Figure 4.24, produces code with a large number of XML tags, potentially reducing human readability and increasing the time taken to write the commands. Such a system increases the possibility of error.

```
<sequenceEngine class="ArpeggioSequenceEngine">
  <normalizeChords>false</normalizeChords>
  <patternEngines>
    <patternEngine class="StringPatternEngine">
      <string>0,0,1,0</string>
    </patternEngine>
    <patternEngine class="StringPatternEngine">
      <string>0,0,1,0,1,1,2,1</string>
    </patternEngine>
    <patternEngine class="StringPatternEngine">
      <string>0,0,1,0,1,1,2,1,2,2,3,2,3,3,4,3</string>
    </patternEngine>
  </patternEngines>
  <patternEngines>
    <patternEngine class="StringPatternEngine">
      <string>0,1,2,1</string>
    </patternEngine>
    <patternEngine class="StringPatternEngine">
      <string>0,1,2,3,4,3,2,1</string>
    </patternEngine>
  </patternEngines>
</sequenceEngine>
```

Figure 4.24: The ‘ArpeggioSequenceEngine’ patch from the *SoundHelix* documentation (Schürger, 2016), exhibiting poor discriminability and high error-proneness.

Such issues can be ameliorated by the syntax checking seen in the Post windows of *SuperCollider* (see Figure 4.25) and *Pure Data*, in which errors are outlined in a limited way. A more thorough error-checking system would be a significant improvement in the software’s usability.

```
-----
ERROR: Command line parse failed
nil
ERROR: syntax error, unexpected CLASSNAME, expecting ')'
in file 'selected text'
line 4 char 9:

      PlayBuf.ar(
      ^^^^^^^
          2, // A stereo audio file, hence 2 channels
-----
ERROR: Command line parse failed
nil
```

Figure 4.25: An example error message in *SuperCollider*.

SuperCollider 3.6 introduced an IDE (Integrated Development Environment) based design, including autocompletion of class and method names—an example is shown in Figure 4.26. Such a system significantly reduces errors introduced by mistyping.

4.2.11 Hard mental operations

Hard mental operations are those that place a high demand on the user’s cognitive resources (Green and Blackwell, 1998); Cooper et al. (2014) also refer to the negative impact of requiring the user to

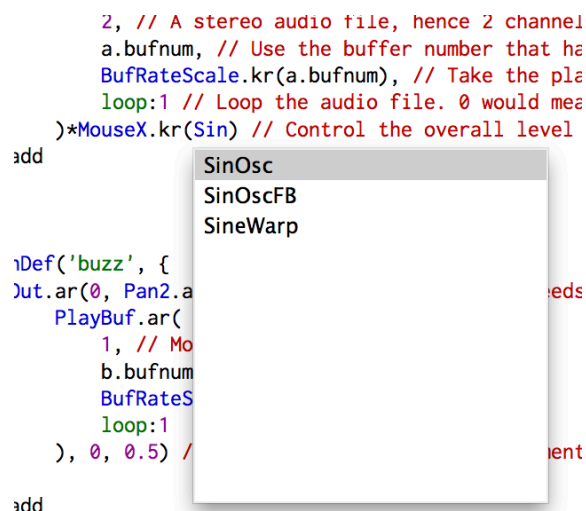


Figure 4.26: Auto-complete suggestions in *SuperCollider 3.6*.

undertake significant cognitive work.

Working in a code-based environment requires the internalisation of signal flow and the logical development of patterns. *ChuckK* (Wang and Cook, 2018) is an interesting hybrid in this respect. Data can be ‘chucked’ from one object to another using the `=>` symbol, the use of which imitates a patch cable; an example can be seen in Figure 4.27. The other text-oriented languages reviewed do not make direct use of graphical or spatial interconnectivity. In this way *ChuckK* makes limited use of Crampton Smith’s second dimension of IxD; visual representation (Moggridge, 2006).

```

// actual FM using sinosc (sync is 0)
// (note: this is not quite the classic "FM synthesis"; also see fm2.ck)

// modulator to carrier
SinOsc m => SinOsc c => dac;

// carrier frequency
220 => c.freq;
// modulator frequency
20 => m.freq;
// index of modulation
200 => m.gain;

// time-loop
while( true ) 1::second => now;

```

Figure 4.27: An example of the `=>` patching syntax in *ChuckK* (Wang and Cook, 2018).

Software using the patching metaphor—graphic systems such as *Noatikl* (Intermorphic, 2015b) or visual programming languages like *Max* (Cycling ’74, 2019)—allow the visualisation of elements such as signal flow and boolean logic; see Figure 4.28 for an example. Externalising the connections between elements reduces the cognitive work required of the user.

4.2.12 Progressive evaluation

Progressive evaluation allows the user to access the current state of their work at any time (Green and Blackwell, 1998). Such evaluation is important in music software as it allows for iterative development, an important compositional technique (Collins, 2005). Progressive evaluation is linked to Tanimoto’s definition of liveness (1990), previously discussed in Chapter 2, in which he considers the types of feedback available to the end users of software. Nash has adapted progressive evaluation for music systems by explicitly linking it with liveness, noting that immediate and rich feedback is

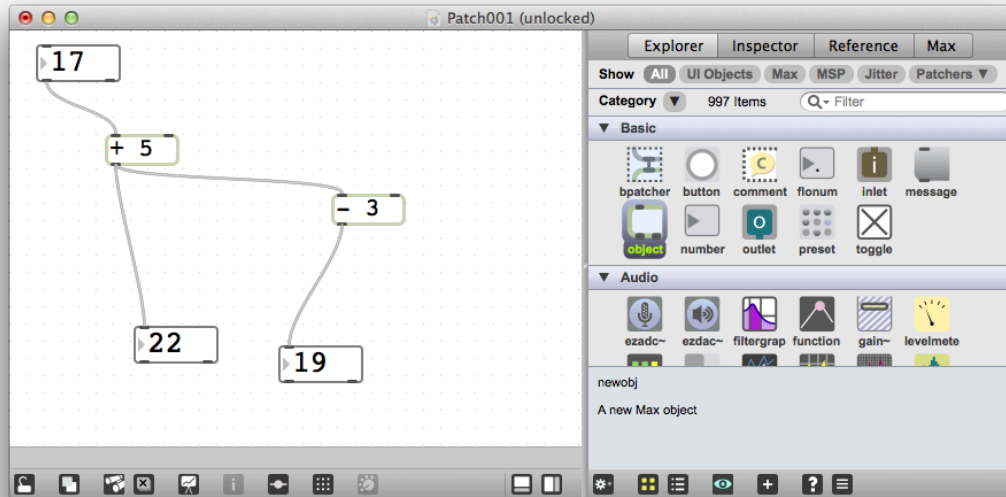


Figure 4.28: A simple patch in *Max* (Cycling '74, 2019), showing the interrelationships between objects denoted by virtual patch cables.

a key factor in facilitating a sense of immersion in the creative act (Nash and Blackwell, 2011; Nash and Blackwell, 2012; Nash and Blackwell, 2014; Nash, 2015).

Two examples of software with minimal progressive evaluation options are both web-based. *Musical Algorithms* (Middleton, 2004) and *Wolfram Tones* (Wolfram Research Labs, 2005), shown in Figures 4.29, 4.30, require the user to upload and download data to a 'black box' with little control of the process. They are effectively offline, non-realtime processes which do not offer feedback before the final product is created.

Maestro Genesis (Szerlip and Hoover, 2012), in using an animal breeding metaphor, offers an interesting framework for interaction and evaluation. The software creates accompaniments for a user-created MIDI file. Several possibilities are presented to the user who selects the most suitable. A 'DNA' command then creates a new generation of accompaniments based on the user's preference. The 'evolution' of rhythm and melody are separated. The design minimises the need for musical understanding as the user is able to make aesthetic choices to influence subsequent generations. The lack of detailed control means that the user needs to carefully curate the source material (the 'parents') used in the generative process.

Code-based systems such as *Csound* require a patch to be completed before the code can be evaluated and run. The use of variables allows for iterative progression without significant refactoring of the code; this leads to an approximation of progressive evaluation (a change—test—evaluate cycle).

4.2.13 Provisionality

Provisionality refers to the degree of commitment the user must make to their actions (Green and Blackwell, 1998). It allows users to make imprecise, indicative selections before making definite choices. Provisionality reduces premature commitment (Section 4.2.4) as it allows a composer to create sketches before allowing for specific details.

Some of the software under review allows the software to make selections within a given range. *SuperCollider* makes use of `.coin` and `.choose` messages for this reason; `.coin`, for example, represents a virtual toss of a coin. Other software, such as *Max*, can make use of pseudo-random numbers in parameters; this allows the composer to issue a command such as 'use a value between x and y'.

1 ALGORITHM

Fibonacci Sequence

This algorithm creates a list of Fibonacci numbers to the n th number or term, where n is selected by you.

[learn more](#)

A. Number of integers in the sequence (3 to 100):

[Get algorithm output](#)

2 PITCH

Next, normalize the algorithm's output by selecting from the options on the right. The values you derive will represent the pitch of each note. Move to Step 3 after making your choices.

[learn more](#)
[keyboard](#)

Scaling:

Use values from to

☒ perform division operation
☐ perform modulo operation

[learn more](#)

[Scale values](#)

Modification:

☒ Convert each to a
☐ Reverse
☐ Invert

[learn more](#)

[Modify Values](#)

ALGORITHM OUTPUT VALUES

DERIVED PITCH VALUES

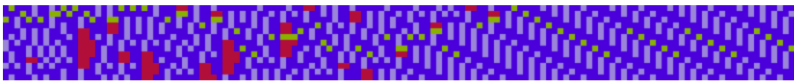
[Start over](#)

Figure 4.29: *Musical Algorithms* (Middleton, 2004).

Generate a composition

[toggle display](#)

ambient | 31.2301430275.9412



[play](#) | [stop](#)
[email](#) | [save](#)

create new composition:
every click gives a unique new composition; go on clicking to find your favorites

classical	dance	hip hop	jazz	world
piano	rock/pop	r&b	blues	experimental
guitar	country	ambient	latin	signalling

[random style](#)
[show variations ▶](#)

composition controls:

GENERATOR	INSTRUMENTATION	PITCH MAPPING	TIME CONTROLS
-----------	-----------------	---------------	---------------

Figure 4.30: *Wolfram Tones* (Wolfram Research Labs, 2005), taken in 2014.

Such selections can increase provisionality in the system, although more complex variations require significant planning which can negate the benefits of being able to create a basic wireframe. DAW software such as *Logic Pro* (Apple Inc., 2021) makes use of audio and MIDI loops to facilitate provisionality in composition and arrangement. Users are able to create sketches using loops, replacing them later in the process. Some of the software under review allows the user to create music following basic harmonic or rhythmic parameters. *Noatikl* has preset objects which can be used to create sequences, and *Impro-Visor*'s preset algorithms allow for quick musical sketches based on a chord progression (Keller, 2019).

Let us consider a composition in which the user wishes to work 'top-down'—that is, to first define a musical structure before then populating each element in that structure. In some genres a repeated section is not an absolute duplicate of a previous section, but is instead a combination of both common and novel elements. For example, each iteration of the musical section may contain a common melodic and harmonic structure, but the second iteration may add a countermelody. One possible design, shown in Figure 4.31, would allow the user to specify the desired structure and then populate the sections. Repeated sections would require multiple 'pools' of information; those attributes common to both, and those specific to each iteration. Note that in Figure 4.31, section A can be populated in three places: the elements common to all iterations, those that are specific to iteration 1, and those specific to iteration 2.

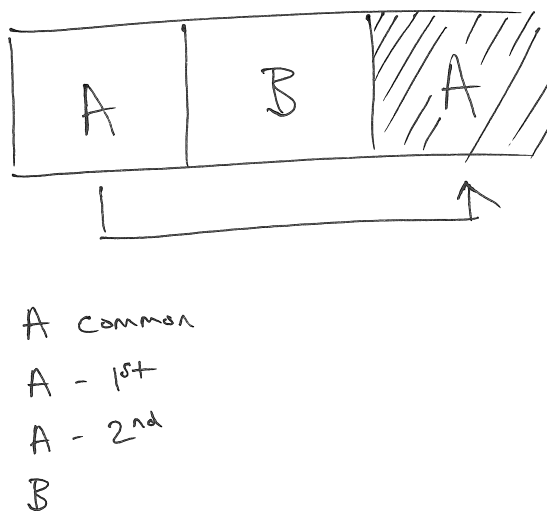


Figure 4.31: Possible layout allowing common material to be shared between sections.

4.2.14 Role-expressiveness

The role-expressiveness of an element relates to how easily the user can infer its purpose (Green and Blackwell, 1998). The use of metaphor (such as mixing desks, synthesisers, piano rolls and staff notation) allows users to quickly understand the potential uses of each editor.

Text-oriented programming languages (such as *Csound*) are not as role-expressive as graphical systems unless the user is experienced in reading and understanding the code. *AthenaCL* by Christopher Ariza (2011) is a composition framework written in Python. As shown in Figure 4.32, the role-expressiveness of the code is low; the system allows for graphical output in the form of piano-roll-style mapping, but there is minimal use of interface metaphor.

Graphical languages (such as *Pure Data*) can be role-expressive as the connections between elements are clear. It is possible for a user to see an image of a *Pure Data* patch and understand the interrelationships of the objects and the purpose of the patch.

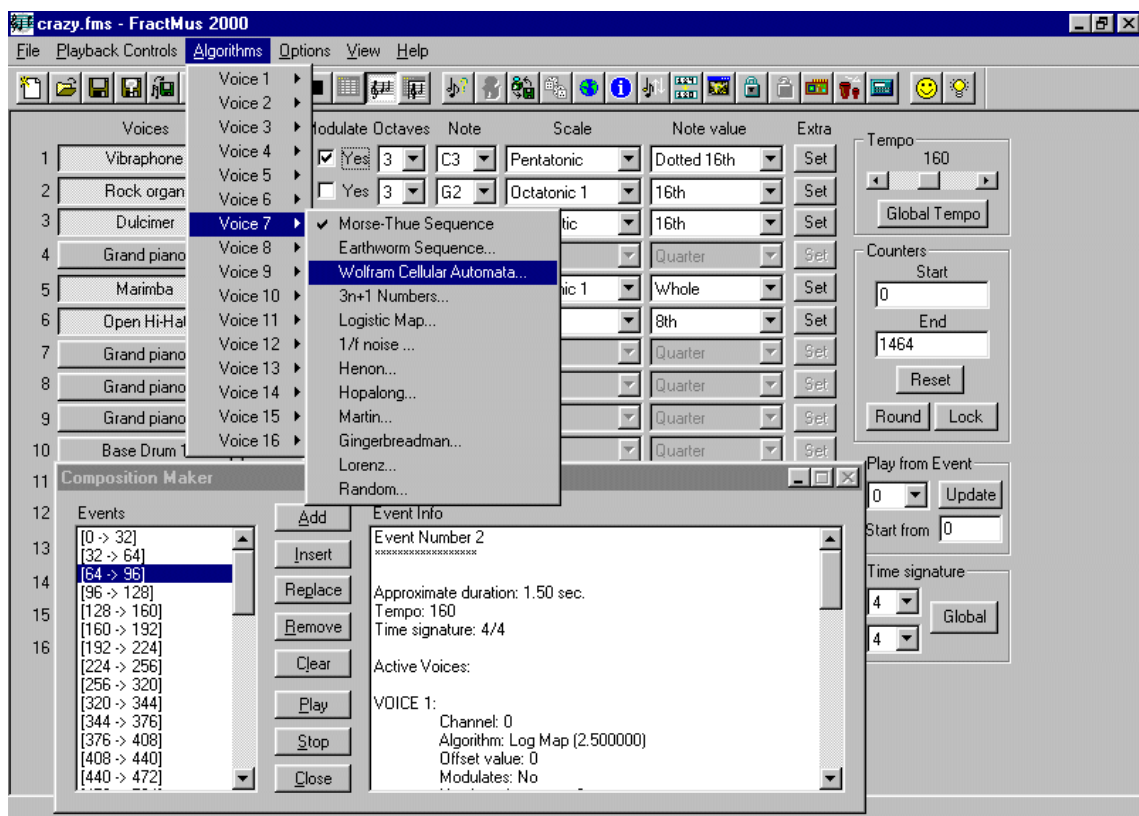
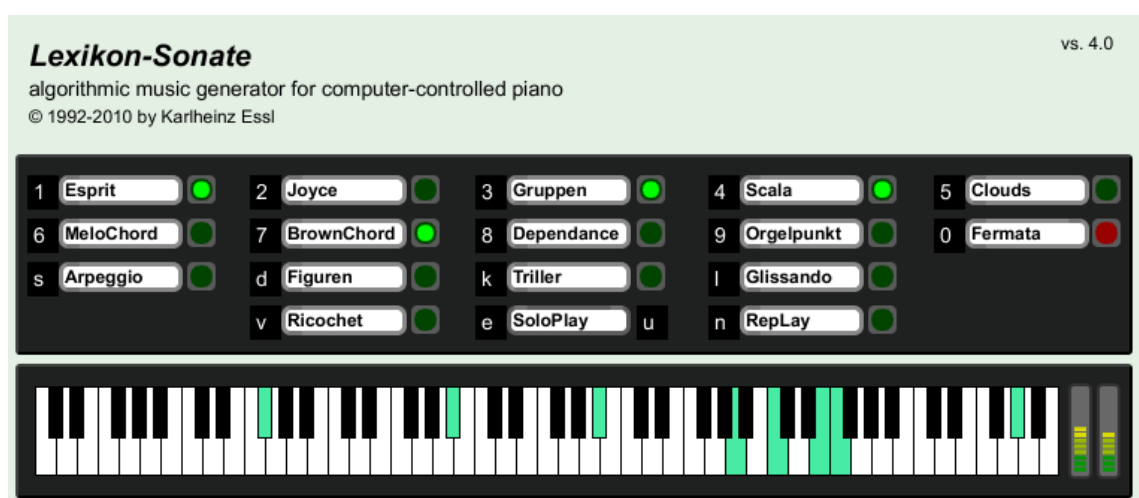
Figure 4.33 shows the interface for *FractMusic* by Gustavo Diaz-Jerez (2012), an algorithmic composition tool based on fractals. The interface is not based on that of an instrument (such as

piano roll) or staff notation, but instead allows the user to select the musical parameters to be used (such as instrumentation, scales and modes, rhythm, octave etc.). As such the interface relies on the user's familiarity with musical terminology. Some icons are ambiguous (a smiley face or a lightbulb, for example).

Impro-Visor (Keller, 2019) uses a lead sheet metaphor. The software is targeted at jazz musicians with an assumption that users will be familiar with the use of staff notation and chord progressions. The interface, shown in Figure 4.35, includes a transport control metaphor (play, pause, stop, record) and presents algorithmic choices as musician’s names to denote the intended style. The role-expressiveness will therefore be high among the target users.

Some important issues in interaction design for algorithmic composition do not appear to be fully addressed by the CDN, or by other design discussion tools noted in this chapter. One such issue is the variety of demands made on compositional representations of time. Of course, notions of time are considered in design discussion tools. For example, Crampton Smith identified time as one of the four dimensions of interaction design (Moggridge, 2006). Time is not viewed as a separate entity in the CDN, although it is implicit in some dimensions. Kutar et al. (2000) reviewed the representation of various time granularities in TRIO, a real-time logic language (Morzenti et al., 1989), with reference to the CDN. There are various representations of time in software generally; research suggests that there is no one preferred type (Kessell and Tversky, 2008).

Payne (1993) reviewed the representations of time in calendars, which primarily focussed on

Figure 4.33: *FractMusic* (Diaz-Jerez, 2012).Figure 4.34: *Lexikon-Sonate* (Essl, 2010).

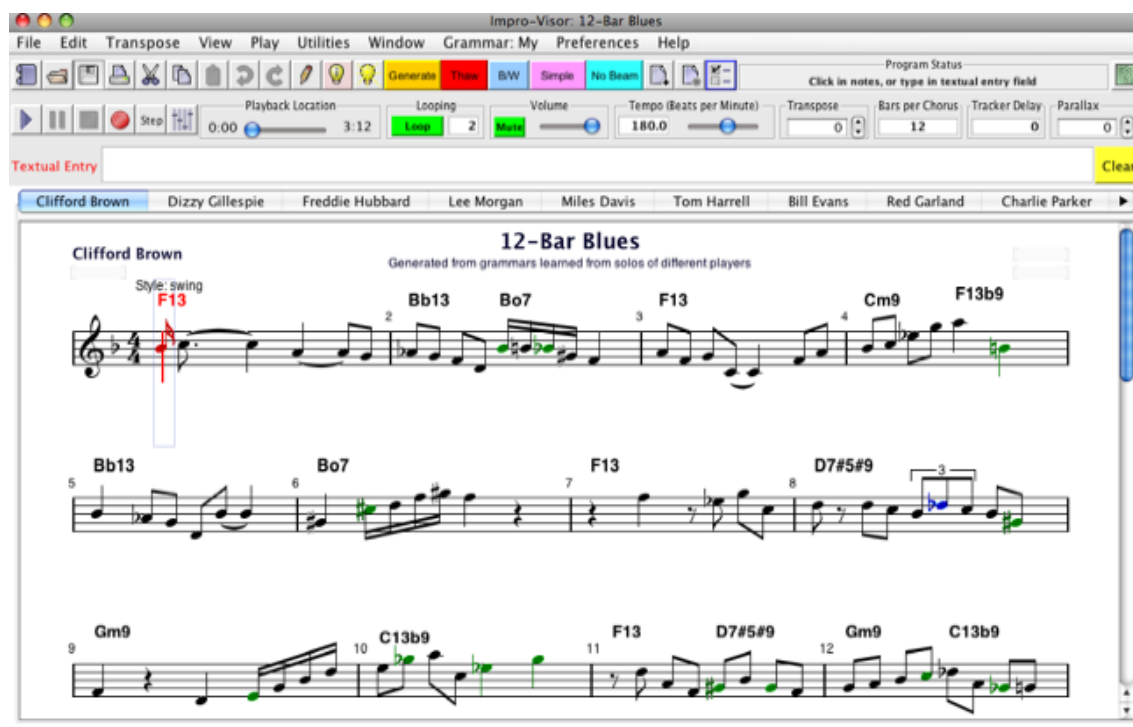


Figure 4.35: The *Impro-Visor* interface (Keller, 2019) utilising a lead sheet metaphor.

the use of horizontal and vertical spatial information to imply the passage of time: in many cases a similar approach can be taken by music software. Sequencers, such as *Cubase* (Steinberg Media Technologies GmbH, 2021), frequently use horizontal motion from left to right to denote the passage of time. Trackers frequently show the passage of time as a vertical scroll from top to bottom, although patterns can be played out-of-order to allow for semi-linear playback (Nash and Blackwell, 2011); see the previous overview of tracker design in Section 2.4.1.

The use of a static horizontal plane to denote time is common (sometimes with a moving pointer or index); *Tune Smithy*, *Maestro Genesis* and *Noatikl* are examples of this. Some of the reviewed software does not directly show the passage of time; offline, non-realtime software such as *Musical Algorithms* output files for use elsewhere. The text-oriented systems reviewed here are generally capable of generating graphical elements but this is not vital to their operation; the software can operate without any visual feedback for the user.

Much musical software makes use of cyclic time (loops), as well as linear time. Both of these kinds of time can be sequenced, or mixed, or arranged hierarchically at different scales, or arranged in parallel streams, or all of these at once. This can also be true of general programming languages, but is often a detailed focus of algorithmic composition software. Software written to perform loop-based music frequently uses a different interface to denote the passage of time. *Live* (Ableton, 2021) makes use of horizontal time in some interface components; other interface elements allow the user to switch between sample and synthetic content in real-time with no time representation. *Mixtikl* is a loop-based system and, in several edit screens, does not show the passage of time at all as the user interacts with the interface.

Manhattan (Nash, 2014), previously introduced in Section 2.8.6, embodies an interesting approach to allow programmatic control over musical form. *Manhattan* adopts an enhanced tracker design, which builds on the standard tracker representation of music as fixed grids (patterns) with a vertical linear time axis. A key *Manhattan* innovation is to integrate spreadsheet-style formulas into this design, which allows for control over musical form via control flow structures such as loops, and both conditional (if-then) and unconditional (goto) branching.

Audiocubes (Percussa, 2013), used with *Improvisor* (Percussa, 2012), use a static view of a per-

petually looping step sequencer and so do not need to show time elapsing. *Audiocubes* are wireless hardware devices that use their orientation with relation to other cubes (via 4 IR ports) to trigger rhythmic and melodic patterns. Patterns are created in the *Improvisor* software (see Figure 4.36) according to the different orientations of each combination of cube surfaces. Performance is therefore achieved by the spatial placement of the cubes.

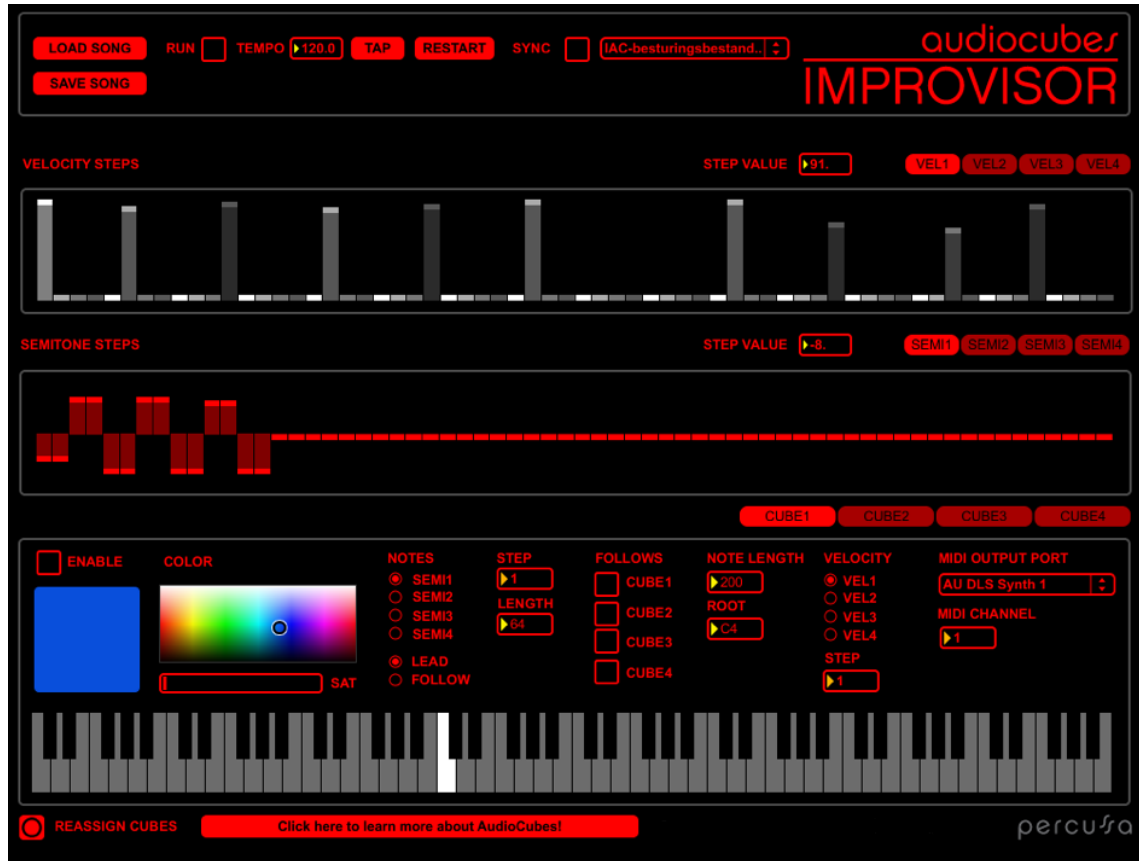


Figure 4.36: The *Improvisor* software (Percussa, 2012).

In a classic paper, Desain and Honing (1993) discuss different implicit time structures in tonal music. They point out that, in order to competently speed up piano performances in certain genres, it is no good simply to increase the tempo. While this may be appropriate for structural notes, decorations such as trills tend to need other manipulations such as truncations without speed-ups or substitutions to work effectively at different tempi. Similarly, elements of rhythm at different levels of periodicity, for example periodicities below 200 milliseconds vs. above 2 seconds, may require very different kinds of compositional manipulation since the human rhythm perception (and composers and performers) deal differently with periodicities in these different time domains (Angelis et al., 2013; London, 2012). In a related sense, Lerdahl and Jackendoff (1983) uncover four very different sets of time relationships in harmonic structures in tonal music.

Honing (1993) differentiates between tacit (i.e. focussed on ‘now’), implicit (a list of notes in order) and explicit time structures. Some of the software under review can be considered in this way; for example, some modes of operation in *Mixtikl* and *Live* utilise tacit time structures, the note lists in *Maestro Genesis* and *MusiNum* are implicit time structures, and software such as *Max* or *Csound* can generate material which uses explicit time structures. The flexibility of many of the programming environments under consideration means that the user can determine the timing structures to be used. Honing (1993) also applies the same process to structural relations (see *Repetition Viscosity* above): he suggests that there are tacit, implicit and explicit structural relations. A system which uses explicit structural relations would allow the musical structure to be both declarative

and explicitly represented.

4.4 Conclusions

The Cognitive Dimensions of Notations have proven to be a useful, if broad-brush, tool in the analysis of algorithmic music software and in the articulation of issues affecting how usable these tools are and how well they work. Some of the reviewed software exhibits high viscosity and requires significant user knowledge. The use of metaphor (staff notation, music production hardware) introduces multiple levels of abstraction which the user has to understand in order to use effectively: some instances of close mapping reduce abstraction but require the user to do more work. Significant premature commitment is not conducive to music composition, and there are clear opportunities for the greater provisionality that a piece of structurally-aware music software could provide. Visibility and juxtaposability are frequently compromised by complex design. Patching software reduces the hard mental operations required of the user by making the signal flow clear, although graphical complexity can have a negative impact on role-expressiveness. Complexity leads to error-proneness in several instances, although there are some tools (such as error-checking and auto-completion) which seek to ameliorate the main problems.

The issue of time raises particular questions and problems. Algorithmic composition tools use varied interaction designs, and may heterogeneously mix diverse elements from different musical, algorithmic, and interaction approaches. Time can be represented in a number of ways, each emphasising different temporal issues (i.e. tacit, implicit, or explicit). To some degree, these issues parallel similar issues in general programming, for example concerning sequence, looping, hierarchy and parallel streams. However, growing knowledge about how people perceive and process different kinds of musical structure at different time scales suggests that the design of algorithmic composition tools may pose a range of interesting new design issues.

The findings from this chapter will be used to inform the selection of candidate notations outlined in Chapter 5.

Chapter 5

Design considerations for an algorithmic composition language

This chapter reviews the principles identified in Chapter 4 and uses these to propose heuristics which inform the selection of two candidate frameworks—Direct Combination and Petri Nets—to use in a series of design exercises. The findings from these exercises will be used to inform the development of a new abstraction in the next chapter.

Recall that this thesis is focussed on finding ways to create nondeterministic composition software which is both usable and musically expressive for non-programmers. In order to better understand the area, Chapter 4 presented a review, using the Cognitive Dimensions of Notations, of a range of software for algorithmic composition.

The chosen Research through Design methodology (see Section 3.2 of Chapter 3, *Methodology*) works on the assumption that possible solutions to the stated problem will emerge gradually through the iterative testing and development of candidate approaches, each leading to its own issues, questions, and solutions. With this in mind, this chapter briefly summarises the findings from the CDN review before showing how they suggest Direct Combination and Petri Nets as candidates for a series of design exercises (Gaver, 2012). This chapter presents annotations of the DC and PN design exercises before summarising findings.

5.1 CDN review outcomes

As detailed in Chapter 4, a Cognitive Dimensions of Notations (CDN) analysis of a broadly representative sample of systems was carried out in order to identify some of the design issues and trade-offs in the different approaches to the interaction design of existing algorithmic music composition systems. For the reader's convenience, the conclusions from the analysis are collected below; note that each identified set of characteristics is associated with one or more systems reviewed.

Requires significant user knowledge Much of the reviewed software requires the user to be competent in the design and implementation of algorithms within either text-oriented or graphical programming environments. Systems not requiring this typically have limited expressivity. Some of the software assumes musical knowledge, and some requires the user to understand both areas.

Use of metaphor frequently requires prior knowledge in another domain Metaphor is often used. For example, several graphical programming languages make use of a patch-cable metaphor, while other systems reference mixing desks, common music notation, or sequencer-like piano rolls. Metaphor can help users to infer purpose and to leverage existing

knowledge but it can also present a barrier to those with limited experience in the referenced fields.

Imposes working practices not conducive to some compositional processes Some reviewed systems require users to commit to some representation of pieces they are creating which is subsequently difficult to change as the piece is developed.

A lack of structural awareness In some systems, the user is not easily able to define, and subsequently change, the music at one or more important structural levels to be used in the composition. This tends not to be a problem with programming languages, but significant programming knowledge is required.

Complex visual design leading to low visibility and juxtaposability The reviewed software exhibits a wide range of graphical design. Graphical programming languages, e.g. using a patching metaphor, allow complex operations but visibility is significantly reduced once the patch's size increases. There are also issues with surfacing pertinent information and allowing the user to easily compare elements.

5.1.1 Heuristics from the CDN analysis

Having used the Cognitive Dimensions of Notations to analyse diverse existing software for algorithmic music composition, and given the specific aim of finding an approach which is both usable and musically expressive for non-programmers, a set of design heuristics to inform the initial design process will now be proposed. Note that these heuristics are intended to be used for the first design only; in keeping with Research through Design, each iteration may produce a new set of principles to be explored in subsequent designs. These initial heuristics are based on intuitive judgement, but are grounded on the twin pillars of analysis and the stated design goal.

Minimise the learning and prior knowledge required of the user. The target user, as outlined in Chapter 1, has little or no programming experience, and few formal musical skills. For this class of user, it is desirable to avoid a traditional text-oriented programming environment. The motivation here is to abstract away much of the low-level syntax using graphical constraints; for example, tasking the user with filling in a template rather than requiring that they learn and use a specific syntax. Of course, the use of a graphical language does not, in itself, make the software more usable or make the underlying structure easier to understand (see Petre, 1995; also Moher et al., 1993) but it does reduce the syntactic knowledge required of users, which is particularly important given the focus on novice programmers. The use of templates to reduce the need for the learner to learn syntax and to prevent combinatorial errors is in common with the design goals of Scratch (Resnick et al., 2009), previously discussed in Section 2.8.3. This heuristic, pertaining to minimising the learning and prior knowledge required, mirrors Nielson's 'recognition rather than recall' heuristic (1995) which suggests that users should be able to retrieve important information from the visual interface. This is partially challenged by Nash and Blackwell (2014) in their first heuristic ('support learning, memorization, and prediction, or "recall rather than recognition"') who argue that, while such designs can be useful for novice users, expert users benefit from interfaces which reduce mediating visual metaphors in favour of leveraging the user's memory, physical interaction, and personal experience.

Be cautious regarding domain-specific metaphors. Many pieces of music software make use of metaphor in order to communicate function or meaning, which is a sensible design decision when the user is already familiar with the source of the metaphor. For example, recording engineers in the 1990s were familiar with mixing consoles, and so *Pro Tools* presented them with a familiar mixing console-like interface to reduce the learning required for effective use (see Section 2.4.3). However, most contemporary novice musicians and music producers gain all their experience from computer interfaces rather than hardware. Thus, rather absurdly, novice users learn to mix via a metaphor grounded on hardware they have never used. I

argue that this can increase the burden on the user, and that a design which avoids domain-specific metaphors (such as patch cables, mixing consoles, or piano keyboards) can reduce the learning burden for the user, and may remove a perception barrier ('this isn't for me because I am not a pianist'). Resisting the search for metaphors facilitates exploration of a wider design space to find better approaches. This heuristic mirrors Nash and Blackwell's third heuristic, 'minimize musical (domain) abstractions and metaphors' (2014).

Allow for multiple compositional approaches; quick changes; auditioning The Cognitive Dimensions of Notations analysis highlighted how different interfaces used in algorithmic composition-capable software have different sets of trade-offs. This was also seen, from a different perspective, in the discussion on common music notation and DAW design in Sections 2.2, 2.4.3, 2.5. Nevertheless, there are situations where interface design has the potential to give strong advantages for particular aspects or processes involved in composition (Fernández and Vico, 2013). For example, it is particularly valuable to be able to switch easily between a top-down and bottom-up approach to composition (Sections 4.2.1, 4.2.4). On a related note, and as argued by Nash and Blackwell (2014) in their second heuristic ('support rapid feedback cycles and responsiveness'), musicians often use trial-and-error when composing, as discussed in Section 2.6. A goal of the system being designed, therefore, should be to allow the user to make quick, impactful, and non-destructive changes and to audition the results rapidly in order to make a final aesthetic judgment.

Enhance visibility and juxtaposability via reduced visual complexity As outlined in Section 2.7, I am interested in the tradeoff between the expressivity and usability of nondeterministic music composition software. The CDN review highlighted the benefits of a graphical metaphors such as patching, but there are problems with visibility and juxtaposability when patches become overly complex. For this reason the design should seek ways in which the interface can be simplified so that the most relevant information can be surfaced for quick reading and editing. Note that a reduction in data visibility risks increasing hidden dependencies, so there may be a trade-off between the two.

5.2 Candidate frameworks

Having used the Cognitive Dimensions analysis in Chapter 4, and the design goals as characterised in Chapter 1, to distil the design heuristics outlined above in this chapter, the design heuristics were subsequently applied for the first time to pick out two representative approaches selected from the literature review (Chapter 2) that seemed to best fit the newly formulated heuristics. The aim was to identify two complementary approaches that could be used as the basis for design exercises in keeping with the design goal and, at the same time, further refine the design heuristics. The two approaches selected from the literature review were Direct Combination (DC) and Petri Nets (PN), introduced in Sections 2.9, 2.10.3. Both DC and PNs were used in a sequence of design exercises, annotated below.

5.2.1 Design exercise using Direct Combination

Direct Combination (abbreviated to DC) was identified as a suitable candidate design principle to explore the heuristics identified in the CDN software review. DC allows users to select any number of interaction objects and uses this selection to constrain the search space of possible commands. In this way, users can indicate which objects they wish to use without having to learn or remember the operations that the system allows. The selection of objects narrows options to those that are contextually relevant.

Music is multidimensional, and expressive music software needs to offer independent control of contextually relevant parameters (such as pitch, rhythm, timbre etc.). DC offers an immediacy in use as every element can work with other elements incrementally and reversibly in a variety of ways.

This kind of interaction, enacted either with intent or serendipitously, is an attractive property for any system which is aimed at giving expressivity and usability to novice users. The use of multiple objects in constraining options may be useful when presenting a complex process to novice users. DC also simplifies menu-driven interfaces which allow control of all parameters; the selection of none, one, or more objects can constrain options to those available and contextually relevant. The user does not always have to select the materials and then make cognitive and physical efforts to seek out the relevant operations. Instead, choices of targeted operations can be summoned simply by selection by the user of one or more materials of interest. Consequently the number of user interface steps are potentially reduced. However, at the same time, the expressivity is potentially very high because this scheme can index operations to a high degree. Of course, a good usability/expressivity balance is not guaranteed.

Sketches and findings from an informal design exercise are shown in Section D.1.1 of Appendix D. While the DC exercise proved useful, issues resulting from the management of a large library of assets and the proliferation of musical functions proved intractable.

5.2.2 Design exercise using Petri Nets

Recall that the findings from the CDN review in Chapter 4 led to the decision to explore Direct Combination and Petri Nets in a series of design exercises, both to seek candidate designs for a nondeterministic composition system for non-programmers and to further refine the design heuristics for designing such systems. The next section presents a design exercise with Petri Nets.

The exercise will explore the following:

- What kinds of processes Petri Nets are suited to representing;
- How Petri Nets can be used or adapted to represent fundamental musical processes;
- Some simple musical fragments expressed in Petri Nets.

By considering emerging findings, tentative design heuristics and design decisions may be informally identified from the above.

5.2.2.1 Petri Nets

As outlined in Section 2.10.3, Petri Nets have been used for the formal representation and structural description of music (see Haus and Sametti, 1991). Whereas Petri Nets can, in principle, represent concurrent processes and synchronisation, in practice these features do not have the temporal precision required for musical purposes. For such reasons, Baratè et al. (2005) proposed Music Petri Nets, an extension of standard Petri Nets, to allow for additional control over nondeterminism via conflict; temporisation to allow for concurrent processes and synchronisation; and probabilistic weight to exert control over nondeterminism.

Within a Petri Net, nodes are either *places* (shown as circles) or *transitions* (shown as rectangles). Arcs (shown as arrows) connect nodes of different types—places can only connect to transitions, and vice versa. A transition is *enabled* when the incoming places present a greater or equal number of tokens to the weights of the incoming arcs. Once a transition *fires*, a number of tokens equal to the weights of the incoming arcs are subtracted from the incoming places. A number of tokens equal to the weights of the corresponding outgoing arc are added to the outgoing place. In Music Petri Nets, music objects can be added to places, and music operators can be added to transitions. A music object can be anything which is musically meaningful, and can be simple or complex, abstract or defined. A music operator can transform music objects via a transformational algorithm (e.g. transposition or time stretching).

Baratè (2009) presents an example, *Peaches en Regalia* by Frank Zappa (1969), shown in Figure 5.1. It can be seen that the initial melody, section A, is loaded into the Music Petri Net via the ‘Load A’ place and is immediately played without transposition; there are no tokens in either the

‘Transpose +’ or ‘Transpose -’ places. The addition of tokens to both of these places would result in a nondeterministic choice between the two. Section A is played once and then repeated for a further three times, controlled by the number of tokens in the ‘Repeat A’ place. Once there are no more tokens in ‘Repeat A’ then a token is passed to the ‘Load B’ place before section B is played twice; this time the music operators control a warping of time so, if tokens were to be added to the ‘Time +’ and/or ‘Time -’ places the section could play at half or double its normal length. After the repeat of section B a token is passed to the ‘Play C’ place, which plays once without any transformation, before a token is passed to the ‘Repeat All’ place. In this example Baratè (2009) shows user-controllable places as white; for example, adding tokens to the ‘Start’ place will control the number of repeats, and both transposition and section repeats can be controlled by adding or removing tokens.

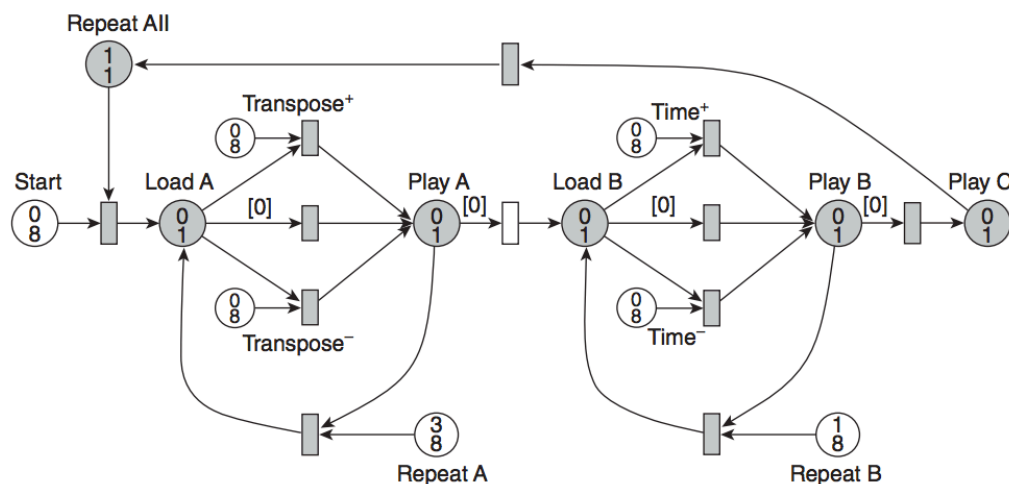


Figure 5.1: *Peaches en Regalia* by Frank Zappa represented as a Music Petri Net (Baratè, 2009).

5.2.2.2 Musical examples using Petri Nets

The initial Petri Net tests were carried out not using Music Petri Nets, but using *PIPE* (Tattersall and Knottenbelt, 2014), an editor for building and evaluating Petri Nets. The tool was useful in evaluating musical possibilities in standard (i.e. not Music) Petri Nets. *PIPE* was written to evaluate Petri Nets without the music-specific temporisation or synchronisation capabilities introduced by Baratè et al. (2005). I discussed extending its capabilities with two of the developers, including testing the JavaOSC library (Ramakrishnan, 2002) to connect *PIPE* and *SuperCollider* via Open Sound Control (Wright, 2005), but the timing requirements of Music Petri Nets would require significant changes to the *PIPE* code. It was, however, a useful tool in understanding the musical moves made possible by Petri Nets. An example of this work is shown in Figure 5.2, in which the Baratè (2009) Zappa Petri Net shown in Figure 5.1 was recreated in *PIPE*.

Alongside the *PIPE* tests, a Music Petri Net system was developed in *Pure Data* (Puckette, 1997). While functional, the system quickly became overly complex due to the graphical patching language issues outlined in Section 4; issues with viscosity, hidden dependencies, premature commitment, visibility, error-proneness, and progressive evaluation. A simple Petri Net example in *Pure Data* is shown in Figure 5.3, showing the complex connections required to add and remove tokens from places when a transition fires; such visual complexity is the result of the limitations of one-way patch cable-style links (see Section 2.8.8).

The next round of design exercises were undertaken using *SuperCollider* (McCartney, 2002) extended using the *Thr44* quark¹, a different and incomplete version of Petri Nets by Miguel Cardoso (2014). Cardoso’s Petri Net implementation does not use Music Petri Nets; instead, it attaches

¹Quarks are packages of *SuperCollider* code containing classes, extension methods, documentation, and server plugins.

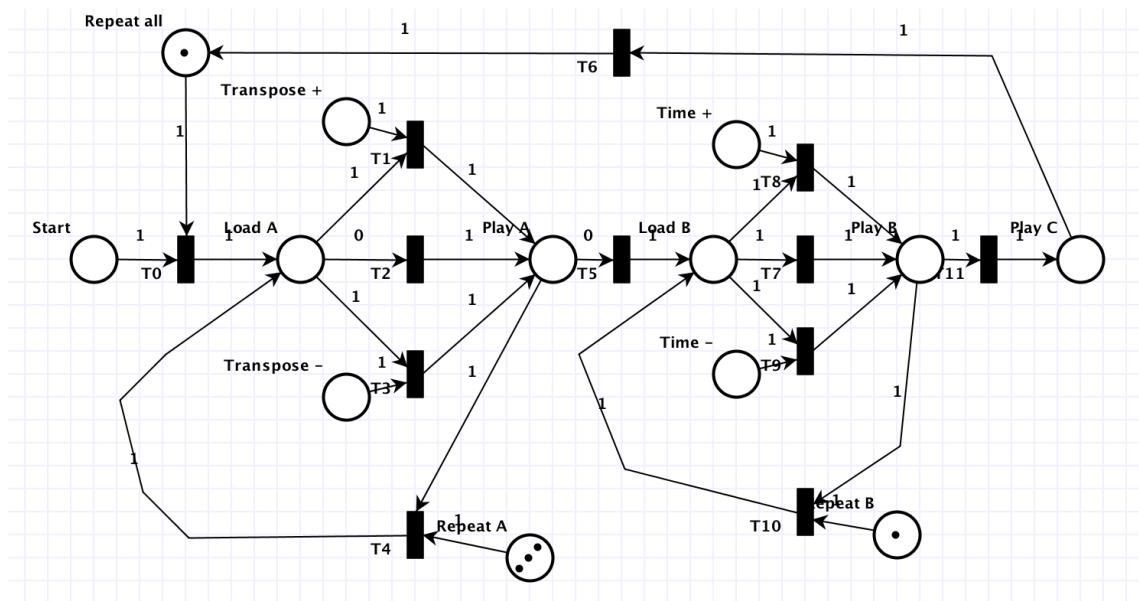


Figure 5.2: An original implementation, in *PIPE*, of the Music Petri Net shown in Figure 5.1.

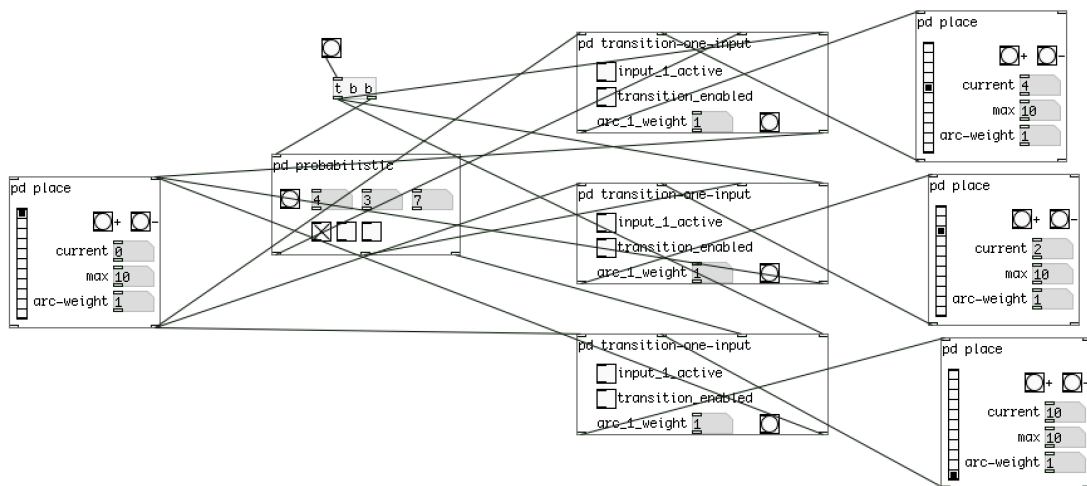


Figure 5.3: A simple Petri Net in an early *Pure Data* (Puckette, 1997) PN system.

music objects to transitions, rather than places. As a result, the Baratè et al. (2005) and Cardoso (2014) versions of Petri Nets for music are not at all compatible. Some key Petri Net functionality is not fully implemented in the *Thr44* library—importantly, playback is deterministic regardless of arc weight. However, a series of tests were run to explore this specific implementation of Petri Nets. These tests made use of simple audio loops which had not been specially created; they were taken from audio stems for the track *Discipline* by Nine Inch Nails (Reznor, 2011). Musical models were created based on the Petri Net primitives reviewed in Section 2.10.3 in Chapter 2; these may be seen as representing the concepts of sequence, conflict, concurrency, synchronisation, confusion, and merging previously shown in Figure 2.18, and which are explained below.

5.2.3 Representing structural musical processes in Petri Nets

This section will briefly consider how Petri Nets can be used to represent the musical concepts of sequence and repetition, parallelism, synchronisation, nondeterministic choice, and merging.

5.2.3.1 Sequence and repetition

As previously explained, the *SuperCollider* Petri Net implementation (Cardoso, 2014) substantially differs from the Baratè et al. (2005) Music Petri Net. Notably, music objects are attached to transitions (rectangles), rather than to places (circles). As an example, Figure 5.4 shows a sequence with two transitions, id:1 and id:3, which are attached to two different audio files. If a token is added to id:0 it enables the firing of transition id:1, which plays the audio file. Once the audio file has finished playing the token will be passed to object id:2, which enables the firing of transition id:3 and the playback of the attached audio file. Finally, a token is added to object id:4. A sequence could be made to loop infinitely by passing the token back to the start of the sequence, or adapted to loop a given number of times (as shown in Figure 5.1).

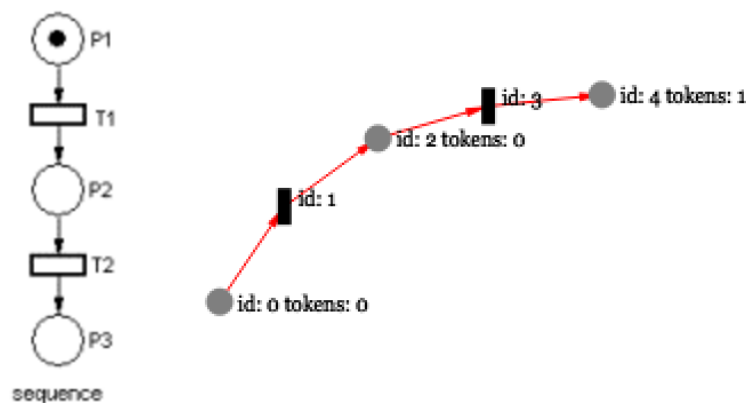


Figure 5.4: Sequencing from Chen (2003) (left) and an implementation in *SuperCollider* (right).

5.2.3.2 Parallelism

Figure 5.5 shows an implementation in *SuperCollider* of what Chen calls ‘concurrency’ (2003) which, in a music context, involves multiple parallel music events running simultaneously. Importantly, this example produces *synchronised* parallel playback of multiple audio files; once a token is added to object id:0 it enables the firing of transition id:1 and the playback of the attached audio file. Once this file has finished playing, tokens are added to three objects; id:2, id:3, and id:4. These, in turn, enable the simultaneous firing of three transitions (id:5, id:6, and id:7) and their attached audio files. While testing concurrent playback with a variety of audio files it was found that the combination of audio files of various durations led to engaging polyrhythmic output.

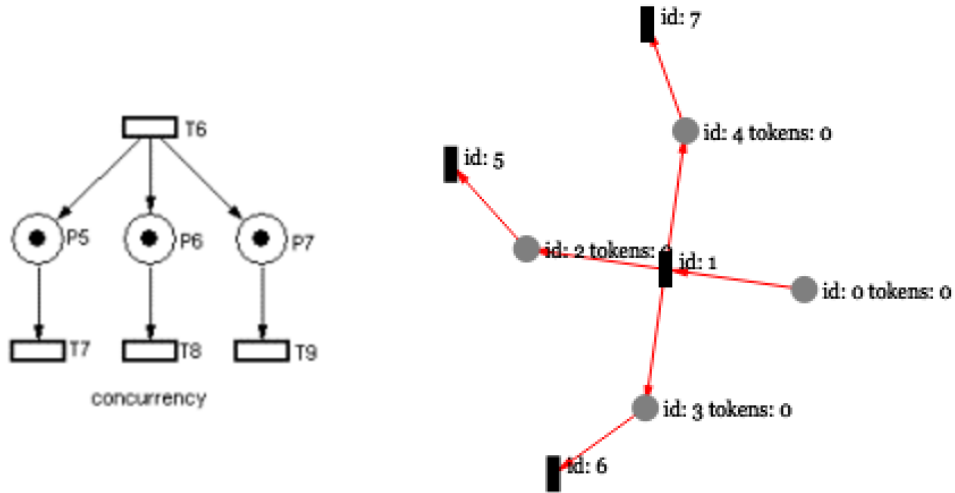


Figure 5.5: Concurrency from Chen (2003) (left) and an implementation in *SuperCollider* (right).

5.2.3.3 Synchronisation

The synchronisation example shown in Figure 5.6 will play the audio file attached to object id: 3 only when the audio files attached to objects id: 0, id: 1, and id: 2 have finished playing.

A musical analogy would be to consider the three incoming places in Figure 5.6 as musicians playing individual parts, with the outgoing place representing a fourth musician. The first three musicians would play their parts in parallel, and the fourth musician would wait until all three had finished playing before starting to play their part. This concept will return in the context of stop behaviour in Section 6.2.

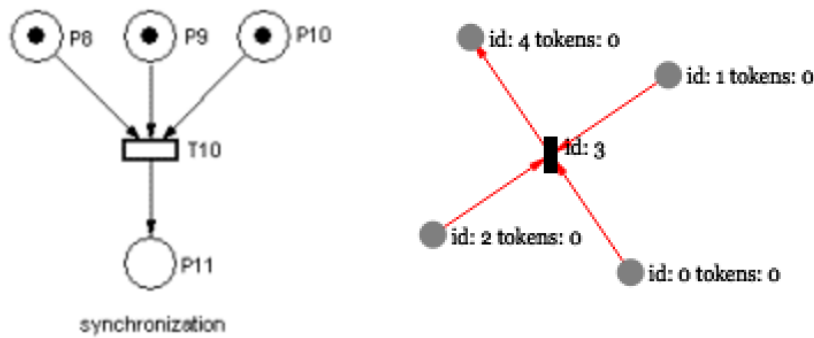


Figure 5.6: Synchronisation from Chen (2003) (left) and an implementation in *SuperCollider* (right).

5.2.3.4 Nondeterministic choice

Section 2.6 presented a review of some uses of nondeterministic processes in algorithmic composition, and a number of the pieces of software analysed in Chapter 4 make use of nondeterminism. Allowing for nondeterministic processes is a key goal for the composition system under development.

In the Chen example labelled ‘confusion’ (shown in Figure 5.7) there are two objects, each containing one token, which enable three transitions. However, when one transition fires it will remove

the token from the incoming object; this will result in the firing of only two of the three transitions. In the implementation in *SuperCollider*, tokens can be added to objects `id: 0` and `id: 1` which should trigger the same nondeterministic choice as outlined by Chen. However, due to an apparent bug in the Cardoso (2014) *SuperCollider* implementation, the PN will always select the first node, leading to the enabling and firing of transitions `id: 2` and `id: 3` on every run. Given this project's stated interest in nondeterminism, this proved to be a significant problem.

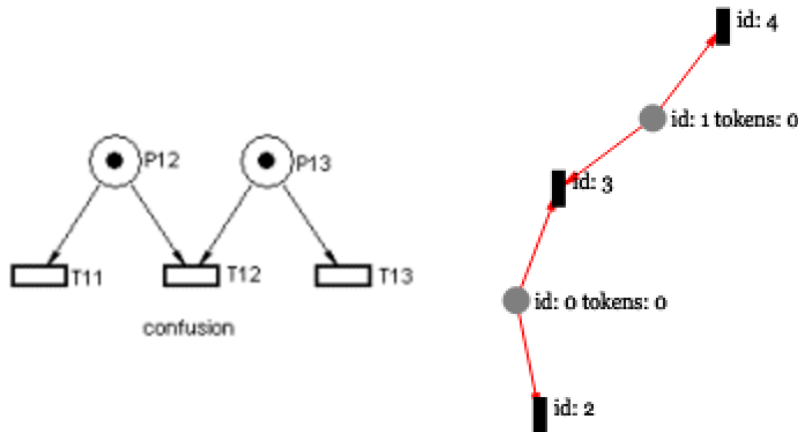


Figure 5.7: Confusion from Chen (2003) (left) and an implementation in *SuperCollider* (right).

5.2.3.5 Merging

The Chen ‘merging’ Petri Net (see Figure 5.8) shows three transitions connected via arcs to a single object. Any one transition will pass a token to the same object. When applied to music, a PN can be created in which a new music event is triggered as soon as the first incoming music event completes. This design will return when considering musical stop behaviour in Section 6.2.

The *SuperCollider* example expands this to include an incoming object to each of these three transitions. In this example, adding a token to any one of objects `id: 0`, `id: 1`, or `id: 2` will result in the enabling and firing of the connected transition (e.g. `id: 3`, `id: 4`, or `id: 5` respectively), each resulting in the playback of an attached audio file. Once playback of the audio file has finished a token is passed to object `id: 6` which enables and fires transition `id: 7`, playing the attached audio file.

5.2.3.6 More implemented musical examples using Petri Net structuring mechanisms

The four *SuperCollider* Petri Nets shown in Figure 5.9 are examples of more complex pieces created by using two or more Chen primitives. These Petri Nets make use of sequence, conflict, concurrency, synchronisation, confusion, and merging. These tests showed that there were minor timing issues in a number of these tests which introduced audible phase and comb filtering effects. Note that, in the Cardoso (2014) *SuperCollider* PN implementation, music events are limited to digital audio samples and, importantly, are attached to transitions. This is in contrast to the Music Petri Net system described by Baratè et al. (2005) in which music events are attached to places.

The PN on the upper left of Figure 5.9, labelled (a), begins with a token added to place `id: 0` at the bottom of the diagram. The sample-loaded transition `id: 1` will then fire, playing the sample. Once the sample has finished playing, concurrent tokens will be passed along two parallel paths, resulting in the synchronised playback of two samples loaded into transitions `id: 4` and `id: 5`. These

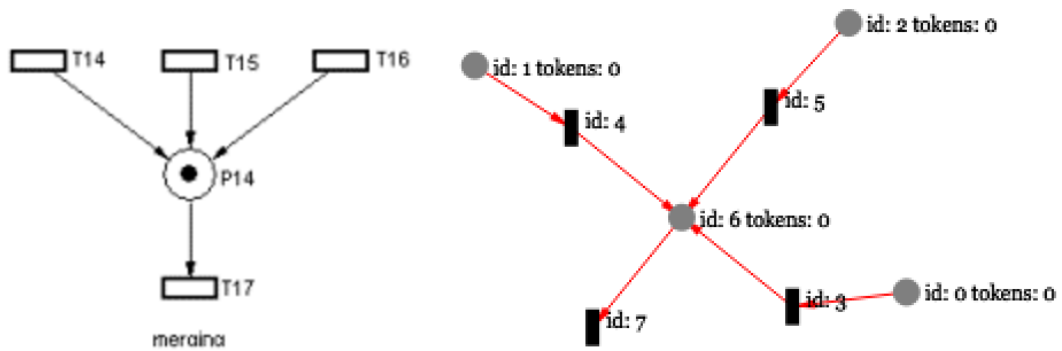


Figure 5.8: Merging from Chen (2003) (left) and an implementation in *SuperCollider* (right).

two paths are then merged to a single transition, id: 8, with the transition fired by whichever of the two incoming paths completes first.

PN (b), on the upper right of Figure 5.9, shows a PN which starts with place id: 0 on the lower right of the diagram. This place leads to a nondeterministic choice with two possible paths; the first containing a sequence of two sample-loaded transitions, and the second containing a sequence of four sample-loaded transitions. Both sequences merge to a common place (id: 11) before playing the final sample-loaded transition, id: 12. Finally, a token is passed to place id: 0 and the PN plays again, making a new nondeterministic selection. The PN will loop indefinitely.

PN (c), on the lower left of Figure 5.9, shows a more complex example which starts with place id: 0 on the left of the PN. The sample-loaded transition id: 1 will fire and, once the sample has finished playing, tokens will be sent down three paths in parallel. Two paths each contain one sample-loaded transition before synchronising at a common sample-loaded transition, while the third path contains a sequence of two sample-loaded transitions. These paths merge at place id: 13, then a nondeterministic choice is made to play one of three sample-loaded transitions (id: 16, id: 17, or id: 18). Finally, a token is returned to the first transition (id: 1) which will play if there are sufficient tokens in id: 0 to allow the PN to repeat. If id: 0 contains no tokens the PN will stop.

PN (d), on the lower right of Figure 5.9, contains two parallel paths; one with a sequence of two sample-loaded transitions, and the other with a sequence of four. The two paths synchronise at transition id: 16, meaning that both paths must complete playback to continue. Finally, a token is passed back to transition id: 1 which can only fire if there is at least one token loaded in place id: 17; if there is, the PN will repeat. If not, the PN will stop.

5.2.4 Design heuristics, observations, and design decisions resulting from the PN design exercise

Based on the above design exercises, and considering design issues and tradeoffs that emerged, the following design heuristics, observations, and design decisions for a nondeterministic composition system for non-programmers were identified.

Musical scope in simple nondeterministic routines The process of testing various Petri Net designs showed that the results from using simple Petri Nets with non-specialist source material could be musically engaging. For example, a small number of simple samples used in a Petri Net containing nondeterministic choice and variable lengths of cyclic sections led to interesting polyrhythmic patterns. Given that the target users are non-programmers they are unlikely to have considered music from an algorithmic perspective. Previous sections have considered the linear frame of reference created by common music notation

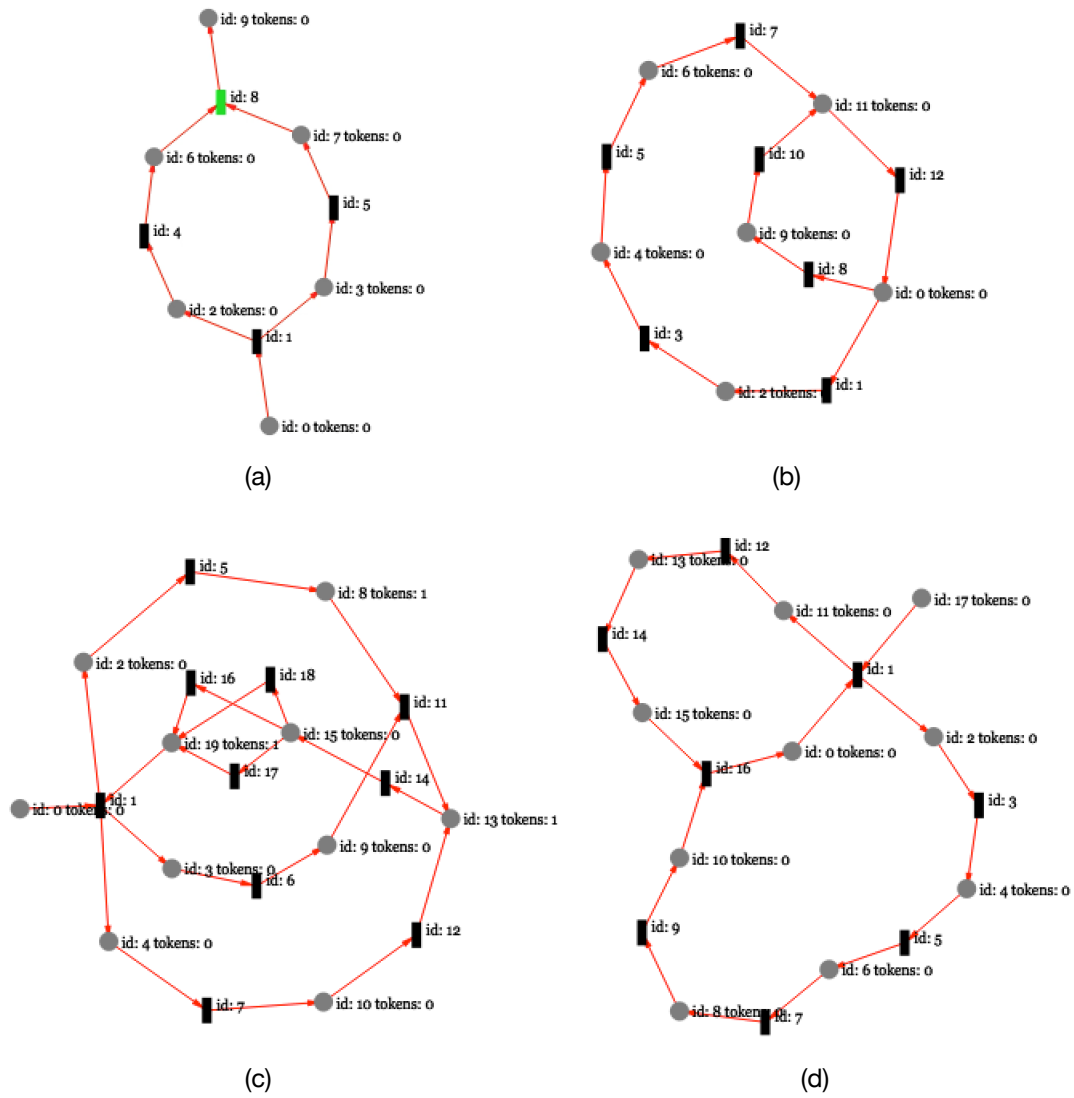


Figure 5.9: Four Petri Net tests created in *SuperCollider* (McCartney, 2002) using the *Thr44* quark (Cardoso, 2014).

and phonography (see Sections 2.2, 2.3) and it can be argued that a notation which surfaces nondeterminism as not only possible but expected may allow for a significant shift in the approach and expectations of end users.

Visualisation is useful for music A visual notation is well-suited to the representation of musical structure. However, the use of tokens, places, and transitions do not offer a clear match with most music processes and so can be both distracting and confusing.

Music Petri Nets are not suitable as a front-end Music Petri Nets are not suitable for the stated target users to use as an interface; while some musical patterns are quite simple to implement, some common musical moves (such as infinite looping vs. a set number of loops) are made unnecessarily complex. However, Music Petri Net software could be used as an intermediate model if paired with a more suitable user-facing interface. Music Petri Nets, as outlined by Baratè (2009), offer sufficient functionality to implement the playback of the interface design outlined from Section 6.1.

The *SuperCollider* PN is not fully realised As previously stated, the *SuperCollider* PN quark does not implement the Music Petri Net system. In addition, there were some faults in behaviour which resulted in deterministic rather than nondeterministic playback, and timing issues which led to comb filtering issues on playback. Therefore, while *SuperCollider* has a number of desirable attributes, the *Thr44* PN implementation is not suitable for purposes of this project.

In summary, the review of Petri Nets and Music Petri Nets—including the use of *PIPE*, *Pure Data*, and *SuperCollider* with the *Thr44* quark—proved useful when considering the nondeterministic musical operations appropriate for novice users. Some simple Petri Nets were capable of engaging and surprising results. While the *Thr44* quark did not fully implement Music Petri Nets, *SuperCollider* proved to have a number of desirable attributes for the playback, creation, and manipulation of music. As a result, create custom *SuperCollider* classes were chosen to implement the back-end of the interface design.

5.3 Conclusions

As described in Section 3.2, a Research through Design approach will be adopted to the creation of a nondeterministic music composition notation and user interface suitable for novice users. Accordingly, a range of prototypes will be developed and analysed in order to better understand the design space. At this point it is instructive to consider a condensed summary of the collected heuristics, observations, and design decisions from:

- The CDN review in Section 5.1.1;
- The DC design exercise in Section 5.2.1;
- The PN design exercise in Section 5.2.2.

These findings will inform the work in subsequent chapters: specifically, Tables 6.1, 6.2, 6.3 map the heuristics from this chapter to the notation design that will be outlined in Chapter 6.

The work required of the user should be minimised. The design needs to be mindful of the existing knowledge and experience of a user who has little to no programming experience, and who may have minimal formal musical knowledge.

The CDN analysis identified the importance of minimising the learning required of the user, and an exploration of Direct Combination showed the benefits of applying a parsimonious approach to the interface in order to constrain the number of interface elements that the user needs to learn. Semantic consistency should be applied to make the interface easy to learn, allowing existing knowledge to be applied in multiple contexts. The power of the interface will be leveraged via the combinatorial power of linking a small number of interface elements, rather than allowing a large number of interface elements.

The CDN analysis suggests that, in order to meet the needs of non-programmers with little music knowledge, domain-specific metaphors with which the user may not be familiar should be avoided. Visual complexity should be reduced in order to enhance visibility and juxtaposability. While powerful, Music Petri Nets make overly complex some common musical moves, such as looping—explorations of both DC and PNs suggest that the most relevant musical moves should be surfaced where possible.

In order to enhance the musical expressivity of the interface, multiple compositional approaches and easy arrangement should be allowed. Users will learn about the possibilities of nondeterministic composition from being able to easily experiment and audition music which uses nondeterministic patterns, and compositional flexibility should be maintained while the user is working. Even simple nondeterministic patterns can be musically meaningful, and there is significant musical scope in combining sequence, synchronisation, concurrency, duration, and weighted nondeterminism.

Chapter 6 will use the findings outlined in this chapter to inform the development of a new abstraction specifically designed for nondeterministic music arrangement.

Chapter 6

Design and development of the preliminary version of Choosers

This chapter presents the design of Choosers v1, a new abstraction capable of sequencing, nondeterministic choice, parallelism, multi-selection, and looping. This design, excluding variables, will form the basis of the first user study outlined in the next chapter. After presenting the design, the development of Choosers v1 is outlined, showing how it is built on the findings from the design exercises of the previous chapter. The development is shown via a series of design sketches, including alternative implementations and a consideration of associated trade-offs.

As seen in the previous two chapters, there are many systems for algorithmic music composition, but generally users have to put up with one or more of the following limitations:

- Users are expected to have programming skills;
- Users are expected to have musical training;
- Users are not afforded fine control over musical structure at all levels.

Chapter 1 introduced the aim to devise and evaluate a system for nondeterministic music composition that will lift all three of these limitations. In pursuit of this goal, Chapter 3 proposed an overall research methodology and Chapter 4 applied a Cognitive Dimension of Notations analysis to a representative range of existing algorithmic music composition systems (and similar) in order to inspire design heuristics for this specific goal. Chapter 5 presented design exercises to further sharpen and extend the needed design principles. The present chapter will first outline the preliminary version of Choosers, a new abstraction, before showing how the design process applied the collected derived principles via a series of focused design exercises.

6.1 Implementation

As a result of the development work outlined in Chapter 5 I finalised a notation, named Choosers, for nondeterministic music composition. This new notation will be tested with end users in Chapter 7. In order to understand the results of the user tests this section provides an overview of Choosers v1 as they were at the time, including a brief outline of pre-final design sketches where they are useful in understanding the design principles. In line with this aim, this chapter is written partially in a reference style. Most, but not all, of the features outlined in this chapter are tested in Chapter 7. A full user guide for the design is shown in Appendix A. The development of this initial version of Choosers will be outlined from Section 6.2.

6.1.1 Sequence

Samples are shown in boxes (see Figure 6.1). Boxes can be freely placed (spatial placement on the page has no effect on semantics) and can be auditioned via a single click. Samples can be assembled into *sequences* using arrows (see Figure 6.2). 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 6.3 shows a sequence (Melody1.wav then Melody2.wav) packaged into a box. This box is used at the start of a sequence, with a duplicate of the box used at the end of the sequence. As the order of the sequence is indicated by the direction of the arrows the boxes can be freely placed, and this sequence contains a mixture of left-right and top-down arrows. Note that the sequence shown in Figure 6.3 represents poor style due to the needless duplication of a sequence. Abstraction techniques will be introduced in Section 6.1.6 to deal with this particular problem (see Figure 6.8).

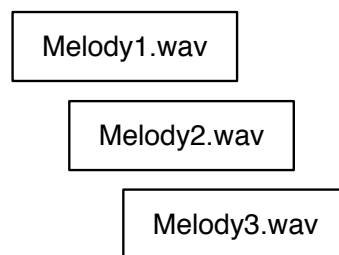


Figure 6.1: Samples are shown in boxes.

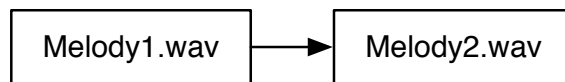


Figure 6.2: A sequence is assembled via arrows.

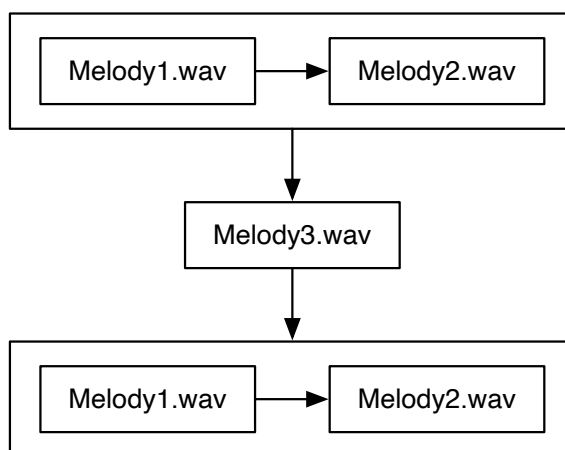


Figure 6.3: A sequence, packaged into a box, and used twice in a sequence. Note that the order of the sequence is indicated by the direction of the arrows. In this instance, the sequence inside the upper box (Melody1.wav then Melody2.wav) will play before the contents of the next box (Melody3.wav) are played. Finally, a duplicate of the first box will play.

6.1.2 Nondeterminism, choice, parallelism and multi-select

Boxes containing samples or sequences can be snapped together vertically to create what are known as *Choosers*. Figure 6.4 shows a Chooser with two lanes, each containing a sample—in this case, a drums sample and a bass sample. The number 1 in the nose cone indicates that, at run time, just one of the lanes will be selected nondeterministically (subject to restrictions described below). On different runs, different choices may be made. In Figure 6.4, by manipulating the number in the nose cone, any number of lanes from 0 to 2 can be randomly chosen, and will be played simultaneously. More generally, 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. Consequently, in Figure 6.4, the drums are twice as likely to be chosen as the bass. By setting weights in all except one lane to zero, it is always possible to make a Chooser behave deterministically. Additionally, weights of ‘A’ (‘always play’) can be used to ensure that one or more lanes are always selected for playback. In cases where two or more lanes have been marked ‘always play’, both will always be selected for play—to make sure this is possible, in such circumstances the nose cone will take a default value equal to the number of such lanes, and will not accept a lower value. However, the user may still set the nose cone to zero (i.e. ‘select no lanes’) even when one or more lanes are set to ‘always play’.

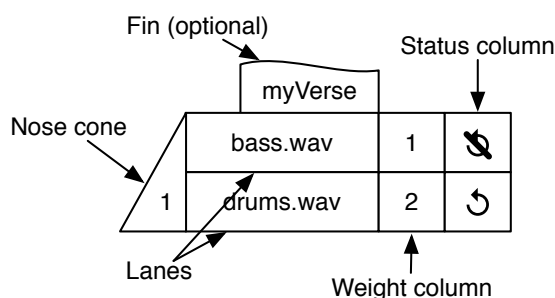


Figure 6.4: An annotated Chooser

6.1.3 Looping

Any sample can be set to *loop* indefinitely when selected on a particular run, or to play just once by the choice indicated in the status column (shown in Figure 6.4). Indefinite looping of a single sample may not always be desired, and for this reason I now introduce Time Choosers. As a Soundable Chooser provides control over soundable content, a Time Chooser provides control over duration. An example of a Time Chooser is shown in Figure 6.5; note that the nose cone of the Time Chooser slopes up to distinguish it from the downwards slope of the Soundable Chooser nose cone. The nose cones also make visually clear cases where both Chooser types are combined, shown in Figure 6.6 and explained in Section 6.1.4.

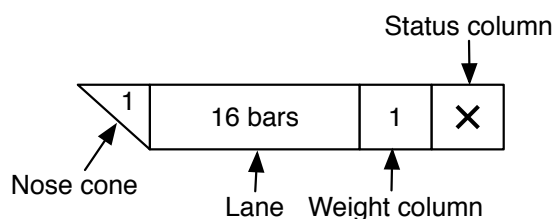


Figure 6.5: An annotated Time Chooser

6.1.4 Deterministic duration and stop behaviour

If the Time Chooser shown in Figure 6.5 is attached onto the bottom of the Chooser shown in Figure 6.4, this produces the *Full Chooser* shown in Figure 6.6.



Figure 6.6: A Full Chooser

When the Full Chooser shown in Figure 6.6 is played, looped drums, if chosen, cannot play indefinitely but instead will be cut off after 16 bars; the Time Chooser’s hard stop sets an exact duration regardless of the length of the audio sample. However, if the status column in the Time Chooser were set to $>$ (indicating a *soft stop*) rather than \times (indicating a *hard stop*) then, after 16 bars, the sample would play to the end of its current iteration. With a hard stop, if the Time Chooser duration cleanly divides the sample duration, every repetition will play in full. If not (e.g. if the `bass.wav` sample in Figure 6.6 had a duration of 3 bars), a hard stop will cut playback mid-sample. The two kinds of stop work similarly with non-looped lanes. If the non-looped bass lane of the Full Chooser (Figure 6.6) were chosen, the bass sample would be guaranteed to start playing once. With either kind of stop, if the sample were less than 16 bars long there would be silence after sample playback until the end of the 16 bars; if the Chooser formed part of a sequence, the sequence would then continue. A non-looped sample longer than 16 bars would be truncated by a hard stop but would be allowed to complete by a soft stop.

Now that Time Choosers and Full Choosers have been introduced, where it helps to avoid ambiguity, Choosers with no attached Time Choosers, such as those shown in Figure 6.4, will be referred to as *Soundable Choosers*.

6.1.4.1 Rests

A Time Chooser can be used alone as part of a sequence—however, when used in this way it will simply result in a musical rest of the specified duration.

6.1.4.2 Nondeterministic duration

More generally, the purpose of a Time Chooser within a Full Chooser is to moderate, in a non-deterministic manner, how long the Soundable Chooser and its individual lanes play. Possible interactions between the settings of Soundable and Time Choosers can make the results more varied than might be imagined.

A Time Chooser’s nose cone can be set to either one or zero. If set to one, one 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 arrangement changes, with the possibility of infinite playback if the Soundable Chooser lanes are set to loop. In this case, 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.

6.1.5 Arrangement

The system is designed to allow 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, working down to the note level. A bottom-up approach is the opposite: the user begins at the phrase level and builds musical sections before organising the macro structure of the piece. Examples of top-down and bottom-up construction using this generation of Choosers can be seen in Section A.3.2 as part of the user guide reproduced in Appendix A.

6.1.6 Nesting and referencing Choosers

One Chooser can be nested inside another Chooser. Figure 6.7 shows a nested child Chooser inside the third lane of a parent Chooser. On playback, the parent Chooser will select and play back one of Melody 1, Melody 2, or the result of the child Chooser. If selected, the child Chooser will play either Melody 3 or Melody 4.

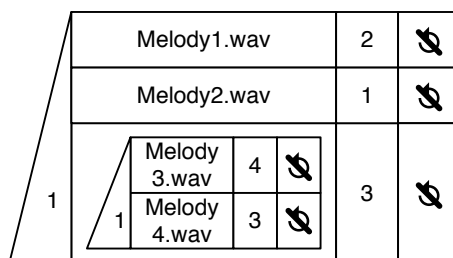


Figure 6.7: An example of a nested Chooser.

An alternative to nesting is to reference a box or Chooser's name inside a box or a lane of a parent Chooser in any combination. An example showing naming and nesting boxes is shown in Figure 6.8. This example is functionally identical to the nested example shown in Figure 6.3; a sequence is packaged via a named box (Melody1.wav then Melody2.wav in a box named myBox), which is then referenced from boxes in another sequence.

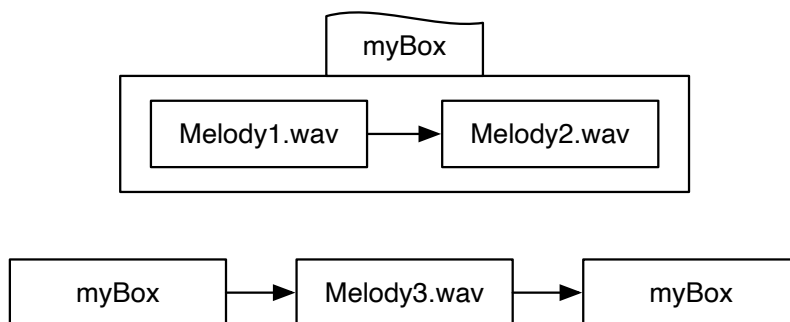


Figure 6.8: A version of Figure 6.3 which replaces nesting with referencing. The sequence packaged into the box named myBox is referenced twice in the sequence at the bottom of the figure.

An example of referencing Choosers is shown in Figure 6.9, in which a Chooser (top) references a second Chooser, named 'myMelody', in the third lane. The Chooser containing the reference is the parent Chooser, and the referenced Chooser is the child Chooser. This example is functionally identical to the nested Chooser example in Figure 6.7; one of three lanes in the parent Chooser will be selected and, if the third lane is selected, the child Chooser will select one of its two lanes. Referencing a Chooser from inside the lane of another Chooser can be preferable to visual nesting (of the type seen in Figure 6.7) in some situations, and the user is free to use either nesting or

referencing in any combination. A Chooser does not need to have a name, but where it does, its name should be unique. The name of a chooser, if it has one, is displayed in its fin (e.g. in Figure 6.9).

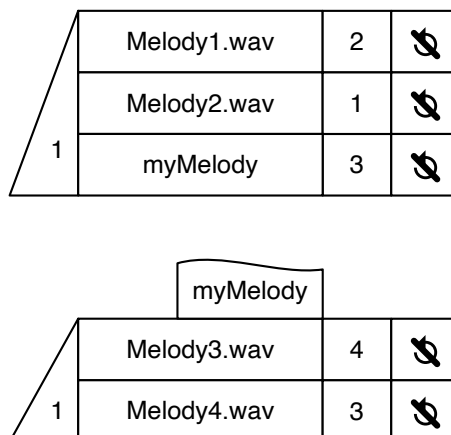


Figure 6.9: An example of referencing a Chooser's name inside the lane of a parent Chooser. This example is functionally identical to the nested example in Figure 6.7.

6.1.7 Variables and Variable Choosers

Variable Choosers allow the user to reuse material (e.g. a number, sample, or Chooser) programmatically by defining it once and then creating multiple references to the single entity. A Variable Chooser is run afresh each time it is called, potentially resulting in a different selection—a new 'roll of the dice'. In keeping with the overall design approach of Choosers, variable assignment may be carried out nondeterministically by using a Variable Chooser with weights (see Figure 6.10). In the case of a Variable Chooser, the fin names the variable whose value is to be assigned by that Chooser. This contrasts with Soundable and Full Choosers, whose fin names always refer to just one thing—the Chooser itself. Importantly, a Variable Chooser's nose cone is fixed to 1; only one lane can be selected. The rounded shape of the nose cone, shown in Figure 6.10, visually distinguishes it from the editable nose cones of Soundable and Time Choosers while maintaining consistent design and maximising readability ('select one lane').

Figure 6.10 shows a Variable Chooser named `myVar` with two lanes, each containing a number. The nose cone has a fixed value of 1 and cannot be edited. Each lane has a weight column on the far right; in this instance, the number 2 is twice as likely to be selected as the number 1. By default, a new selection of one of the two lanes will be made each time `myVar` is referenced by another entity (e.g. in a Chooser's nose cone or weight column).

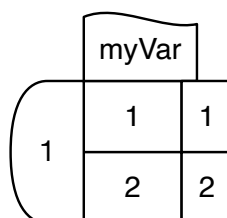


Figure 6.10: A Variable Chooser called `myVar`.

Variable Choosers can refer to the following types of data:

Numbers Variables can be assigned to numbers. An example is shown in Figures 6.10, 6.11.

Status A Variable Chooser can be assigned to a status which may be applied to a Soundable Chooser lane (i.e. loop or non-loop) or of a Time Chooser lane (i.e. hard or soft stop).

6.1.7.1 Pinning a variable value

While Variable Choosers select afresh each time they are called, there are some instances when the user may want to reuse the same result from a variable. For example, if one melody was chosen for a musical section the user may want the same melody to be used later in the piece. This can be done by appending an '@' and a user-defined ID, in the form of a text string, to the end of the variable name when it is used. Every time the user refers to a variable using the appended name the same result will be returned, allowing for the same output each time.

6.1.7.2 Example: using pinned variable values

Figure 6.11 shows the Variable Chooser myVar (shown previously in Figure 6.10) in use. In this example there is a sequence of three Choosers (First, Second, Third), with both First and Third using multiplication notation to control the number of repeats before the sequence continues to the next Chooser, or the piece ends. The number of repeats will be taken from the output of the Variable Chooser myVar (\times myVar).

If the variable name is used without appending a number or name then the variable Chooser will re-run each time it is used; this results in a 'fresh roll of the die' and a potentially different result. If the user would like the same selection to be used for both repeats they can 'pin' the value by using the Name@IDformat, with the name being the name of the variable (in this case, myVar) and the ID being a name given by the user. An example of this is shown in Figure 6.12, in which both references to the Variable Chooser named myVar will use the same selection as both references to the Variable Chooser are appended with the same ID, which in this case is @sectiona.

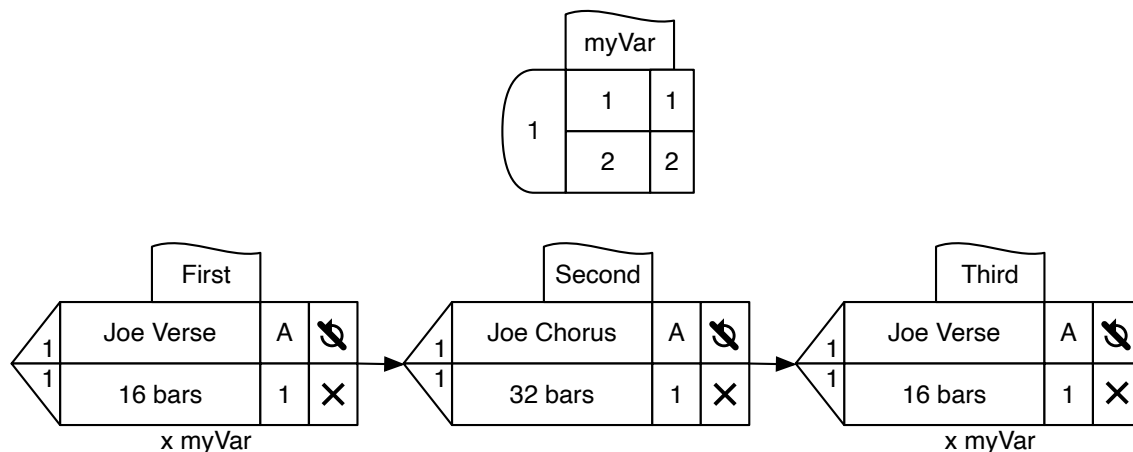


Figure 6.11: The variable myVar is used to control the number of repeats of two Choosers. The variable selection will re-run each time the Variable Chooser is referenced, which may lead to different values being used for each repeat.

6.2 Preliminary sketches in the design space

Now the design of Choosers v1 has been presented, this section will outline the design exercises that led to the decisions embodied in Section 6.1. For the reader's convenience, the set of derived principles and heuristics that were derived from the work presented in previous chapters, and which informed the work presented here, are summarised below:

- Minimise the learning required of the user, especially domain-specific knowledge which novice users are unlikely to have;
- Apply a parsimonious approach to interface design, constraining the number of interface elements that the user will need to learn;

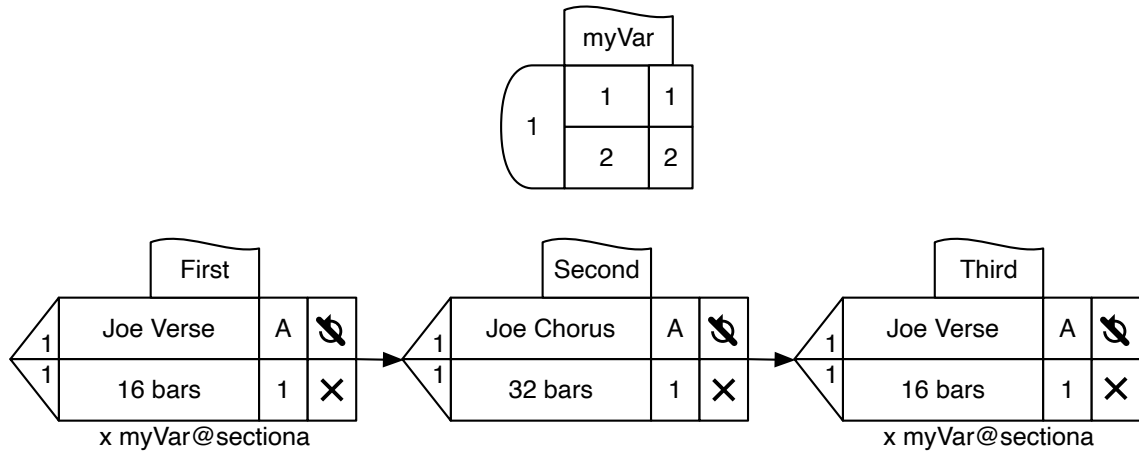


Figure 6.12: Both references to the Variable Chooser append a user-definable string of text (in this case, @sectiona) to the end of the name, showing that the same selection should be used for both.

- Apply semantic consistency to the interface design so learning in one context can be applied in another;
- Offer advanced functionality via progressive disclosure and the combined use of a small number of interface elements;
- Provide a simple interface to enhance visibility and juxtaposability;
- Bring common musical moves to the surface of the interface, and allow more advanced functionality via combination;
- Allow for multiple compositional approaches—specifically for top-down and bottom-up construction—and maintain compositional flexibility as the music becomes more complex;
- Offer an interface which will allow novice users to make quick changes, audition, and experiment with nondeterministic music.

6.2.1 Abstracting from Music Petri Nets

The design exercises outlined in Sections 5.2.1, 5.2.2 present the findings from music-specific explorations of Direct Combination (DC) and Petri Nets (PN), including Music Petri Nets. As part of the design exercises annotated in Chapter 5 Petri Nets were created in *PIPE* (Tattersall and Knottenbelt, 2014) and *SuperCollider* (McCartney, 2002).

In order to create the initial abstraction some aspects of tabular spreadsheet design were used which, as demonstrated by Nash (2014) and noted in Section 2.11, have particularly interesting potential advantages for non-programmers as they provide cues to the user to show what should be entered. These elements form part of a design which aims to hide the visual complexity of the Music Petri Net while surfacing the most relevant musical functionality to the user; an early sketch is shown in Figure 6.14.

Taking Zappa's *Peaches en Regalia* as a working example, each section in Figure 6.14 is populated with an abstraction that represents three possible routes. This mirrors the transposition functionality of the Music Petri Net by Baratè (2009), shown in Figure 6.13; the + and – lanes represent transpositions. Rather than notating weights next to incoming arcs, as in Music Petri Nets, the lanes in the new abstraction have a weight column on the right—note that the weights of those lanes that represent transpositions in Figure 6.14 are set to zero, meaning that they are not currently selectable by the system. This is to match the behaviour of the Zappa Music Petri Net shown in Figure 6.13 and explained in Section 5.2.2, where the transposition paths are not currently active due to a lack of tokens in their corresponding objects.

This simplification of Music Petri Nets suggested an alternative notation for weighted non-deterministic choice (i.e. the use of a weight column to replace weighted arcs), and subsequent

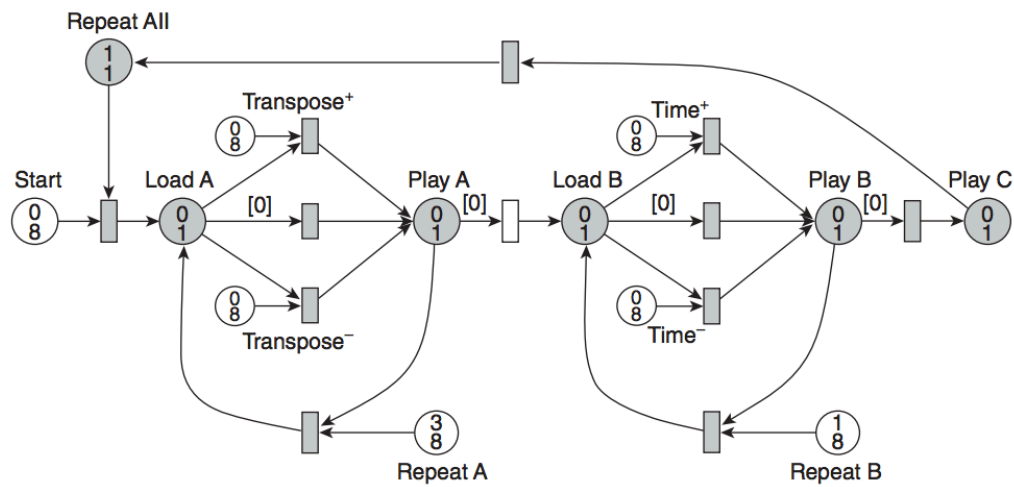


Figure 6.13: *Peaches en Regalia* by Frank Zappa represented as a Music Petri Net (Baratè, 2009). This is a duplicate of Figure 5.1 reproduced here for the reader's convenience.

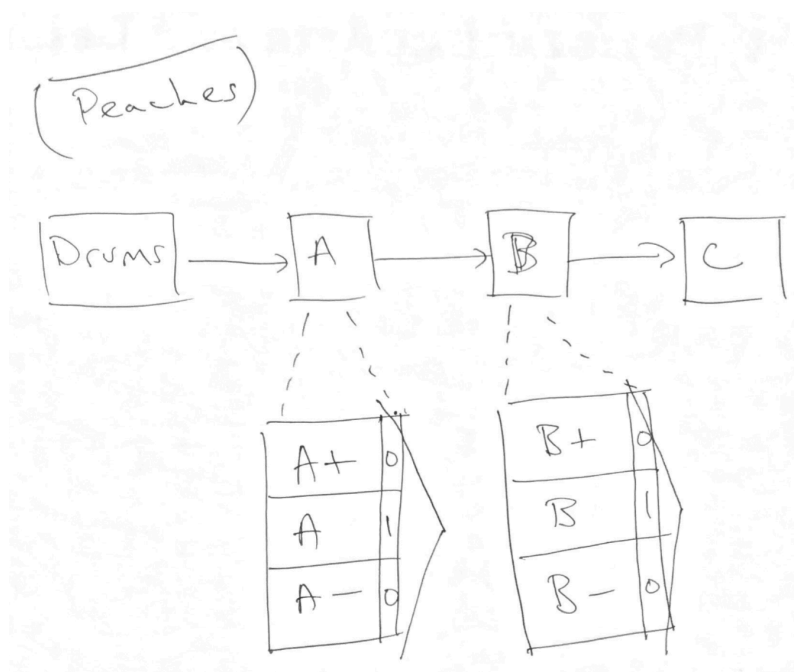


Figure 6.14: A simplified version of the *Peaches En Regalia* Music Petri Net (Baratè, 2009) shown in Figure 6.13. This version uses a flowchart-style musical arrangement and spreadsheet-style abstractions to represent the transposing sections and the weighting of nondeterministic choices.

design exercises considered scenarios to test this approach. A sketch to explore a rough design for synchronous playback is shown in Figure 6.15. As in Figure 6.14, a macro view of the musical arrangement is shown across the top, with arrows denoting order. Each object in the arrangement refers to a named object (A, B, and C), shown as boxes in the second row of Figure 6.15. Each of these boxes contains an abstraction which will select one of three possible lanes; as in the previous example, the intention is to play the section unchanged or to allow for transposition up or down. One of the lanes will be selected, and the selection is controlled via the weight column; as in Figure 6.14, the weights are currently set to allow only the non-transposed version to be selected. The example in Figure 6.15 adds a second level of nesting—the macro arrangement is shown at the top, with references to three nested objects (A, B, and C). The ‘Section 1’ object is, in turn, nested in the ‘A’ object. Colour is used to group samples into sections for selection; one sample from each of the five coloured sections will be selected (i.e. one of three drum samples, one of two bass samples, etc.). The selected samples—one from each coloured section—are then played concurrently.

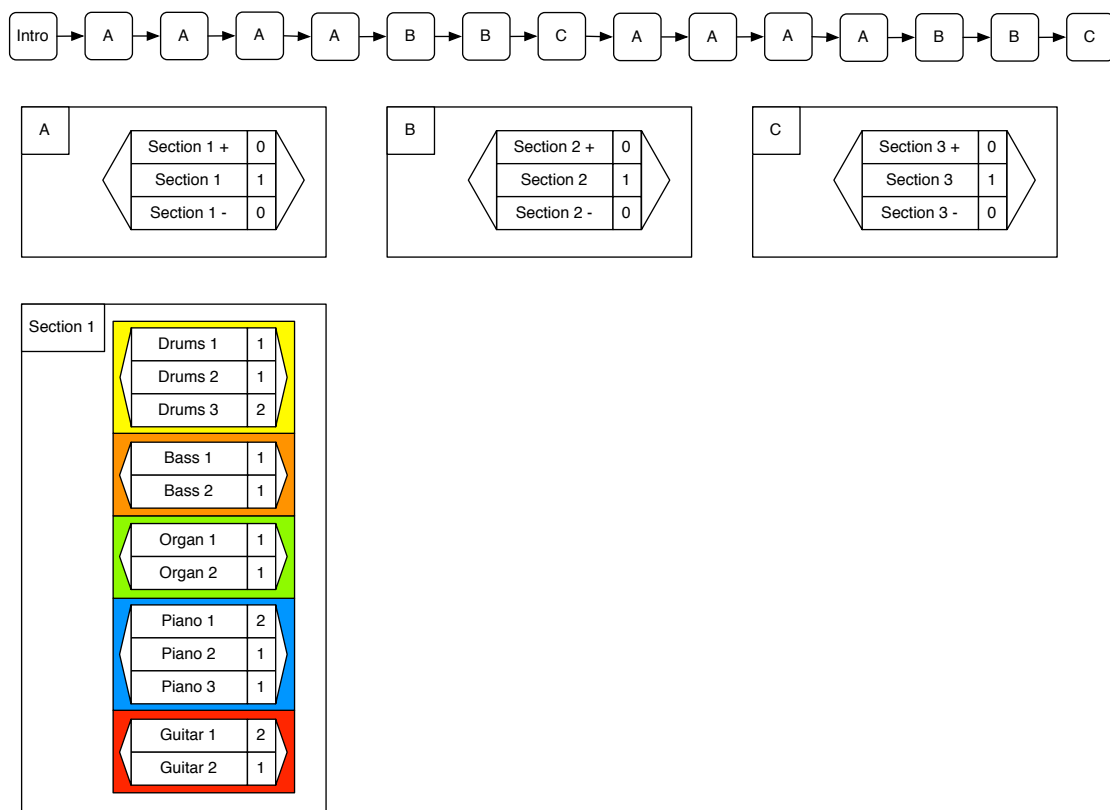


Figure 6.15: A design sketch extending the ideas behind Figure 6.14 to include explicit use of audio samples, nesting, concurrency and weighted nondeterminism.

This concept is expanded further in Figure 6.16, which encompasses concurrency, synchronous playback, nondeterminism, and weighted choice. As in Figures 6.14, 6.15, the user has a macro view of the musical arrangement at the top of the image, allowing for quick changes to the arrangement of the piece. Each section of the arrangement overview is populated via a box—Figure 6.16 shows the contents of ‘A’ and ‘B’. ‘A’ contains a single musical element—‘Drums’—which in turn is populated by another box which contains two samples, drums1 and drums2. The word ‘option’ was used in this sketch to show that only one of the samples will be selected. The two samples have a selection weight of 50%, making them equally likely to be selected, and the selected sample will repeat four times. By contrast, the ‘B’ section contains two concurrent musical elements, ‘Bass’ and ‘Drums’. The bass part consists of one of three possible samples, and the chosen sample will play until the drums finish. The ‘Drums’ part similarly consists of one of three samples, and the selected sample will

repeat either two or four times before stopping. Note that no selection weights have been specified in this instance, resulting in an equal weighting for all possible selections.

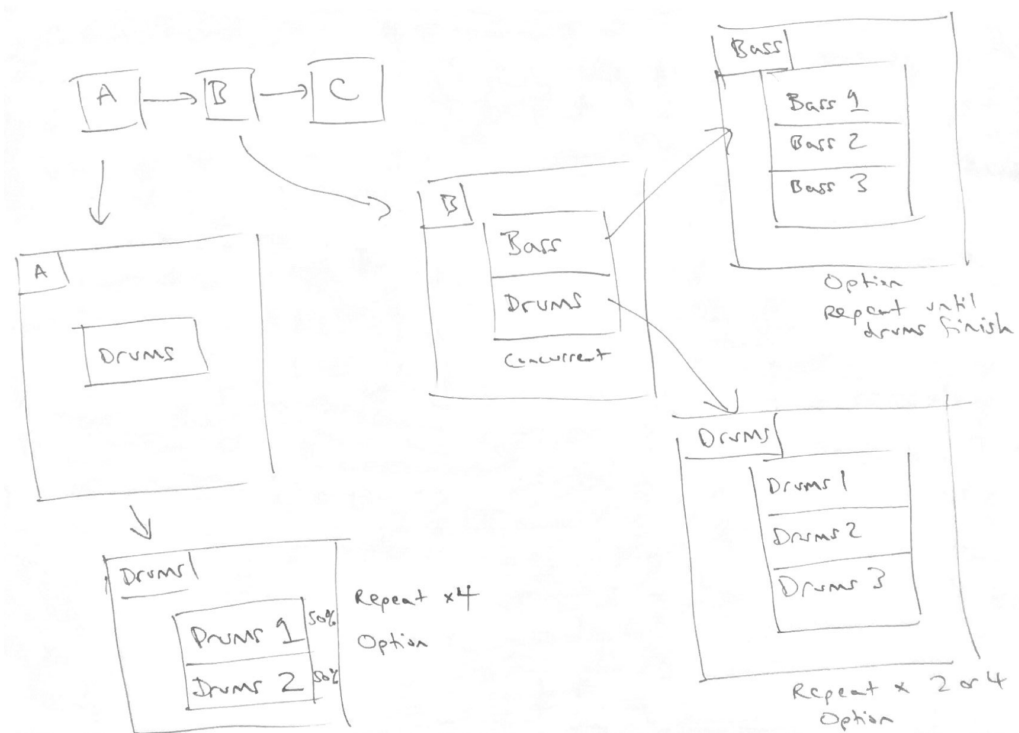


Figure 6.16: A test which further explores the sequencing and abstraction elements from Figure 6.14 and Figure 6.15. A sequence of musical sections is shown in the upper left, with the sections populated via named abstractions. The abstractions, in turn, are populated by instruments/samples which are selected via weighted nondeterministic choice.

One weakness of the notation as sketched in Figure 6.15 is that although, for example, in section 1 the diagram expresses a choice between which drums will play, the number of samples that will play simultaneously in ‘Section 1’ is fixed by the current diagram at five (namely one selection each among bass, drums, organ, piano, and guitar). In order to resolve this issue, a multi-choice mechanism, added to Figure 6.17, was designed to work in the following way. A triangular object on the left of the lanes acts as a brace to group the lanes together visually; the intention is to show that by default all connected lanes will play simultaneously. The number in the triangle shows how many lanes will be selected to play, in a similar way to the number of tokens added to a Petri Net. The name proposed for this triangular element is the *nose cone*. Music Petri Nets have weighted arcs in order to allow for weighted nondeterministic choice and, to allow for this kind of behaviour, a weight column has been added to the right of the lanes.

Each lane contains an instruction to ‘repeat until drums finish’. This behaviour mirrors that of the Chen ‘synchronisation’ Petri Net previously discussed in Section 5.2.3.3; a more considered design will be presented in Section 6.2.4.

6.2.2 Considerations for nondeterministic weighted choice

Figures 6.14, 6.15, 6.17 make use of a new abstraction, the key function of which is nondeterministic choice. This abstraction will be referred to as a *Chooser*.

Section 5.2.2 noted various structural features that occur in music that Petri Nets can represent concisely, and the fledgling representation under development should ideally be able to represent at least these features. At the same time, Petri Nets do not always represent these features intuitively, and it is hoped that the developing representation may be able to do better for musical purposes.

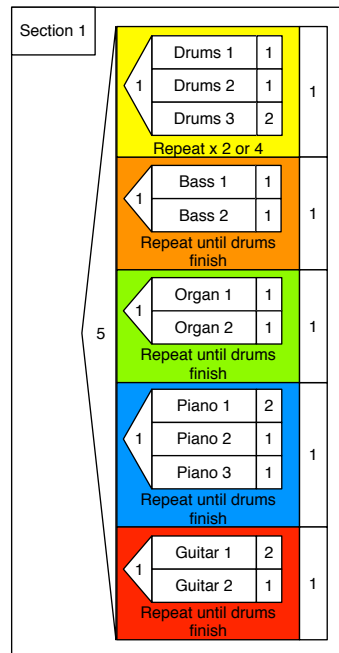


Figure 6.17: An abstraction which combines both a nose cone on the left and a weight column on the right.

Consequently, it is helpful to review the way these structural musical features can be represented in the two systems.

Sequence Early designs of Choosers used simple graphical shapes where possible. In order to allow for both top-down and bottom-up construction and low-viscosity changes in structure a simple graphical sequencing design was adopted: a *single* arrow can enter or exit each element in a sequence, with order indicated by the direction of the arrows. Multiplication notation (e.g. ‘ $\times 2$ ’) is commonly used and understood by musicians (e.g. lead sheets, chord charts) and so is allowed underneath boxes (representing samples) or Choosers. Very early designs made use of branching paths as a mechanism to enable parallelism and choice, but an early decision was taken to maintain structural simplicity as a core design focus; both parallelism and choice were enabled via lanes and nose cones, as shown in Section 6.1.2.

Concurrency In a Petri Net, concurrency is created by sending tokens into places when a transition is fired. As shown in Section 5.2.3.2, if the user requires that three places run concurrently, one transition will be connected to three places. In Choosers, the places are replaced by lanes, and the number of tokens is replaced by the number in the nose cone. The number of lanes which are played in any given Chooser depends on the nose cone, as detailed in ‘conflict’ below. Once selected for playback, all lanes in the same Chooser are played concurrently.

Conflict Conflict is a fundamental part of the design of Choosers. If the nose cone value is less than the total number of soundable lanes there will be conflict, requiring the weighted choice of lanes. Figure 6.18 shows a simple early sketch of a Chooser. The Chooser contains three soundable lanes and has a nose cone value of 2. As such, two of the three lanes will be selected to play. The likelihood of the lane selection is controlled by the *weight column* to the right of the lanes.

Merging Music requires control over the duration of events. Given that Choosers can mirror the functionality of concurrency in Petri Nets, a system to represent the equivalent of a Petri Net merge is also required to complete a Chooser and, if used in a sequence, trigger the next Chooser. A potential implementation of merging is outlined below.

A simple system to control the duration of a Chooser was tested, with initial designs making use of a single duration placed in a lane at the bottom of a Chooser; this is referred to as a *time lane*.

The time lane's role is to stop any playing musical material after the specified duration has elapsed. In this proposal, the time lane is silent and the other lanes will result in soundable playback. If the Chooser were part of a sequence, and if another Chooser was scheduled to run after the currently playing Chooser, the stopping of the first Chooser by its time lane (or by the completion of the sample(s) it contained) would then trigger the second Chooser to begin playing. If a Chooser only contained a duration then it would create a musical rest of the given duration.

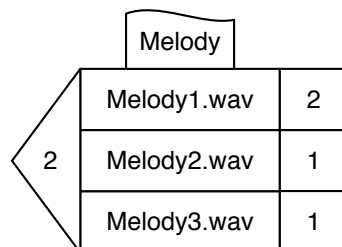


Figure 6.18: An early sketch of a Chooser, showcasing concurrency, conflict, and weighted choice. Note that the nose cone is set to 2, meaning that two of the three lanes will be selected to be played. The weight column (far right) shows that the Melody1.wav sample is twice as likely to be selected as the other two lanes.

In this design iteration, the time lane takes priority and is always the first lane to be selected in a Chooser. Consider Figure 6.19, and note that the nose cone is set to 2, meaning that two lanes will be selected to play. The first lane to be selected will always be the time lane, so Figure 6.19 would result in the selection of the silent time lane plus one soundable lane. If the nose cone were to be set to 1, only the duration would be selected and the Chooser would play silently for the duration; this would create a rest.

An example of an early sketch showing a nested Chooser for nondeterministic duration control can be seen in Figure 6.19, which shows a nested Chooser in the time lane which will result in the selection of a duration of either 8 or 16 bars. The two durations have the same weight, making their selection equally likely. Another example, also allowing for nondeterministic duration, is shown in Figure 6.20; this example shows the use of a name tab (later referred to as a 'fin') to name Choosers and to reference them inside other Choosers. In this example there is a Chooser named 'Melody' which has a time lane populated by the name of another Chooser, 'Length'. This allows for the nesting of a Chooser inside a lane of another Chooser. In this instance, the Chooser named 'Length' will result in the selection of one of two possible durations, with 16 bars twice as likely as 8 bars. The selected duration will then moderate the duration of the soundable content in the parent Chooser, named 'Melody'. Note that only one duration can be selected per Chooser.

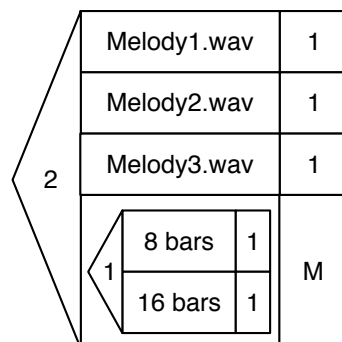


Figure 6.19: An early sketch to explore a design for a time lane, which is always at the bottom of a Chooser. In this instance, the time lane contains a nested Chooser to allow for nondeterministic duration. The status column of the time lane uses 'M' for 'timing master'.

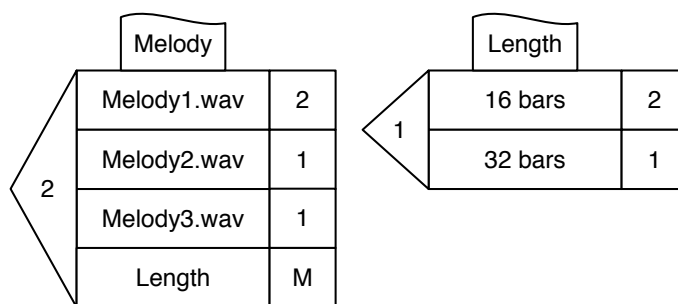


Figure 6.20: An alternative to graphically nesting a Chooser in a lane; this sketch shows how Choosers can be named and referenced from inside other Choosers.

As the system is designed for novice users it would be beneficial to enable nondestructive changes that are easy to revert, allowing users to audition results in order to learn the system via experimentation and to judge musical outputs (see Section 5.1.1).

The design of this preliminary version of Choosers is guided by the heuristics listed in Section 5.1.1; these heuristics were generated from the CDN review presented in Chapter 4. Choosers relate to findings on Direct Combination (Section 5.2.1) by presenting a semantically consistent and parsimonious design which allows for complex functionality via the combinatorial usage of a limited number of widgets. Choosers are capable of mirroring the behaviour of simple Petri Nets but with control over duration and a tabular template to reduce the syntactic burden on the users, who are novice programmers. These characteristics relate to the findings from the Petri Net design exercises (Section 5.2.4).

6.2.3 Looping and its implications

Chapter 5 identified the importance of making structural changes easy for the target user. While not quite universal, repetition is generally fundamental to music. Popular music makes significant use of repeating phrases, and the use of short looped samples has been a key compositional tool in electronic music since the 1980s. To accommodate this a variety of ways of dealing with looping were considered.

Given that users may not be formal musicians I wanted to allow the productive use of audio samples and other files (such as MIDI) with arbitrary durations. Time lanes were already available as a candidate mechanism to constrain the duration of longer files; looping was introduced to allow for shorter files to repeat until stopped. To this end an additional column was introduced; the status column. If a lane contained soundable content the status column would offer a binary loop on/off control. Figure 6.21 shows an early annotated sketch of a Chooser with the status column on the far right.

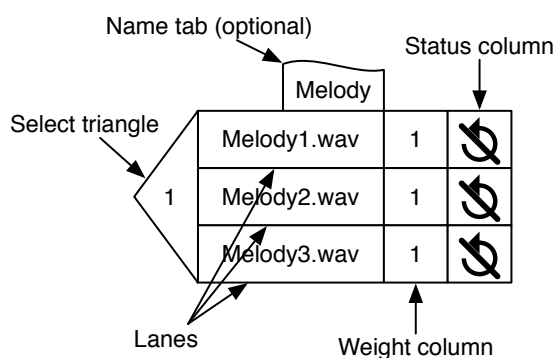


Figure 6.21: An early annotated Chooser, introducing the status column. All three lanes have looping disabled.

The simple addition of a loop on/off control allows for more complex behaviour than might be expected. Figure 6.22 shows a sketch which tests the impact of loops when a Chooser is nested inside another Chooser. The parent Chooser is set to select one of three lanes. If the bottom lane is selected it will, in turn, trigger the selection of one of two samples. Note that all loops are disengaged. Assuming that the bottom lane of the parent Chooser will be selected, the behaviour of the various combinations of loops is as follows:

Parent loop off, child loop off—single choice, play once A lane will be selected, will play once, and the child Chooser will stop. This will stop the parent Chooser and, if there is another Chooser sequenced to play next, it will be triggered.

Parent loop off or on, child loop on—single choice, loop A lane in the child Chooser will be selected and will loop until the Chooser is stopped. As the child Chooser will play until stopped the parent cannot loop it, making the parent loop setting immaterial in this instance.

Parent loop on, child loop off—reselection, loop A lane in the child Chooser will be selected. Once it has finished the child Chooser will stop. At this point the parent Chooser will start it again, prompting a new selection. This will repeat until the Chooser is stopped.

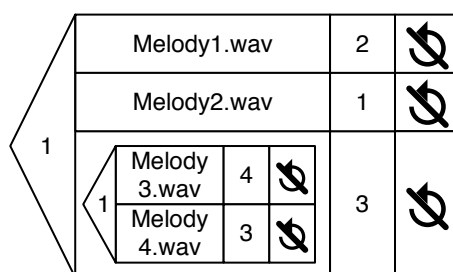


Figure 6.22: A sketch showing a nested child Chooser in a lane of the parent Chooser. All lanes have weights and a status column showing that all lanes have looping disengaged.

6.2.4 Hard and soft stops

At this point in the design process, once the user has the capacity to loop soundable content and can limit duration via a time lane, new designs were created to test the ways in which soundable content might be stopped or have its duration truncated. The motivation here was musical; one musical move is to use repeated phrases which must be stopped on a specific beat or bar, whereas another is to allow phrases to complete before they are stopped. Such moves are made possible by Petri Nets, as shown in Section 5.2.2.

When considering this issue, common practice in digital musical instrument design was reviewed. Samplers are dedicated hardware or software instruments which play back recorded audio samples (Russ, 2009). They typically allow for the user to load or record their own samples, and to control how they will be played back. Most samplers have two modes for sample playback; one mode will play the sample while the user holds down a key or button, stopping the sample when the key or button is released. The other mode, sometimes called ‘one shot’, was originally designed for drum or percussion sounds which are not typically sustained or truncated. Instead, the sample is played once, in its entirety and without looping, when the button or key is pressed. ‘One shot’ sample playback allows a musician to trigger and play an entire sample (such as a cymbal) without having to hold the button or key until the sample has elapsed.

At this point in the development process each Chooser optionally included a time lane which constrained the duration of the Chooser; when the duration elapses the time lane stops any soundable content currently playing. The next round of designs explored selectable stop behaviour, inspired by, but not limited to, the ‘one shot’ setting of samplers and by the behaviour of the merging and synchronisation Petri Nets previously outlined in Section 5.2.3. During the early design

phases of Choosers, the process in which soundable content was stopped as soon as the duration had elapsed was referred to as a ‘rude stop’ or a ‘hard stop’. The new behaviour, which disengaged any loops and then allowed soundable content to complete the current iteration before stopping, was dubbed a ‘polite stop’ or ‘soft stop’. The Choosers shown in Figures 6.23, 6.24 show early experimental designs for stop behaviour. The designs use a draft hard stop icon (a warning triangle) in the status column of the time lane. The stop behaviour is global across all soundable lanes.

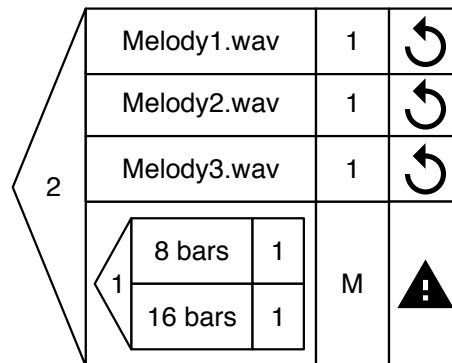


Figure 6.23: An early design to explore stop behaviour; the triangular ‘warning’ icon represents a hard (or ‘rude’) stop, meaning that any soundable content that is playing when the duration is reached will be stopped immediately.

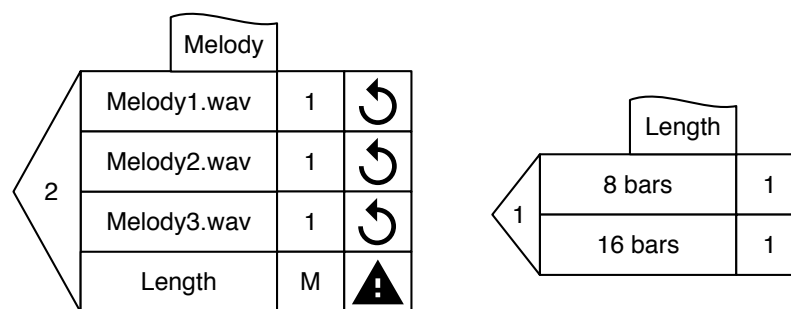


Figure 6.24: This sketch is functionally identical to Figure 6.23, but uses named Choosers.

There were several experiments to test the required functionality and implementation of the time lane. While the previously noted rule requiring the time lane to be selected before soundable lanes seemed to work well (see Section 6.2.2), a number of alternatives were created to explore the design space. Examples included the ability to make any lane the time master, including soundable lanes—this was done by adding a ‘M’ for ‘timing master’ to the weight column. The resulting additional expressivity was useful, especially the ability to use soundable content to control duration.

In line with the design decisions originally formulated in Section 5.2.4, the nondeterministic control of musical duration was considered to be a key function. To explore various ways in which this could be achieved, multiple time lanes in the parent Chooser were enabled, necessitating the removal of the ‘M’ notation so that the weight column could instead be used for the time lanes. A number of sketches were created to explore the emerging Chooser design, starting with the Chooser with one nose cone and two time lanes shown in Figure 6.25. A number of sketches explored visually distinguishing time lanes from soundable lanes, including the designs shown in Figures 6.26, 6.27, 6.28. These sketches make use of alternative icons for the two types of stop—a smiley face for a polite/soft stop, and a warning triangle for a rude/hard stop.

A key development came from a reconsideration of the time lane prioritisation rules; the motivation was to reduce the user’s mental load and to enhance the readability of the notation. This process led to a range of design exercises, briefly outlined below.

Figure 6.25 shows a Chooser with one nose cone and two time lanes. In this example the nose cone is set to 2; the first selection will be one of the two durations (8 or 16 bars, with 8 bars twice as likely to be selected). If 8 bars is selected it will result in a rude/hard stop, whereas if 16 bars is selected it will result in a polite/soft stop. The second lane selection will then select one of the three ‘Melody’ lanes.

While seemingly viable, the design of Figure 6.25 adds complexity to the lane selection rules (‘select one of the time lanes, then use the remaining nose cone value to select soundable lanes’). In order to explore the impact on the user of visually distinguishing lane types, the next three sketches tested visual differentiation of the time lanes via a three-dimensional design (Figure 6.26), additional vertical lines (Figure 6.27), and the use of background colour (Figure 6.28). A fourth sketch introduced a horizontal double line to both visually and functionally distinguish soundable content (above the lines) and time/duration content (below the lines). A Chooser without any time lanes would have the double line at the bottom of the Chooser to show that content could be added. An example of this design is shown in Figure 6.29. While functional, the design did not allow for the granular control offered by later designs with two nose cones, outlined below, and it did not significantly enhance the visual distinction between the two types of lane.

The sketch shown in Figure 6.30 uses a nose cone for the soundable lanes only, with no nose cone for the time lanes. In this rough design a time lane will always be selected and therefore does not need to be included in the nose cone. Instead, the nose cone is used only on the soundable lanes, making the selection of one of the two time lanes implicit. This design provides an interesting development as it uses the nose cone to bracket the soundable lanes, visually separating them from the time lanes. It also more clearly shows the number of soundable lanes to be selected, as the user does not need to subtract one from the value in the nose cone to account for the selection of the time lane.

Figure 6.31 adds a second nose cone to control the time lanes. This is an important change as it led to the eventual formalisation of two types of Chooser, to be outlined in Section 6.2.5. Importantly, the lower nose cone in Figure 6.31 (controlling the time lanes) is limited to either 0 or 1. If this time lane nose cone allowed for two or more durations to be selected, various candidate semantics would be possible. For example, if two durations were selected, the user might assume that different durations might be applied to different soundable lanes. This might have interesting applications in polyrhythmic music. However, given that there could be an arbitrary number of soundable lanes, it is unclear how n time lanes should be assigned in a clear way to m soundable lanes in the general case. Such semantics seem likely to cause confusion. Consequently, it was decided that at most one time lane can be chosen and its duration applied to all soundable lanes. Still, polyrhythmic music should be made possible, and indeed this can readily be achieved via nesting (as will be seen in Section 6.2.8, with an example of two different simultaneous durations shown in Section 10.4.1 of Chapter 10).

In the design shown in Figure 6.31, soundable and time/duration content now have independent nose cones, meaning that various combinations are made possible, allowing various musically useful distinctions to be expressed as follows:

Soundable Chooser nose cone > 0; Time Chooser nose cone = 1 Soundable output with moderated duration.

Soundable Chooser nose cone > 0; Time Chooser nose cone = 0 Soundable output with infinite duration; the Soundable Chooser runs as though there is no Time Chooser.

Soundable Chooser nose cone = 0; Time Chooser nose cone = 1 Silent playback of a given duration—a musical rest. The Soundable Chooser is set to ‘select none’.

Both Soundable and Time Chooser nose cones = 0 Chooser is skipped.

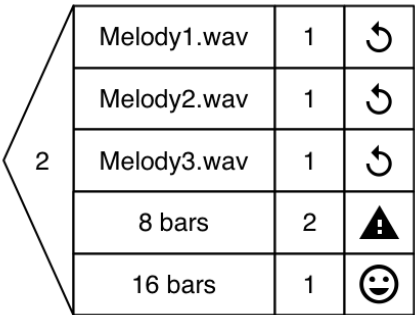


Figure 6.25: A Chooser with one nose cone and two time lanes.

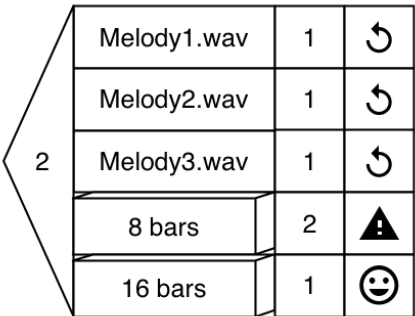


Figure 6.26: An early sketch of a Chooser with visually differentiated time lanes using a three-dimensional lane design and a shared nose cone.

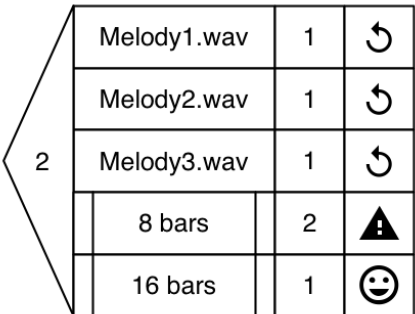


Figure 6.27: A sketch of a Chooser with visually differentiated time lanes. This example uses additional vertical lines in the time lanes, and maintains a shared nose cone for all lanes.

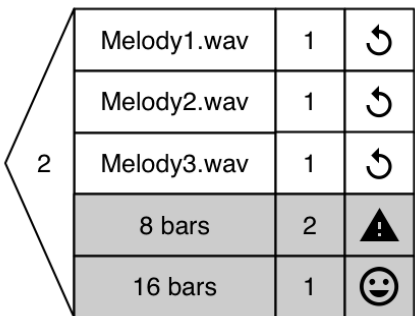
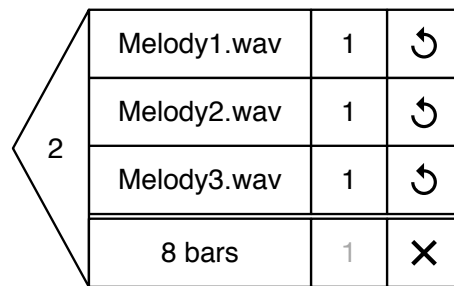
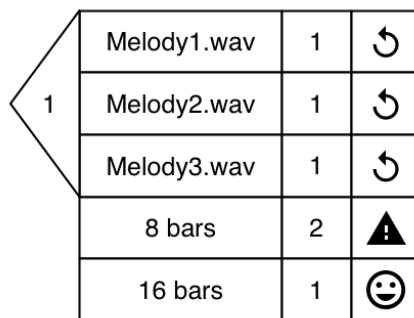


Figure 6.28: A sketch of a Chooser with visually differentiated time lanes. As with the previous two examples, this design maintains a shared nose cone for all lanes.



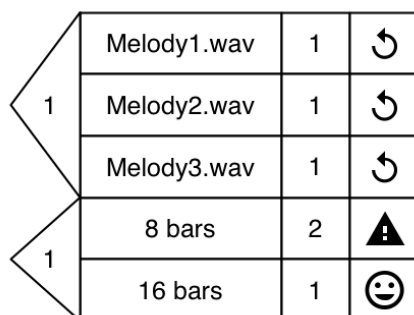
Melody1.wav	1	↺
Melody2.wav	1	↺
Melody3.wav	1	↺
8 bars	1	✕

Figure 6.29: Testing a double line between the soundable (upper) and time (bottom) lanes in a Chooser. This design uses a shared nose cone. Note the early test of an alternative icon for a hard stop (✕).



Melody1.wav	1	↺
Melody2.wav	1	↺
Melody3.wav	1	↺
8 bars	2	⚠
16 bars	1	😊

Figure 6.30: A Chooser with a separate nose cone for the soundable lanes. This example has no nose cone for the time lanes, with the assumption that one time lane will be selected. This design was rejected as it removes the ability to remove the time lanes from selection.



Melody1.wav	1	↺
Melody2.wav	1	↺
Melody3.wav	1	↺
8 bars	2	⚠
16 bars	1	😊

Figure 6.31: A sketch of a Chooser with differentiated soundable and time lanes via the use of two nose cones. This design distinguishes lane types while allowing the nose cone values to be independently set. The final design, shown in Figure 6.32, uses a downwards-sloping nose cone for the Soundable Chooser and an upwards-sloping nose cone for the Time Chooser.

6.2.5 Differentiating Choosers: Soundable and Time Choosers

The double nose cone Chooser, shown in Figure 6.31, was adopted as part of the fundamental design of Choosers.

At this point the design involved two distinct elements with differing content and behaviours: a *Soundable Chooser*, containing only soundable material, and a *Time Chooser* which contained time/duration material. The nose cones were changed in shape to a downward-sloping triangle for Soundable Choosers and an upwards-sloping triangle for *Time Choosers*. The change in design made each Chooser type visually distinct, as well as hinting at their combinatorial usage. When used together, as shown in Figure 6.32, the combination is referred to as a *Full Chooser*, or just a Chooser for short.

2	drums.wav	A	↺
	bass.wav	1	↺
	marimba.wav	1	↺
1	8 bars	1	✕
	16 bars	1	➤

Figure 6.32: A Full Chooser, consisting of a Soundable Chooser on the top and a Time Chooser on the bottom.

Given the new distinction between the two types of Chooser there was an opportunity to introduce the use of soundable content to constrain duration in the following ways. A soundable file (e.g. an audio sample) can be used in a lane in a Time Chooser and, if selected, the file's duration will be used by the Time Chooser. The Full Chooser shown in Figure 6.33 contains a Soundable Chooser which will select and then loop one of two soundable files (Melody1.wav or Melody2.wav). The Time Chooser contains a third soundable file, Melody3.wav, set to a hard stop. When run, the Full Chooser will select one of the two soundable files for looping playback, and playback will be stopped as soon as the duration of Melody3.wav has elapsed. Melody 3.wav will not be audible as it is in the Time Chooser. If the user wants the sample to be audible *and* to control duration the sample will need to be added to both the Soundable and the Time Choosers, as seen in Figure 6.34¹.

1	Melody1.wav	1	↺
	Melody2.wav	1	↺
1	Melody3.wav	1	✕

Figure 6.33: A Full Chooser with an audio sample in a Time Chooser lane. The duration of the sample will set the duration of the Full Chooser; as the lane is set to a hard stop the currently playing sample (either Melody1.wav or Melody2.wav) will stop as soon as the duration has elapsed. Melody3.wav will not be played as it is in a Time Chooser lane.

¹This design will be revisited and revised in Section 8.1.4 in response to the findings from the first user study outlined in Chapter 7.

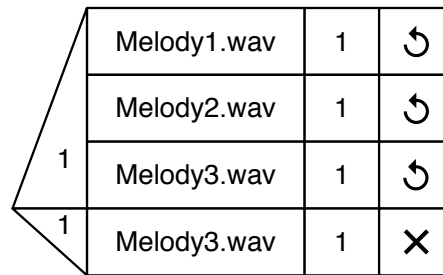


Figure 6.34: Using the current semantics, if the user wants the Full Chooser to use Melody3 .wav for both duration and soundable output, it must appear both in the Time Chooser and in a selectable lane in the Soundable Chooser.

6.2.6 External input

Choosers were designed to operate as a standalone tool, but optional interaction with other software or hardware is also desirable. Open Sound Control (Wright, 2005), a network communication protocol designed for interactive computer music, was identified as a pragmatic candidate protocol for such interaction. OSC messages could be integrated in Choosers for both incoming and outgoing interactions. For example, choosing a lane could cause a stored OSC message to be sent to external networked software or hardware. Equally, OSC messages could act as inputs, as illustrated in Figure 6.35. In this way, Choosers could be integrated with a wide range of systems (e.g. accepting triggers from a video game engine) or hardware (e.g. facilitating live performance).

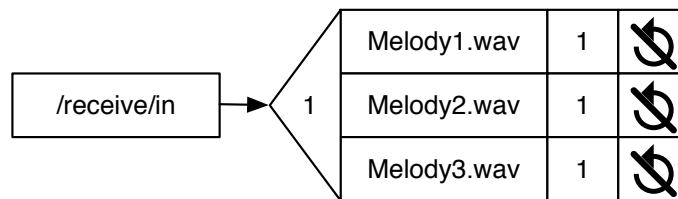


Figure 6.35: A sequence with an OSC message connected to a Soundable Chooser. The OSC message acts as a ‘start’ message for the Chooser, allowing for software or hardware control of playback.

6.2.7 Variables

Nesting Choosers, and enabling references to named Choosers from inside either Choosers or boxes, clearly adds to the expressive power of Choosers. However, expressive power could be enhanced by allowing variables to refer to other elements of the language—and, in keeping with the ethos of Choosers, having this happen nondeterministically. In order to provide a consistent set of tools for the user for such a feature, a third type of Chooser—a Variable Chooser—was developed. Variable Choosers allow for the selection of either numbers or status values (e.g. loop/non-loop, hard/soft stop). As they are used to select a single variable value, the nose cone of Variable Choosers is constrained to either one or zero (no selection). By default, variables are run afresh each time they are called (e.g. when a Chooser which references the Variable Chooser is run), which may result in a different selection each time.

A sketch testing the functionality of Variable Choosers is shown in Figure 6.36. In this example there is a Full Chooser on the right of the figure, consisting of a Soundable Chooser with three equally-weighted non-looping audio samples and a Time Chooser with three durations, all set to hard stop. The ‘2 bar’ duration is twice as likely as the other durations to be selected. A Variable Chooser, named ‘Var’, will select either 0 or 1 with equal probability each time it is called. The Variable Chooser ‘Var’ is referenced from both of the nose cones in the Full Chooser and, as variables

are run afresh each time they are called, the nose cones may receive different values. If, when run, the Variable Chooser selects a 1 for both Soundable and Time Chooser nose cones then one sample will be selected for playback, and the duration of the playback will be constrained by the selected duration. If the Soundable Chooser nose cone is set to 0 and the Time Chooser nose cone is set to 1 then no sample is selected, resulting in a rest for the chosen duration. If the Soundable Chooser nose cone is set to 1 and the Time Chooser nose cone is set to 0 then a sample is chosen and will play once, unconstrained and in its entirety. If both nose cones are set to 0 then the entire Chooser will be skipped.

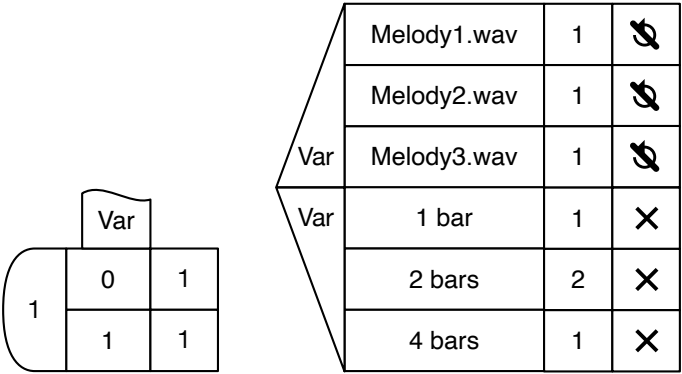


Figure 6.36: A sketch used to explore the functionality of Variable Choosers. The Variable Chooser named ‘Var’ (left) will select either 0 or 1 with equal probability when it is called. ‘Var’ is referenced from the nose cones in both the Soundable and Time Choosers combined into a Full Chooser on the right. Note that the default behaviour of a Variable Chooser is to run afresh each time it is called, leading to four possible outcomes (1 and 1; 1 and 0; 0 and 1; 0 and 0).

Nondeterministic assignment to variables is a flexible feature that is intended to have significant musical uses. However, there are some musical situations in which unfettered nondeterministic choice may be unnecessary or counterproductive. For example, having made a sequence of non-deterministic choices, it may be musically useful to subsequently exactly revisit those choices—after all, as argued earlier, repetition is a well-nigh universal and desirable feature of music. To this end, some sort of journaling feature for nondeterministic variable choices seems desirable. In considering this issue I was influenced by the simpler distinction between copies and aliases of MIDI and audio regions in *Logic Pro* (Apple Inc., 2021), *Cubase* (Steinberg Media Technologies GmbH, 2021), and other DAWs. Both copies and aliases can be moved as the user requires but, while the copy may be modified independently, the alias remains linked to the original object in the sense that any changes made to the original will be reflected in the alias. For example, if the hi-hat pattern is changed in the original MIDI sequence, all aliases will update to reflect the change.

The standard DAW copy behaviour is to create an independent copy, with aliases accessible as a secondary option (e.g. via a menu or by using a modifier key). Given that Choosers were developed to bring nondeterministic musical moves to the fore, the default behaviour of nested Choosers is for a new choice to be made on each run. However, there are some instances when the user may want to reuse the same result from a variable; for example, it may be desirable to select and subsequently reuse a rhythmic or melodic pattern. The design for variable reuse was outlined in Section 6.1.7.

6.2.8 Nesting vs. referencing Choosers

In order to accommodate for differing use cases, as well as for user preference, Choosers allow for both visual subpatch-style nesting and for text-style naming and referencing. In a simple Chooser, nesting one Chooser inside a lane of a parent Chooser may be appropriate; using the language of CDN from Chapter 4, placing the nested Chooser inside a lane enhances closeness of mapping, reduces hard mental operations, increases role expressivity, and reduces hidden dependencies. The

alternative system is to name a Chooser via its fin and to reference that Chooser from the lane of a parent Chooser, or from inside a box. This option offers a different set of trade-offs which may make it better suited to more complex Choosers and arrangements. The ability to create a high-level overview of musical structure and populating each section via references to Choosers offers a close match to the user's conceptual model. More complex examples may make more than one reference to a single Chooser, reducing viscosity and enhancing visibility. Additionally, where a named Chooser is referenced from multiple locations, changes to the named Chooser will be reflected across all instances. Such behaviour is familiar to programmers, and DAW users will recognise this functionality as similar to aliases (see Section 6.2.7).

6.2.9 Comparing Choosers with Petri Nets

Petri Nets are an expressive and powerful notation which was a key influence in the design of Choosers (see Section 5.2.2). Given that Choosers have been designed to maintain musically meaningful expressivity it is instructive to consider trade-offs in readability and viscosity between the two designs. This section will briefly show how the important structural features of music from Baratè (2008) can be accomplished in Choosers. Music Petri Nets are not compared to Full Choosers, which include duration moderation. Music Petri Nets assume that each music object will play in its entirety before it is released, whereas Choosers were designed to include a first-class duration moderation mechanism via Time Choosers.

Figure 6.37 shows sequencing in both Choosers and Petri Nets. Figure 6.38 shows a single process followed by two simultaneous processes in parallel, referred to as a 'split' by Baratè (2008). The rules of Petri Nets state that a transition is enabled when the incoming places contain tokens that are equal in number to the arc weight going from the place to the transition. When the transition fires it removes the tokens that enabled it from the input places, and distributes tokens to the output places according to the arc weights. Figure 6.38 shows that the transition is enabled by a single token in a single place (MO1). Once fired, the transition will remove the incoming token from MO1 and will add a token to both MO2 and MO3, resulting in their simultaneous playback. In the case of this very simple relationship the Choosers instance is more visually cluttered, reducing readability but surfacing more options and allowing for quick parameter changes via the nose cone, lane weights, lane loop settings.

Nondeterministic choice between two alternatives, called 'alternative choice' by Baratè (2008), is shown in Figure 6.39. It is instructive to walk through the two diagrams informally as a rough gauge of the cognitive load associated with the two notations in this case. In the case of the Petri Net diagram the reader must first identify that the leftmost place currently contains a token, thus enabling both of the transitions in the Petri Net. The reader will need to know that the lack of arc weights means that the arcs are of equal weight, meaning that an equal nondeterministic selection between the two transitions will be made; when one of the two transitions fires the incoming token will be removed, thereby disarming the other transition. Upon the firing of one of the transitions, a token will be passed to either object MO1 or MO2, resulting in the playback of either music object. By contrast, in the case of the Choosers diagram, the reader must first interpret the orientation of the nose cone to identify that this is a Soundable Chooser as this has different semantics from a Time Chooser. Then they must interpret the number in the nose cone to mean that one of the two samples will be selected nondeterministically and played; the weight column shows the likelihood of selection and the status column shows that neither sample will loop. Music Petri Nets allow for arcs to be weighted in order to control the likelihood of selection. Figure 6.40 shows an example of arc weights, alongside the control of probabilistic weights in Choosers via lane weights.

Figure 6.41 shows an extension of Figure 6.40 with infinite looping enabled on three of the four music objects. An example of arithmetic $\times 2$ notation in Choosers, compared with a PN repeat, is shown in Figure 6.42. Figure 6.43 shows an example in which one lane out of three will be selected and played, and then the Chooser will reselect and run afresh. Arithmetic repeat notation, as used in the Chooser in Figure 6.43, means 'play this object twice'.

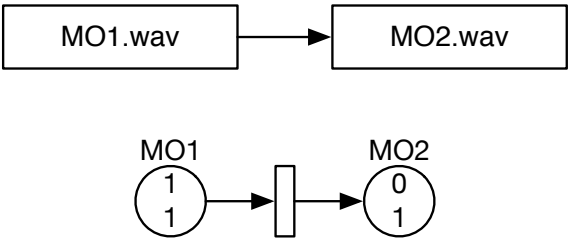


Figure 6.37: A sequence in Choosers (top) and in Music PNs (bottom).

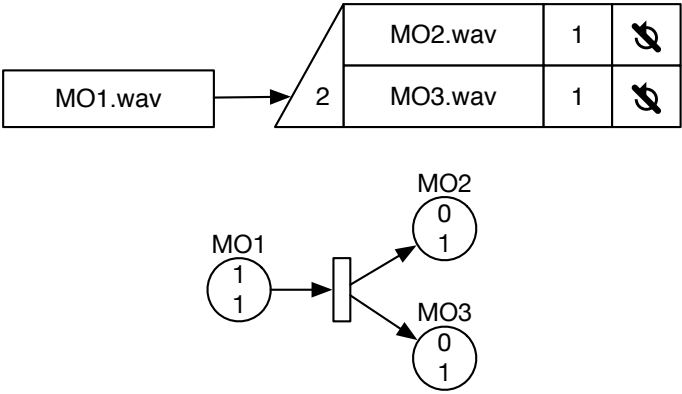


Figure 6.38: Splitting from one to two simultaneous music objects in Choosers (top) and Music PNs (bottom).

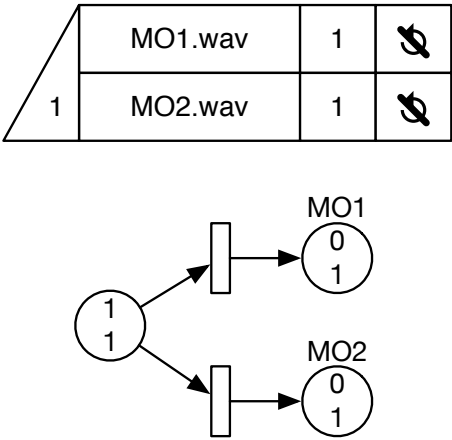


Figure 6.39: Alternative choice in Choosers (top) and Music PNs (bottom).

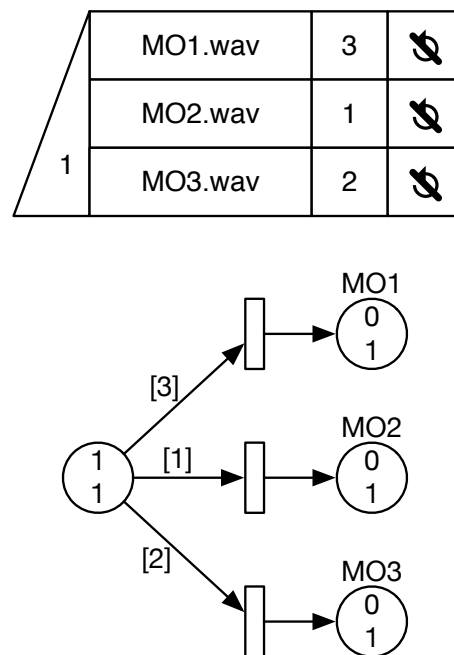


Figure 6.40: Probabilistic weight in Choosers via lane weights (top) and in Music PNs via the probabilistic weight of arcs (bottom).

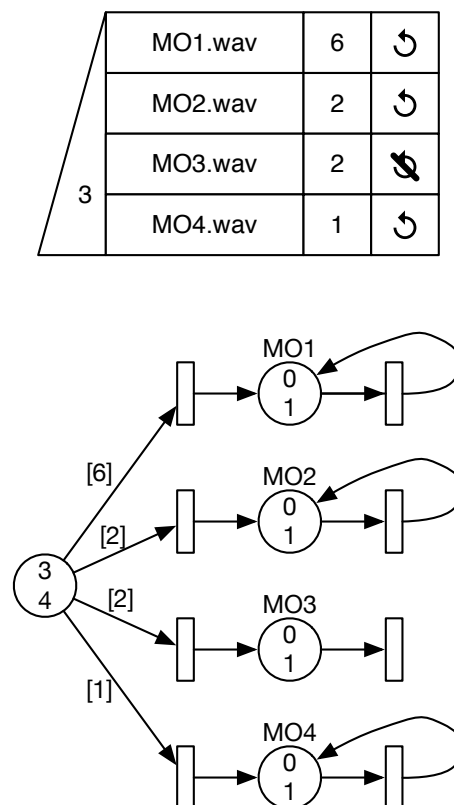


Figure 6.41: An extension of Figure 6.40 with infinite looping enabled on three of the four music objects.

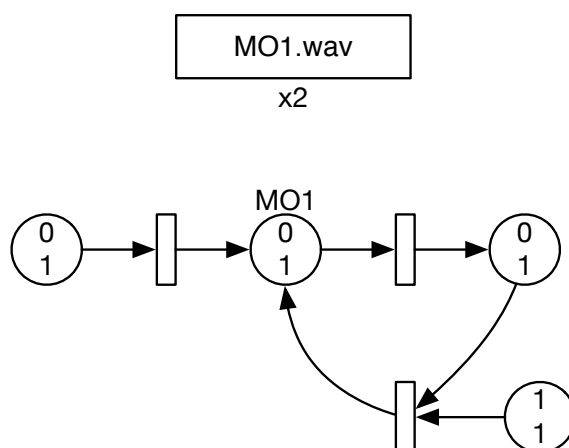


Figure 6.42: Arithmetic 'x2' notation in Choosers (top), and a PN repeat (bottom).

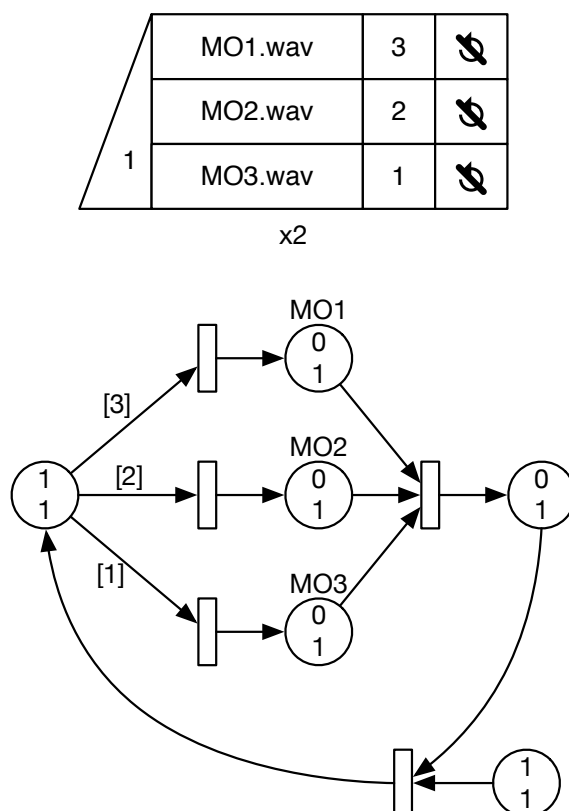


Figure 6.43: Arithmetic repeat notation, as shown here, means 'play this object twice'. In this case one lane will be selected and played, and then the Chooser will reselect and run again.

The comparison of Choosers with Music PNs shows that Choosers are more visually complex when used to represent very simple musical content, with a resulting trade-off that musically useful parameters (e.g. control over the number of synchronous elements, nondeterministic choice including weights, looping) have been surfaced. However, when progressing to slightly more complex musical structures, Choosers maintain the visibility of musically meaningful parameters for change and auditioning. When representing the same musical structures, Music PNs become more viscous and harder to read and manipulate, placing a higher cognitive load on the user.

6.2.10 Reflections on the design of Choosers

Before leaving this chapter, let us take a moment to reflect on the design process so far, taking into account lessons learned in the last three chapters. The design of Choosers makes use of a tabular framework or template to remove the burden of learning syntax from the user. The design does not rely on domain-specific metaphors, such as common music notation or mixing desks. Choosers makes use of a constrained number of consistently-designed widgets, with advanced functionality available via combinatorial usage. The most musically relevant parameters (for the purposes of this thesis) have been moved to the surface, with auditioning readily available to users to test the results of changes. Referencing and sequencing allow for top-down and bottom-up construction (Section 6.1.5).

In order to more fully detail the decisions outlined above, user needs and specific Chooser features drawn from the last three chapters have been mapped to the Chooser design features presented in this chapter. The three tables below draw on the findings of the previous two chapters respectively in turn: the CDN review (Table 6.1), the design exercises using Direct Combination (Table 6.2), and Petri Nets (Table 6.3). The full CDN review is presented in Chapter 4 with findings and heuristics listed in Section 5.1. The Direct Combination design exercises are outlined in Section 5.2.1, and the Petri Net tests are presented in Section 5.2.2.

Table 6.1: Mapping user requirements, findings from the CDN review (Chapter 4), and the resulting feature in Choosers.

User need	Findings from the CDN review	Chooser feature
The target user does not know how to program.	Providing the user with a template to fill in removes the need to learn textual syntax.	Choosers present a visual object to the user to fill in with soundable content and durations.
Minimise knowledge required, specifically domain-specific knowledge—music notation/theory or music production.	Avoid domain-specific metaphor for novice users where possible (Section 5.1).	Choosers do not use manuscript (music domain) or mixing desk/patching metaphors (music production domain).
User does not want to have to learn how to use multiple widgets.	A parsimonious approach to design will reduce the number of widgets, and a consistent design will allow users to apply learning across the interface.	Choosers use a limited number of widgets with consistent design principles (Section 6.2.1; Section 6.2.5; Section 6.1.2). Advanced functionality is achieved via combination (Section 6.2.5) rather than additional widgets.

User need	Findings from the CDN review	Chooser feature
User needs to be able to read the notation without opening multiple windows.	Maximise visibility and juxtaposability in the interface while reducing hidden dependencies.	Surface the key parameters to the top level so they are always visible. Progressive disclosure used to reduce visual complexity when parameters are not useful.
User should be able to build a piece of music using either top-down or bottom-up construction.	Premature commitment and viscosity should be reduced to allow for structure changes throughout the compositional process.	Naming and referencing Choosers (Section 6.1.4) allows for the separation of sequence (macro) and content (micro). The user can assemble and edit musical structures at either scale (Section 6.1.5).
User needs to be able to quickly make and audition changes which result in musically meaningful changes.	Identify the key parameters required for common musical moves, and surface them with good visibility. Lower viscosity by allowing arrangement changes throughout the compositional process.	Design allows for simple changes with significant impact, e.g. disabling via zero nose cone setting (Section 6.2.2) or rearranging a sequence (Section 6.1.5).
Maximise the chances of musical success for novice algorithmic composers.	Identify the most common musical moves and map them to default/surface actions.	Choosers put common musical moves on the surface; the default values will generally produce sensible and musical output.
Allow advanced users to create complex compositions.	Allow for less common musically meaningful actions via nesting, referencing, and combination.	Advanced functionality is accessed via nesting (Section 6.1.6); naming and referencing (Section 6.1.4); variables (Section 6.1.7); and use of the @ notation to pin values.

Table 6.2: Mapping user requirements, findings from the DC design exercises (Section 5.2.1), and the resulting feature in Choosers.

User need	Findings from the DC design exercises	Chooser feature
User does not want to have to learn how to use multiple widgets.	Parsimony in showing a limited number of interface elements; design constraint.	Progressive disclosure; show musically relevant parameters—loop, weight, stop (Sections 6.2.1, 6.2.2, 6.2.5).
User needs to be able to read the notation without opening multiple windows.	Tabular interface showing relevant options; semantic consistency; visibility of interdependencies.	Consistent interface elements; parameters made visible by placing them on the top level of the interface; interdependencies shown via Chooser nose cone shape.

User need	Findings from the DC design exercises	Chooser feature
Simple functionality should be easy, and advanced musical moves should be possible.	Enhance functionality via combinatorial power rather than spawning widgets.	Advanced functionality available via nesting (Section 6.1.6), naming and referencing Choosers, and via variables (Section 6.1.7).
User should be able to build a piece of music using either top-down or bottom-up construction.	Compositional flexibility via separation of structure and content.	Allow for arrangements to be made using boxes which reference Choosers—content therefore separate from the macro structure.
The system should not confuse the user with unnecessary options. The user should be able to reuse a defined musical object.	Options are constrained to those which are useful given the current selections. Allow for a nonlinear or linear musical event to be used more than once in a piece of music via naming and referencing.	Progressive disclosure used to surface controls when they are required. Once named using the optional fin, a Chooser can be referenced multiple times, at different times, and from different parts of the piece.
User can compose by assembling small musical fragments, or by defining structure before populating sections.	Allow for easy creation and subsequent manipulation of musical structure, including top-down or bottom-up construction.	Naming and referencing Choosers and boxes (Section 6.1.6) allows for the separation of macro arrangement and object content, lowering viscosity.

Table 6.3: Mapping user requirements, findings from the PN design exercises (Section 5.2.2), and the resulting feature in Choosers.

User need	Findings from the PN design exercises	Chooser feature
Algorithmic music is new to most users, so musical success should be within easy reach.	Encourage simple and musically meaningful moves. Experimentation is very important in understanding the possibilities of algorithmic music.	Surface musically meaningful and impactful changes to the top level—nose cone and weights (Section 6.2.2); sequencing and referencing (Section 6.1.5)—and allow for quick auditioning of changes.
The notation needs to be suited to music work.	A visual notation is well suited to music; structure is often shown graphically. PNs are suitably expressive but are overly complex for the target user (Section 5.2.2).	The Chooser offers a music-focussed abstraction of the key functions. Sequencing uses flow chart-style arrows (Section 6.1.5); Choosers offer a tabular template for completion, reducing syntax overhead for the user.

User need	Findings from the PN design exercises	Chooser feature
Nondeterministic choices need to be controlled without complex syntax.	Focus on the number of selections and relative weights for each potential selection.	Choice and multi-choice shown via stacked lanes, nose cone, and lane weights.
Duration control needs to be accessible and flexible.	The user should be able to impose duration control in two ways; to stop music events immediately (Section 5.2.3.5) or allow them to finish (Section 5.2.3.3).	Time Choosers to moderate duration with the choice of a hard (immediate) or soft (allow to complete) stop.

6.3 Conclusions

This chapter has presented the initial design of Choosers, reflected on the design in the light of user needs, and recapped the findings from the analyses of the last two chapters. It has also outlined the design process that followed the findings from the Cognitive Dimensions of Notations review from Chapter 4 and the design exercises which explored both Direct Combination (Section 5.2.1) and Petri Nets (Section 5.2.2). These reviews found that the design should not rely on existing domain knowledge, such as common music notation or music production equipment; it should minimise the learning required to make use of the software; it should surface the most appropriate musical parameters; it should allow the user to make changes and audition the results; it should allow for a variety of compositional approaches; and it should minimise the syntactic knowledge required (e.g via the use of tabular templates). By combining elements of both DC and PNs, music-focused abstractions have been developed which may reduce expressivity for arbitrary processes with preconditions but which aim to prioritise and surface a subset of musically-meaningful moves (see Section 6.2.1). The key abstraction outlined in this chapter, the *Chooser*, allows for sequence, concurrency, alternatives, nesting and referencing via naming, and repeats (including ‘repeat until x has finished’ logic). The next chapter, Chapter 7, will present the findings from the first user study.

Chapter 7

First user study

This chapter presents the design of, and findings from, a participant programming walk-through evaluation of Choosers v1. While participants were able to create nondeterministic music, various design issues were identified which led to the proposal of candidate refinements outlined in this chapter and more fully explored in the following chapter.

Previous chapters have described the design process of a nondeterministic music composition notation and graphical programming language which is suitable for novice users, leading to the first complete design, Choosers v1, outlined in Chapter 6. This chapter will present a programming walkthrough evaluation (Bell et al., 1992, see Section 3.4) of Choosers v1, carried out with six pairs of undergraduate Music Technology students. The purpose of this evaluation was to:

- Test the ability of self-taught music producers without programming skills to use Choosers v1 in order to carry out a range of rudimentary nondeterministic composition tasks;
- Identify user experience issues in the current design;
- Identify tensions and trade-offs in the interaction design of the system.

In the evaluation, pairs of participants were introduced to each element of the graphical programming language via short tutorial videos. Participants were given a range of practical tasks to complete on paper or a whiteboard. The evaluation facilitator (i.e. the thesis author) played a Wizard of Oz role, rapidly translating participants' graphical solutions into runnable code that was fed into a non-graphical prototype version of Choosers v1 so that participants could hear the musical results of their work.

7.1 Problem setting

Various music programming languages are capable of algorithmic composition, although they generally require significant programming skills (Bullock et al., 2011) thereby making them inaccessible to many users. Previously, in Chapter 4, the Cognitive Dimensions of Notations framework (Green and Petre, 1996) was used to review the usability of a representative selection of software capable of algorithmic music composition (see Chapter 4 for the detailed results). The findings of the review included the following. First, most existing software requires the user to have a considerable understanding of a number of different types of syntactic constructs in either graphical (e.g. *Max*, *Pure Data*) or text-oriented (e.g. *SuperCollider*, *ChuckK*, *Csound*) programming languages: such knowledge requires a significant learning overhead. Second, users are often required to have an understanding of musical notation and/or music production equipment such as mixing desks and patchbays. Third, several programs imposed working practices unconducive to compositional

processes. Fourth, in some cases the user was unable to define, and 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 to navigate.

7.2 Method

The methodology for this first user study, presented below, adheres to the walkthrough protocol previously presented in Section 3.5.

7.2.1 Participants

Six pairs of undergraduate Music Technology students participated in this study. Each pair was introduced to the graphical programming language via a series of short tutorial videos (Bellingham, 2020c). Users were given a range of practical tasks to complete on paper (Figure 7.2) or on a whiteboard (Figure 7.3), and their outputs were played by the facilitator using a set of *SuperCollider* (McCartney, 2002) classes written to implement the musical abstractions behind the system. In order for the facilitator to provide near-instant playback (using the Wizard of Oz technique), a series of templates had been previously prepared and were edited during the sessions to match the participant sketches. An example of the *SuperCollider* code used by the facilitator which is then processed by the rest of the code, is shown in Figure 7.1; this example shows a Chooser created using paper templates. The user tests were videoed and transcribed to assist in the analysis presented here.

All participants were asked to complete a short questionnaire before taking part in the user tests. Of the twelve participants, all had played a musical instrument at some point, and ten had some formal training via a GCSE Music qualification. Six participants did not read any music notation: of those that did, most could read common music notation as well as chord notation. None of the participants could read and simultaneously play either common music notation or chord charts. All participants were familiar with DAWs, with *Logic Pro* (Apple Inc., 2021) mentioned by all participants. Other DAWs mentioned included *Pro Tools* (6 mentions), *Cubase* (2 mentions), *FL Studio* (5 mentions), *Reason* (1 mention), and *Ableton Live* (1 mention). *Pure Data* (Puckette, 1997) a visual audio programming language, was mentioned by two participants. Ten participants had experience using hardware such as drum pads or control surfaces to control music software. The participants were not habitual performers; seven of the twelve participants do not perform with or for others. Of those that do, three perform in church, and four occasionally play with friends in private. Five of the twelve participants felt they had some experience in computer programming. Of these, two considered writing for the web (HTML and CSS) to be programming. Given that HTML/CSS development is focussed on markup and layout it can be seen that more than two thirds of participants do not have experience of writing algorithms in a programming language. Only one participant listed the use of both *Pure Data* and *SuperCollider*. Eight of the twelve participants did not know what algorithmic music was at the start of the user tests; the remaining four participants felt that they knew what algorithmic music was, but had not created any.

Each pair of participants were asked to take part in eight scenarios, as detailed in Section 7.3. The two participants in each pair were free to discuss the work and to ask for clarification with the administrator of the test. Users were asked to act as active participants in the research, and to help in categorising any issues that were raised. The participants were asked to use the following categorisations, taken from the programming walkthrough method (Bell et al., 1991, 1992) and previously outlined in Section 3.5.1:

- Questions (e.g. why does the loop do that?);
- Problems (e.g. I don't understand what these lanes are for);
- Suggestions (e.g. maybe the cone should be a different shape);
- Other observations (e.g. I like the fins).

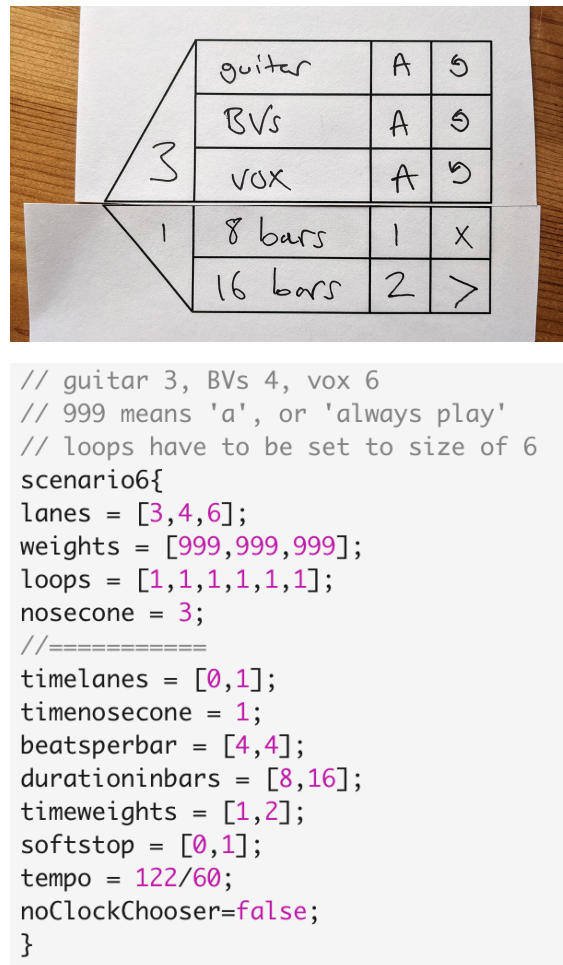


Figure 7.1: An example of a participant sketch made using paper templates and the corresponding *SuperCollider* template code, edited by the facilitator during the sessions to enable participants to hear the results of their work.

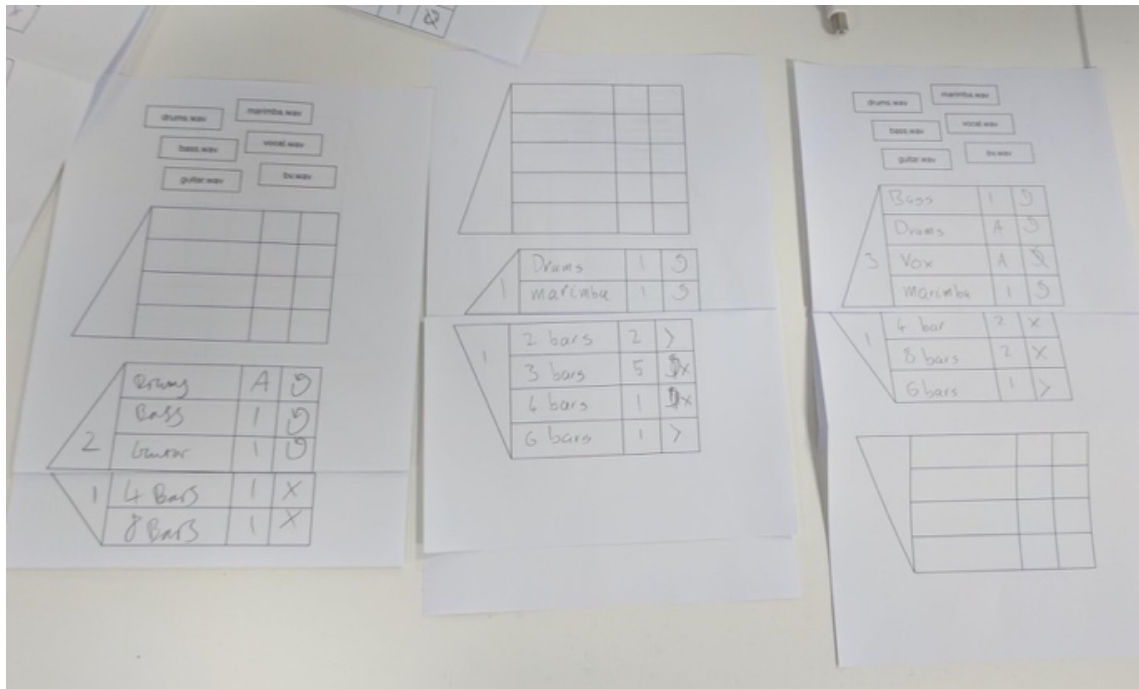


Figure 7.2: Participant Chooser work using paper templates.

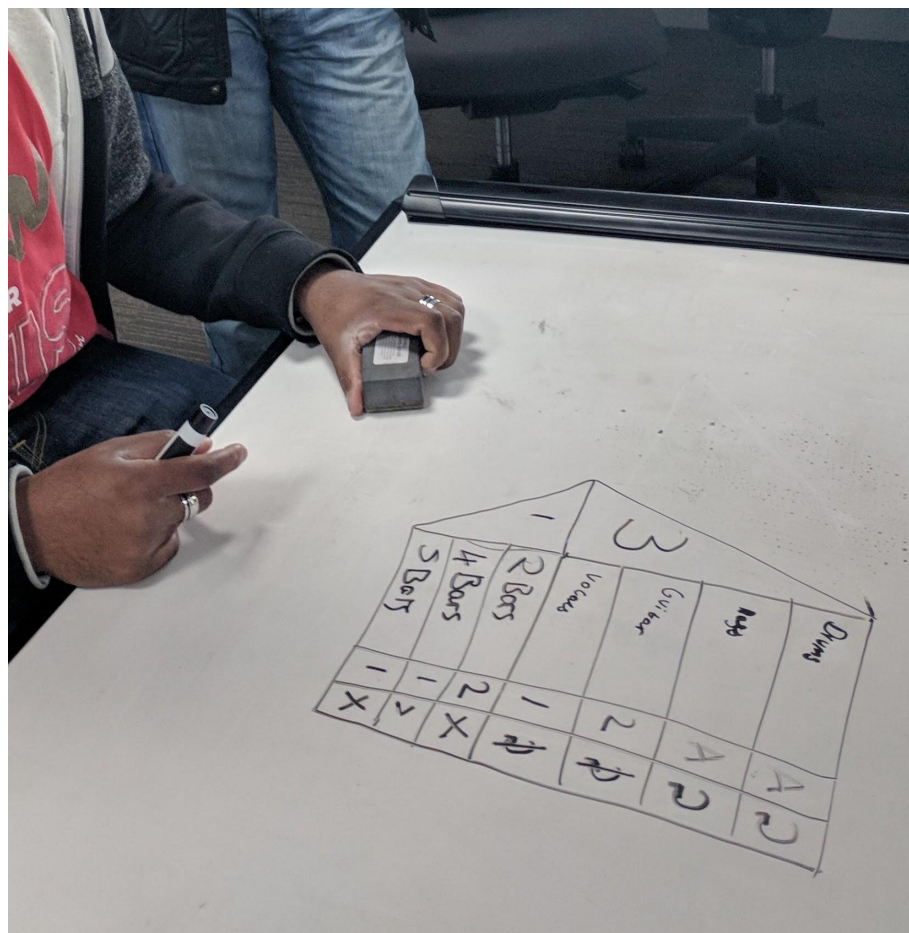


Figure 7.3: Participant Chooser work using a whiteboard.

The results from the eight scenarios issued as part of the user tests will now be presented. The images used, verbal instructions issued, and playback of tutorial videos, are shown below. In each case, italicised text is used to provide context, followed by the instructions given directly to the participants. Each scenario will be followed by a brief overview of the results, and a more detailed reflection on the design issues will be presented in Section 7.5.

7.3 Walkthrough scenarios

7.3.1 Scenario 1: Chooser basics

Participants were shown a tutorial video (Bellingham, 2020c) which introduced lanes, nose cone, weighting including always play, and loop/non-loop functionality. They were then shown the Soundable Chooser shown in Figure 7.5, and were asked the following questions:

- 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?

Findings from scenario 1

This scenario prompted a number of clarifying questions from participants, all of which in this case could be answered simply by playing the video for a second time. All participants were able to complete this scenario without error.

7.3.2 Scenario 2: making a simple Chooser

Participants were issued with the paper templates shown in Figure 7.6 and were given the following tasks:

- Using the provided template (Figure 7.6), 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.

Findings from scenario 2

The Soundable Chooser design was understood quickly by most participants. The layout of the Chooser's vertically stacked lanes created some interesting assumptions from participants in two groups. The participants appeared to apply existing knowledge of horizontally-scrolling DAWs, such as *Pro Tools* or *Logic Pro*, in which vertically stacked lanes play concurrently. However, in the Chooser design, such lanes contain material which may or may not be selected for playback and concurrency cannot be assumed without also considering the number in the nose cone, the lane weight, and the 'always' feature described below. This assumption was made twice in the user tests, and both instances prompted clarifying discussions which resolved the issue.

The use of 'A' ('always play') in the weight column of Soundable Chooser lanes was understood quickly by all participants. Users were asked to prevent the guitar from being available for selection, and all groups chose to change the guitar's weight to zero. This answer was correct, but was not the only way to achieve the desired result. One participant correctly suggested that the bass could also be set to A, meaning that the guitar would not be selected given the restriction imposed by the number in the nose cone.

Scenario 1

1	drums.wav	1	↺
	bass.wav	1	↺

Each pair of participants was shown a video which introduced lanes, nose cone, weighting including always play, and loop/non-loop functionality.

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

1	2 bars	1	✕
	4 bars	2	✕

Each pair of participants was shown a video which introduced Time Choosers, multiple lanes, and nose cone restrictions.

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

Each pair of participants was shown a video which introduced Full Choosers, duration control via the Time Chooser, and the hard and soft stop mechanism.

			✕
--	--	--	---

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:

2	drums.wav	A	↺
	bass.wav	1	↺
	guitar.wav	1	↺
1	4 bars	1	✕

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.

3	drums.wav	3	↺
	bass.wav	2	↺
	marimba.wav	1	↺
1	8 bars	1	✕

- 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.

3	guitar.wav	A	↺
	BVs.wav	A	↺
	vocals.wav	A	↺
1	8 bars	1	✕
	16 bars	2	>

- 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

Each pair of participants was shown a video which introduced the sequence mechanism.

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.

Final questions

At the end of the user test, each pair was asked the following three questions:

Can you see anything this would be useful for?

Can you see any ways in which this is similar to other tools you have used?

Is there anything that is made easier by this system? Anything which was not possible made possible/hard and made easier?

Figure 7.4: The eight scenarios used in the first user study. In each case, italicised text is used to provide context (e.g. the use of tutorial videos). The main text shows the instructions given directly to the participants.

1	drums.wav	1	🔇
	bass.wav	1	🔇

Figure 7.5: Soundable Chooser used in scenario 1 of the first user study.

Figure 7.6: Blank paper template used in scenario 2 of the first user study.

7.3.3 Scenario 3: Time Choosers

Participants were shown a video (Bellingham, 2020c) which introduced Time Choosers, multiple lanes, and nose cone restrictions. They were then shown the Time Chooser in Figure 7.7 and asked the following questions:

1	2 bars	1	✗
	4 bars	2	✗

Figure 7.7: Time Chooser used in scenario 3 of the first user study.

- Describe what will happen when this Chooser is run by itself.
- The nose cone 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?

Findings from scenario 3

Two groups found multiple vertically aligned lanes confusing, as in the previous scenario. One participant assumed that both durations would play together, presumably due to the behaviour learned through the use of DAWs. Another user thought that the first duration would play, followed by the second duration. In both cases the participants were shown the relevant section of the tutorial video again, after which they had no further problems with the Time Chooser mechanism.

7.3.4 Scenario 4: Full Choosers

Participants were shown a video (Bellingham, 2020c) which introduced Full Choosers, duration control via the Time Chooser, and the hard and soft stop mechanism. The participants were then asked to undertake the following task using the template shown in Figure 7.8; for the reader's benefit, an example expected result of scenario 4 is shown in Figure 7.9.

- Keeping the Soundable Chooser you made for scenario 2, create a single-lane Time Chooser using the supplied template (Figure 7.8).
- 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?

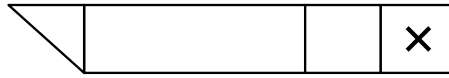


Figure 7.8: Paper template used in scenario 4 of the first user study.

2	drums.wav	A	↻
	bass.wav	1	↻
	guitar.wav	1	↻
1	4 bars	1	×

Figure 7.9: An example expected result of scenario 4 from the first user study.

Findings from scenario 4

The user tests were designed to introduce a Time Chooser as playing silence for a given duration or, from an alternative viewpoint, introducing a musical rest. The tutorial and scenario order subsequently introduced the Time Chooser's fundamental role, which is to provide a mechanism for constraining the duration of a Soundable Chooser. Introducing Time Choosers in this way may have created some confusion among users (and consequently user testing in later chapters will reverse the order in which the two functions are introduced). Four of the six groups found that the rest functionality of the Time Chooser was not obvious, although all participants were able to use it effectively after discussion. When the user test moved on to the use of a Time Chooser to control a Full Chooser's duration, one participant continued to interpret the Time Chooser as a rest (as it was when introduced in Scenario 3) rather than as a duration control for the Full Chooser. The user had learned that a Time Chooser was a rest and found the new usage initially confusing; they continued to use the logical framework of rests, rather than re-contextualise the functionality in the Full Chooser. Tellingly, a significant number of participants guessed that Time Choosers would be used to control the duration of Soundable Choosers before the functionality was introduced. This will be considered in more detail in Section 7.5.2.

Conceptually, hard and soft stops were understood immediately by half of the participants. These users were able to make musical use of both stop types. Those who did not initially understand the difference were almost all able to clarify the stop behaviour via a short discussion. One participant was surprised by hearing the effects of a soft stop in his group's final scenario work ('why isn't it stopping?'), prompting another conversation. The stop functionality is further explored in Sections 7.5.1, 7.5.4.

Two groups felt that, while the icon for a hard stop was understandable, the icon for a soft stop did not make its function clear. When the intention behind the icon choice was explained (\times as a hard stop as it mirrors a 'stop' road sign, $>$ as a soft stop as it looks like an arrow to allow the material to continue) some participants felt that it was more understandable. One participant remained unsatisfied with the soft stop icon, although they could not offer an alternative.

You know that's a hard stop [pointing at the hard stop icon] because you think it's like a traffic sign, but I feel like with a soft stop it's hard to really know what this [the soft stop icon] means.

Just as the icon choice had an effect on learnability, the language used for the stops also seems to have been significant. For those participants who did not immediately understand the stop functionality, the hard stops were explained as ‘rude’ and soft stops as ‘polite’. These were the original names used in early development, and they were useful in helping the participants to contextualise the stop behaviour. It is not clear from this process whether one name is more descriptive or learnable than the other as not all participants were introduced to these terms; those who were introduced to the rude/polite terms encountered them as a secondary adjective term to support the hard/soft terminology.

7.3.5 Scenario 5: Understanding a musical example

Participants were shown the Full Chooser shown in Figure 7.10 and were asked the following questions:

3	drums.wav	3	↺
	bass.wav	2	↺
	marimba.wav	1	↺
1	8 bars	1	✕

Figure 7.10: Full Chooser used in scenario 5 of the first user study.

- Look at the Chooser in Figure 7.10 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?

Findings from scenario 5

When asked to produce a Full Chooser which allows for infinite playback, the users were expected to set the Time Chooser’s nose cone to 0 in order to prevent it from controlling the Soundable Chooser’s duration. One user wanted to be able to set the time lane’s duration to ∞ .

The only thing I would say I would have it so it would loop for a certain amount of times for example the drums would be infinite, the bass would loop twice, and then once that’s gone it would select the guitar or marimba.

While infinity is not a currently allowed option for duration, it would potentially be a valid input. Two participants wanted to use the ‘A’ (always play) mechanism to set infinite playback; their logic was that they wanted to override the set duration. Another user wanted to be able to loop the Time Chooser to create infinite playback.

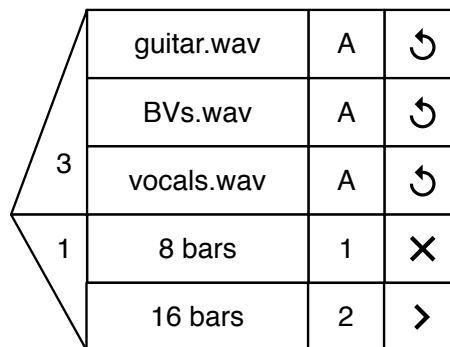
One user asked for a specific visualisation, in which a looping lane in a Soundable Chooser would segment to show the number of times the lane contents would play within the duration set by the Time Chooser. For example, if a duration of 8 bars was selected, and the upper soundable lane contained a 2-bar sample and the lower lane contained a 4-bar sample, the upper lane would show 4 segments and the lower lane would show 2 segments.

It might be good to have ... an analyser so ... you can see how many bars [long a sample is] ... a button you press to reveal the information of how long [a sample] is, because at the moment you have to remember it yourself.

One pair of participants was interested in visual feedback in the form of a progress bar. This presents some non-obvious trade-offs and challenges to the system which will be explored in Sections 7.5.1, 7.5.3.

7.3.6 Scenario 6: Multiple Time Chooser lanes

The participants were shown the Full Chooser in Figure 7.11 which contains a Time Chooser with multiple lanes. They were then asked the following questions:



3	guitar.wav	A	↺
	BVs.wav	A	↺
	vocals.wav	A	↺
1	8 bars	1	✕
	16 bars	2	➤

Figure 7.11: The Full Chooser used in scenario 6 of the first user study.

- 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?

Findings from scenario 6

This scenario, which introduced multiple Time Chooser lanes, was incorrectly explained by two pairs of participants who did not understand the lane selection mechanism. This error may be linked to the application of an understanding of horizontally-scrolling DAWs (Section 2.4.3). The use of numbers for multiple parameters may also have made the functionality unclear.

This scenario asked users to consider the possible nose cone values given three Soundable Chooser lanes with a weight of 'A' (always play). The correct answer (either three to play all lanes, or zero to skip the Soundable Chooser) was quickly identified by most participants. One pair of participants questioned the semantic logic of allowing the 'always play' directive to be overridden and, if it can be overridden, why this can be only performed in one context. This was an interesting example of users expecting consistency in design to allow them to apply their knowledge from within the system, and will be considered in more detail in Section 7.5.3.

7.3.7 Scenario 7: creating a specific Full Chooser

The participants were provided with a range of paper templates for both Soundable and Time Choosers, completed examples of which can be seen in Figure 7.12. The participants were given the following task:

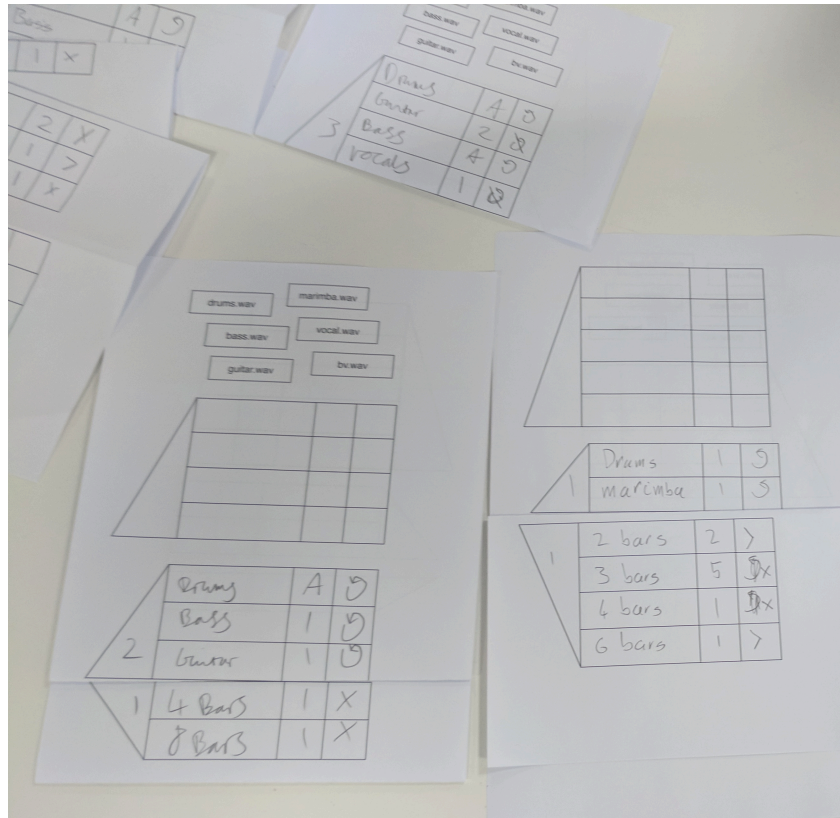


Figure 7.12: Completed paper Chooser templates used in scenarios 7 and 8 as part of the first user study.

- 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.

Findings from scenario 7

This task was performed quickly and accurately by all groups. The users in two groups requested preset Choosers to enable them to build a viable piece of music quickly. Presets could also provide tutorial support by providing a framework around which users could experiment.

7.3.8 Scenario 8: Sequence; playground

Participants were shown a video (Bellingham, 2020c) which introduced the sequence mechanism. They were then asked to complete the following task:

- 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.

Findings from scenario 8

One participant asked about the possibility of using branching arrows to create a nondeterministic selection:

Would it be silly to have two arrows pointing into each box?

This question led to a clarifying discussion regarding the ways in which nondeterministic selection is handled in Choosers.

The participants were able to create some interesting musical material, and all enjoyed hearing the results of their work. This was the first time that any of them had created algorithmic music. Three participants felt that the user interface used numbers for too many parameters.

I mean, I feel like there's a bit too much numbers, does that make sense? I feel like this [lane weights] should be like a graph, something like a key, to help the user understand it.

Upon questioning, their concern was that there was insufficient differentiation between quite different user interface elements, making the interface both difficult to learn and confusing in operation.

If it's for those who ... wouldn't get into coding, then once you start introducing numbers—I mean I'm not against numbers, I think it's important that people should work with numbers—but if the object is to create an interface which is going to be attractive, icons are better.

Numbers are used in the nose cone, the time lanes (e.g. '8 bars'), and the weight column. This will be considered in Section 7.5.5.

One suggestion was to change the Time Chooser nose cone to an on/off icon given that the only legal options were zero and one. It is interesting to note that some users were able to identify that a Boolean value is required here, and the issue will be returned to in Section 7.6.

You could use zeros and ones to say on and off already, but ... it could be an icon, like an on-off switch, in case you don't understand the one or zero [in the Time Chooser nose cone].

Most users used this final task to test their understanding of the system, rather than using it to create a cohesive piece of music. Of the six groups, only one decided to create a piece of music which developed thematically. To do this they wanted to reuse a Chooser, making minor changes to it. We discussed copying the original Chooser, and the participants felt that this met their requirements.

One group was interested in reusing some elements of the composition created in the final scenario, and so we discussed the nesting functionality which was left out of this user study (see Section 6.1.6 for an explanation of nesting in Choosers). They understood the concept and indicated that it met their requirements.

7.3.9 Final formative questions

At the end of the user test, participants were asked the following three formative questions:

- Can you see anything this would be useful for?
- Can you see any ways in which this is similar to other tools you have used?
- Is there anything that is made easier by this system? Anything which was not possible made possible/hard and made easier?

Findings from the final formative questions

Three groups commented on the ‘boring’ design of Choosers v1. The layout of Choosers was not seen as problematic, but some users wished for a more stylish and polished presentation.

It’s quite a dull thing to look at in terms of aesthetic ... design-wise it’s good but it could be better ... because I just wasn’t a fan of the design itself I just kept getting lost, it’s too boring to look at.

Two groups requested colours to enhance usability; within one group, one user wanted automatic colour selection (denoting lane type) and the other user felt that user-controlled colour selection would better support sorting and arrangement tasks. Multiple users were interested to know if lanes could be rearranged to visually organise them into instrumental or musical groups. One user suggested that lane arrangement could be an alternative to the weight column—moving a lane higher would result in a higher probability of playback. This is similar to one mechanism which was considered and rejected before the user tests; it was replaced by the weight column as the column allows for multiple identical lane weights, quick auditioning, and user-controlled lane ordering to assist with musical arrangement. One user requested instrument icons for Soundable Chooser lanes, partly due to their unfamiliarity with one of the instruments used in the scenarios (the marimba).

7.4 Identified issues from user testing

A summary list of the main findings from the first user study of Choosers v1 is shown below. These findings will be loosely categorised and discussed in more detail in Section 7.5.

Confusion in the behaviour of ‘always play’ Some users expected the ‘always play’ weight to take precedence over a zero in the nose cone. The precedence rule in Choosers v1 is the other way around.

No ‘play all lanes’ command There was no simple way to instruct a Chooser to play all available lanes simultaneously, regardless of the number of lanes.

Desirability of infinity as a duration Some users wanted to create infinitely-repeating playback by enabling lane loops and changing the time lane duration to infinity. Infinity is not an available setting in Choosers v1.

‘Always’ confused with ‘infinite’ ‘Always play’ was sometimes misconstrued as controlling duration (‘repeat infinitely’) rather than affecting lane weight (‘always choose this lane’). The phrase ‘always play’ is insufficiently specific and users may reasonably presume that it could refer to either behaviour.

Requests for DAW-like visualisations Some users wanted Soundable Choosers to present a visual segmentation to show the number of repeats of a music object (e.g. a sample) in a given duration. Other users asked for a DAW-style progress bar.

Overuse of arithmetic notation Some users felt that the user interface used numbers for too many parameters.

Vertical alignment misconceptions Some users misunderstood vertical alignment, assuming that vertically-stacked time lanes would play sequentially. Additionally, some users presumed that vertically-stacked time lanes would result in the selection of multiple simultaneous durations.

Confusion over the main role of Time Choosers Time Choosers were introduced as the mechanism for creating musical rests, rather than in their main role as controlling the duration of Full Choosers. Some users found this confusing.

Lack of familiarity with algorithmic music Most users were not previously aware of algorithmic music, leading to some fundamental conceptual issues concerning algorithmic and nondeterministic music composition in general rather than this particular implementation.

Confusion over hard and soft stops Some users found soft stop behaviour confusing. Hard stops were better understood, perhaps as an immediate stop is common behaviour in both DAW and music playback software.

Lack of flexibility in stop behaviour There was a lack of compositional flexibility due to stop behaviour being set globally per Chooser.

Hard and soft stop icons were not universally understood The icon for hard stop was generally seen as clear and understandable, but the icon for soft stop was considered unintuitive by some participants.

‘Boring’ visual design Some users felt that the design was not visually exciting or enticing.

Visual arrangement tools Some users requested visual arrangement tools, such as user-controllable coloured lanes, automatically coloured lanes, and the ability to rearrange lanes (e.g. by instrument category) without affecting the behaviour of Choosers.

Soundable content in Time Lanes confusing Some users felt that the rules surrounding the use of soundable content (e.g. a sample) in a Time Chooser was confusing.

7.5 Reflection on design issues

The findings from the user tests outlined above have various implications for the design of Choosers. These will be considered under five headings: musical issues; programming-related issues; shared and existing knowledge; metaphor; and arithmetic.

7.5.1 Musical issues

Control over repetition and duration are fundamentally important in a music system. The design of Choosers brings both to the surface while allowing for subtle control over the interaction between repetition (via lane loops and multiplication notation) and duration (via Time Choosers containing durations and/or soundable content, and hard/soft stop behaviour). The hard and soft stop system can be thought of in a number of ways. For musicians, one example use might be to consider soft stops as suitable for melodies, and hard stops for accompaniment. Melodies are allowed to finish, whereas accompanying elements are stopped when the duration of the Chooser elapses.

Four of the six pairs of users found the description of stop behaviour in the tutorial video confusing, and benefitted from a verbal explanation with supporting visual examples drawn on a whiteboard. These verbal and visual examples formed part of the enhanced documentation developed for the second user study (documentation development will be shown in Section 8.2 in Chapter 8, and the second user study will be presented in Chapter 9).

Three of the fourteen users were keen to have a visual indication of current position with respect to duration, such as a progress bar. While this seems a reasonable request, it is complicated by the nondeterministic nature of the system; for example, the duration of a lane cannot necessarily be predicted if it has nested contents of nondeterministic duration (e.g. a Chooser selecting between lanes of varying duration).

A significant number of participants found the use of Time Choosers for both rests and Chooser duration to be confusing. Pending the next user study, it was tentatively assumed that this was due to the tutorial order; rests were introduced before the Time Chooser’s primary function, which is to control the duration of a Chooser.

None of the participants had experience in algorithmic composition, so these sessions performed two different kinds of work—testing the interface while at the same time introducing concepts and tools for nondeterministic composition. This lack of prior experience led some participants to presume that the concepts themselves were novel. Some time was spent discussing the desirability of algorithmic processes rather than this specific implementation. Two participants assumed that the process of creating music using Choosers would result in a linear audio file, which indeed it can, but many use cases would require the music to remain nonlinear. Future evaluations

could explore the nature of the apparent resistance to nonlinear playback, including how it is related to expectations set by commercial music creation software and linear playback.

One group specifically wanted a mechanism to allow them to implement thematic development by referencing patterns used earlier in a Chooser sequence. The design of Choosers allows for this via nesting, although this mechanism was not included in the user tests for simplicity. The users were shown nesting in response to their questions and agreed that this met the need they had expressed.

7.5.2 Programming-related issues

As previously shown in Section 6.2.5, the Soundable Chooser nose cone slopes down and the Time Chooser nose cone slopes up—this suggests that they can be joined together and communicates the required upper/lower order to the user. Interestingly, some users guessed the combination of Soundable and Time Choosers, suggesting that the nose cone shapes of the two Chooser types helped communicate their combinatorial usage.

The interface and interaction designs aims for consistency across Soundable and Time Choosers, where possible. Participants in the user tests successfully reused elements of the Soundable Chooser system when manipulating duration, but there were some cases where such reuse or re-contextualisation was not possible, as will now be described. Interestingly, the actions of the users in these cases would have made sense neither from a musical nor programming perspective, but the rationale behind these requests is instructive as it shows how some users understood the system. For example, in scenario five (Section 7.3.5) two participants tried to use the ‘A’ (‘always play’) mechanism outside of lane weights as an ‘override’ or ‘maximum’ setting; in this case they wanted to use it as a duration to create infinite playback (‘always play and never stop’). In a similar example, one user wanted to be able to loop a Time Chooser. If the system were to be changed to allow for a set number of repeats per-lane, rather than only allowing for a Boolean choice between no loop or an infinite loop, such a move might be desirable. This will inform the development of loop behaviour in Choosers v2 (to be considered in Section 8.1.3, and in Section D.2.1.2 of Appendix D).

Users will also require access to metadata; for example, to check the length of a sample loaded into a soundable lane in a Chooser. Such metadata could be shown via a tooltip, accessed by hovering the mouse over a lane.

7.5.3 Shared and existing knowledge

One design motivation is to enable people to quickly understand the system. As one tool for achieving this, the Chooser design tacitly draws on a number of systems of existing knowledge.

Some users wanted to be able to leverage their existing understanding of DAW software, and found it frustrating that they needed to learn new paradigms for duration, synchronicity, and so on. This may be seen as an example of technological framing (Orlikowski and Gash, 1994, see Section 2.5). The knowledge gained by using other music software can be useful, but it can also prove counterproductive if the design of the software being learned is sufficiently different. As a result, there is much to be gained by following standard design conventions where possible, as this maximises the user’s ability to reuse existing knowledge. Similarly, the consistent use of similar semantics can enhance the learnability and usability of software (Green and Blackwell, 1998, see Section 4.2.8). Interestingly, one pair of users expected a greater degree of consistency in scenario 6 (Section 7.3.6); the participants correctly identified that the interplay between the nose cone and ‘A’-weighted Soundable Chooser lanes was a special case which required them to learn new rules. This issue led to the infinity option which will be introduced in Section 8.1.1; the infinity option maintains functionality while enhancing consistency.

7.5.4 Metaphor

Interface metaphors are common and can be useful in communicating the roles of the software and setting realistic expectations. However, such design decisions are only powerful if users are already familiar with the artefacts from which the interface is derived. Related to technological framing is the assumption, ubiquitous in Digital Audio Workstations, that signal flow and processing will be applied using a mixing desk metaphor. Such virtual desks often make use of skeuomorphism (such as the fader caps and rotary potentiometers in *Pro Tools*), although some other designs present a more abstract design—see Section 2.4.3 in Chapter 2 for an example.

Given that DAWs are now capable of performing all mixdown tasks, and the financial cost of consoles and outboard effects processors can be prohibitive, many users learn in a virtual studio environment rather than on hardware. A number of DAWs were designed to mimic hardware in order to leverage existing knowledge and ease the transition from hardware to software. However, 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.

Some users felt that the user interface was ‘boring’, lacking the use of colour, metaphor, and skeuomorphic design common in DAWs (see Section 2.4.3 in Chapter 2). This is understandable given that all participants were familiar with at least one DAW and that the design of Choosers under consideration did not use colour, texture, or visual emulations of hardware.

Some users had difficulty understanding the outcome of hard and soft stops in Choosers. The vast majority of music production software is focussed on the creation of linear music; as a result, none of the user test participants had encountered it, and did not have a frame of reference for why it might be desirable. As a result, there is not a clear existing metaphor for what is referred to here as a ‘soft’ stop. Users agreed that the \times icon represented a traffic stop sign and that it was a suitable analogy for ‘stop now’, but the $>$ icon used for a soft stop was not immediately understood as there is no readily accessible metaphor.

7.5.5 Arithmetic

The use of numbers and arithmetic relationships in an interface can be a valuable organising tool, as they are more or less universally familiar and can 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. Upon questioning, the issue was that numbers meant different things in different parts of the interface. The Chooser design presented to participants in the user tests made use of integers in five different ways: for the number of simultaneously playing soundable elements, weight, duration, repeats, and Time Chooser on/off.

Despite this, for different reasons, the user issues surrounding the ‘always play’ option led to an extension of the range of numerical concepts used in the interface, by allowing the metaphorical use of ∞ as a weight (to outrank any positive integer weight). This will be discussed in Section 7.6 where changes to Chooser design will be proposed to address the various issues that have been identified so far.

7.6 Design problems and candidate solutions for the second user study

Given the problems for some users with the use of integers for multiple parameters in Choosers v1 (see Section 7.5.5), the use of a simple on/off icon will be explored as a candidate solution for the Time Chooser nose cone in Choosers v2. Interestingly, one pair of users suggested this change in the user tests. Scenario five, outlined in Section 7.3.5, showed that two users wanted to leverage the ‘always play’ mechanism beyond the weight column, and one user wanted to set the duration of a time lane to infinity. I propose a change to Choosers which allows for both mechanisms.

More specifically, the proposed design change will enable the user to allocate a maximum possible weight (∞) for a lane, thereby guaranteeing that it will play if the nose cone number is high enough to allow all such lanes to play. When allowing ∞ as a weight, a useful metaphor is to think of lanes with weight ∞ as having paid for ‘priority boarding’, as when boarding an aircraft. Lanes with weight ∞ will always be chosen before any lanes with any finite weight. Compared with the ‘always play’ mechanism, this has the potential for greater clarity when the number of maximally-weighted lanes exceeds the number of the nose cone. In such cases, under the current ‘always play’ system it is not obvious whether ‘always play’ should override the nose cone or vice versa. Under the proposed system, the nose cone would determine the number of lanes to play, and if that was less than the number of lanes with weight ∞ , the winners would be chosen from those lanes at random. The use of a maximum value (∞) for the nose cone of a Soundable Chooser (‘play all available lanes’) and for the duration of a time lane (‘play forever’) will also be considered.

To avoid an unnecessary misunderstanding identified in the first study, future work will introduce Time Choosers in the context of a Full Chooser, with the ‘rest’ functionality introduced later as a special case. Tutorial materials will provide a clear explanation and will offer context and examples. The value of all of these proposed changes will be tested empirically.

One of the limitations of this user study has been that all participants were familiar only with mainstream popular music-making; these users had yet to develop an appreciation of the desirability of nondeterministic selection of lanes, phase music, and other alternative approaches. To address this issue, and to consider potential wider uses, Chapter 11 presents interviews with domain experts, including a composer whose work has included indeterminate and aleatoric processes.

At this stage, while the design is undergoing refinements based on the findings from the user test presented in this chapter, it is worth considering how the unusual combination of usability and affordances may be suited to music in which users would benefit from easy access to non-linear playback. Some classic Minimalism techniques (Potter, 2002), such as phasing (Scherzinger, 2005), are easily achievable using Choosers. Choosers is readily adaptable to take external input (e.g. a footswitch) via such protocols as OSC to trigger changes in the music in real-time (for an example, see the sketch in Section 6.2.6 in Section 6.2). Choosers also have the capacity to create nonlinear versions of existing recordings by loading alternate takes into Chooser lanes. The playback could range from very close to the original (e.g. nondeterministically switching between vocal takes of the same melody) to playing significantly different material (e.g. branching to play different sections), depending on the decisions made by the creators.

7.7 Conclusions

This chapter has presented the design, implementation, and analysis of the first user study. While participants were capable of using Choosers to create nondeterministic music, the current design of Choosers presented a number of issues. A discussion on the key issues, presented in Section 7.5, led to the identification of candidate solutions in Section 7.6. In order to respond to this chapter’s findings, Chapter 8 will use them to reflect on and elaborate key design issues before exploring, annotating, and explaining the resulting changes to Choosers.

Chapter 8

Design considerations and inspection evaluation for the second user study

This chapter describes a redesign of Choosers by synthesising the findings from the first user study outlined in the previous chapter. A series of programming walkthroughs, performed as heuristic evaluations without participants, are used to understand the impact of the proposed design changes. The resulting design, Choosers v2, will be used in the second user study presented in the following chapter.

At this point this thesis has outlined the design exercises (Chapter 5) which were used to inform the design of Choosers (see Chapter 6). Chapter 7 outlined the first user study, including a discussion of the findings from this first participant test. This chapter will consider the main issues that were uncovered by the first user study and show how the design of Choosers v2 was amended in response, including explanations and annotations of design decisions. Figure 8.1 shows an example of the completed design, demonstrating the use of the infinity symbol ∞ (used to represent infinity in non-technical contexts, and in mathematics to represent potential rather than actual infinity); the removal of columns; new stop icons; a new weight icon; and the ability to make soundable content in a Time Chooser lane audible via a mute/unmute icon.

In keeping with the use of a Research through Design methodology (see Section 3.2), the design of Choosers v2 involved the development of various alternative solutions to the issues identified in Chapter 7, listed in Section 7.4, and discussed in Section 7.5. Some design branches were identified but, due to resource constraints, were not the subject of subsequent participant testing. Following the work of Gaver (2012), an annotated portfolio of multiple iterative designs is presented below.

8.1 Adopted changes to the notation

8.1.1 Infinity

One of the issues that emerged from the first user study was linked to the use of 'A' ('always play'). As outlined in Section 7.6, a small number of participants wanted to use 'A' to mean 'the maximum possible'; for example, two participants wanted to use it as a duration. It therefore seemed desirable to offer a 'maximum' option which is more consistent with the arithmetic notation used for weights, nose cone values, and durations.

8.1.1.1 Infinity as weight—priority boarding

Given that lane weight is expressed using numbers, it can be argued that the use of infinity is more consistent than using a single exceptional mnemonic alphabetic character ('A'). Additionally, allowing infinity as a weight gave us an opportunity to reconsider the selection rules and to both clarify

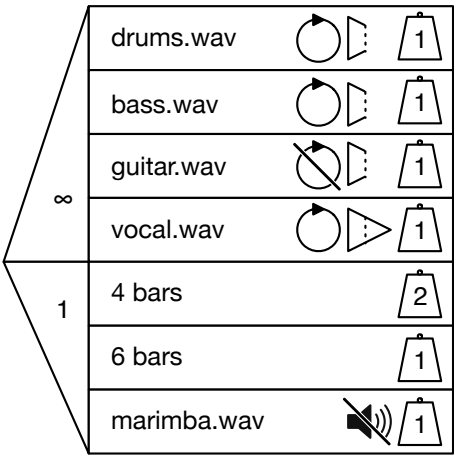


Figure 8.1: An example of the final design of Choosers v2. Note the use of infinity (used here in the Soundable Chooser’s nose cone); no internal columns; new icons for hard stop (the upper three Soundable Chooser lanes) and soft stop (the vocal . wav lane); new weight icon; and mute/unmute icon for soundable content in a Time Chooser lane (the marimba . wav lane).

and simplify the ways in which seemingly contradictory parameters are resolved. This required a new frame of reference (Orlikowski and Gash, 1994)—see Section 2.5—created via a loose analogy intertwining two metaphors; priority seating, and a lottery for the remaining seats.

In the analogy the nose cone number can be thought of as representing the number of seats available on an aeroplane. Lanes with infinite weight represent the priority boarding passengers, and they are given the first seats on the plane. If there are still seats left after the priority boarders have taken their seats, there is a lottery held among the remaining passengers; the lane weight represents the number of lottery tickets each passenger has, and more tickets will make selection more likely. If there are too few seats to accommodate the priority boarders there is a straight lottery between the priority boarders to fill the available seats.

The analogy is useful in explaining the behaviour of seemingly contradictory settings. If the nose cone number is lower than the number of infinity-weighted lanes then the selection will only occur between the infinity-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.

The ‘always play’ design from the first user study is shown next to the updated ‘infinite weight’ design for Choosers v2 in Figure 8.2.

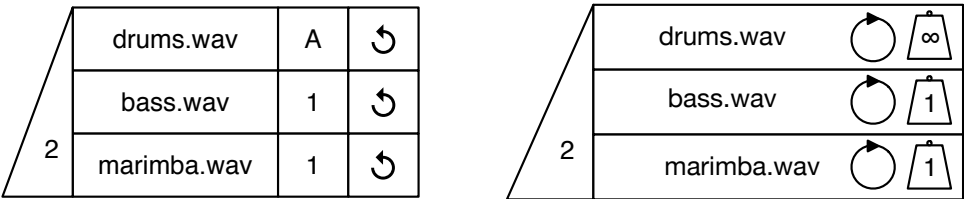


Figure 8.2: The ‘always play’ weight used in Choosers v1 (left); the use of infinity as a weight, used in Choosers v2 (right).

8.1.1.2 Infinity as duration—play forever

The use of infinity as a duration provides improved design consistency while offering the behaviour that users expected in the first user study. In allowing the user to specify an infinite duration—‘play forever’—it offers an alternative to running a Soundable Chooser without a Time Chooser, or disabling the Time Chooser’s nose cone in a Full Chooser by setting it to zero. In this way the user

is offered three routes to create the same result, allowing both for personal preference and for easy auditioning in a number of different contexts.

The user is free to use infinity with or without a unit of duration; that is to say, a Time Chooser lane can contain ' ∞ ', or ' ∞ bars' (or beats, minutes, seconds, or any other legal duration unit). The option of including a unit of duration reduces viscosity when making a change either from or to a finite duration. For example, consider changing a Time Chooser lane from '16 bars' to ' ∞ bars', or visa versa.

Figure 8.3 shows three ways to create infinite playback in Choosers.



Figure 8.3: Three ways to create a Chooser with infinite playback: a Soundable Chooser with looping lanes and no Time Chooser (left); a Full Chooser with a deselected Time Chooser via a zero in the nose cone (middle); a Full Chooser with ∞ as the duration¹ (right).

8.1.1.3 Infinity as nose cone value—play all available lanes

The inclusion of infinity as a valid nose cone value was not in response to observed behaviour from the first user study, but instead came from a desire for consistency. Using the same priority/lottery metaphor as infinite weights, using infinity in the nose cone instructs the Chooser to select all lanes with a weight greater than zero (all those 'with a ticket'). Infinity in the nose cone is a convenience which reduces the mental work required (Cooper et al., 2014) if the user wants to hear the result of all selectable lanes together. Additionally, setting the nose cone to infinity and then deselecting lanes via zero weights offers an alternative, subtractive, approach to arrangement.

Infinity as a nose cone value is shown in Figure 8.4.

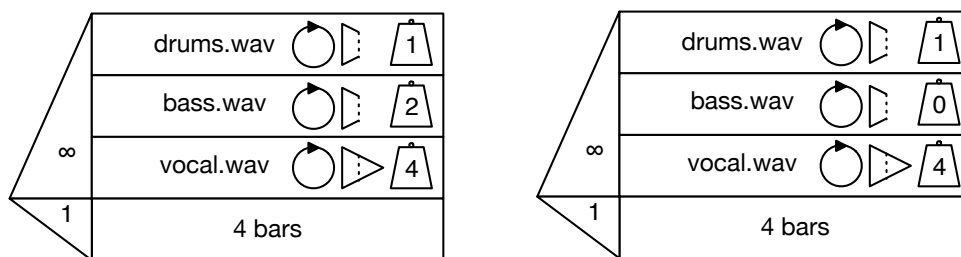


Figure 8.4: Infinity used in the nose cone to play all available lanes. Note that the example on the left has three lanes with non-zero weights, so all three lanes will be selected to play. The example on the right has one lane with a weight of zero, meaning that only two lanes are available and so only two lanes will be selected to play.

8.1.2 Stop behaviour

While testing the previous v1 Choosers design it became clear that the use of a global stop behaviour was a limiting factor in the creation of some musically engaging processes. The previous behaviour,

¹Note that, in this figure, the duration of ∞ is shown without a corresponding duration unit as the duration unit is no longer relevant in this context. An option would be to always show the duration unit (e.g. ' ∞ bars'), which would both provide consistency and lower viscosity by clearly signalling to the user how the value may be changed. A third option is to show duration units only for finite durations.

outlined in Chapter 6, used a stop command in Time Chooser lanes global to the Full Chooser. The motivation for exploring options came from considering the behaviour of lead/melody lines, which often need to play in their entirety, and rhythm/backing parts, which support the lead/melody line and often stop when the lead line has finished. Initial informal design pilots focussed on a new behaviour—a lane that could react to another when set to ‘listen’ to the lead line. However, in a move motivated by consistency and parsimony, two key changes were implemented.

First, stop behaviour moved from a per-Chooser global control in Time Chooser lanes to per-lane behaviour in the Soundable Chooser. An early example of this can be seen in Figure 8.6, with a later design used in the simple sequence shown in Figure 8.5. This change allows the user to control stop behaviour in a more granular way. Secondly, soundable content used in a Time Chooser lane to specify a duration could optionally be made audible via a mute/play switch. This allows for an audible lead line to control the duration and, once the audible lead line has finished, other lanes can be set to stop immediately via hard stops. This process, including draft icons, is outlined in Section 8.1.4.

Figure 8.5 shows the use of per-lane hard stops and Time Choosers to set exact durations for each Chooser, regardless of the length of the soundable material. Note that the `bass.wav` audio file is set to not loop in the first two Choosers; if it is shorter than the Time Chooser duration the sample will stop and there will be a rest, or a gap in playback, until the Time Chooser duration has elapsed and the next Chooser in the sequence is played. In the second Chooser the `drums.wav` lane is set to loop, leading to the possibility of the bass stopping (depending on the bass sample’s duration) and the drums continuing to play. In the third and final Chooser in the sequence, both Soundable Chooser lanes are set to loop, meaning that they will play for the entirety of the Time Chooser duration before stopping.

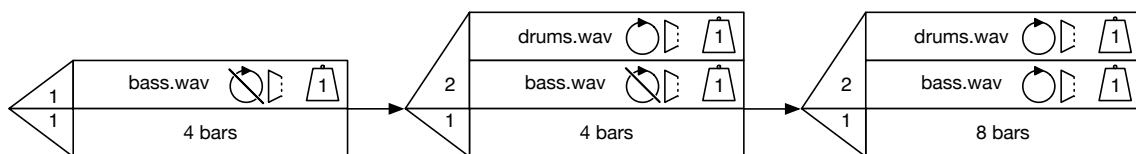


Figure 8.5: A sequence to show the use of Time Choosers and hard stops for exact duration control.

8.1.2.1 Time Chooser nose cone

As outlined and discussed in Section 7.3 and Section 7.5, multiple participants in the first user study were confused by the meaning of the number in the nose cone. After the first user study, options for the Time Chooser nose cone were considered. An on/off icon was sketched out to replace the 0/1 binary choice used in Choosers v1; an example design is shown in Figure 8.6. The motivations for the proposed change were to reduce the use of arithmetic notation (numbers) and to make the two permitted choices immediately clear. An overuse of arithmetic notation had been identified as potentially problematic in the first user study. The use of an on/off switch informs the user that there are only two possible settings; arithmetic notation suggests that numbers greater than one are selectable.

While it was deemed a suitable candidate design the on/off Time Chooser nose cone design was not used in the second user study for the following two reasons:

1. The use of 0/1 notation is consistent across both Soundable and Time Chooser nose cones, allowing users to use the same mental model ('pick 1 of these lanes');
2. An icon could increase the work done by the user; they have to recognise the icon and parse its meaning. The load on the user is not a trade-off as they have already had to do the work to understand the arithmetic nose cone design in Soundable Choosers. So, there is a choice of adding another step or reusing the existing mechanism and allowing the user to leverage their existing understanding.

drums.wav	1	✕	↺
bass.wav	1	➤	↺
marimba.wav	1	➤	↺
8 bars	1	✕	

Figure 8.6: A sketch showing an on/off button replacing a Boolean 0/1 choice in the nose cone of a Time Chooser. Note that a stop icon is still used in the Time Chooser in this early sketch; this was a test of a master stop behaviour control which would set all stops in the Soundable Chooser lanes to either a hard or soft stop in a single action.

8.1.3 New icons and a column-based layout

The design of Choosers v1 (see Chapter 6) made use of a consistent number of columns per lane in both Soundable and Time Choosers in order to visually align common parameters such as weight. As outlined in Section 8.1.2, the design of Choosers v2 moves stop behaviour from a global control in Time Chooser lanes to per-lane behaviour in the Soundable Chooser. This change resulted in an inconsistent number of lane icons between Soundable and Time Choosers, leading to sketches which explored alternative designs to maintain vertical parameter alignment where appropriate. Some early sketches, such as Figure 8.6, explored the use of an inconsistent number of columns per lane to maintain vertical alignment. An alternative design explored the removal of internal columns and the use of progressive disclosure to show icons only when relevant; an example is shown in Figure 8.1. This design was chosen for Choosers v2 as it allows for vertical alignment, progressive disclosure, and a variable number of parameters per lane. Due to the loss of columns this design required the development of icons to communicate each parameter; the icon designs are detailed below.

User feedback from the first user study showed that some users were confused by the use of numbers for multiple parameters in the interface. Numbers meant different things in different parts of the interface resulting in a lack of clarity, and motivating the review of icons detailed in Section D.2.1 of Appendix D. The final icon designs for Choosers v2 are shown in Figure 8.1. A secondary motivation for reviewing icon design was to facilitate the removal of internal columns while improving the legibility of the design.

8.1.4 Management of soundable content in Time Choosers

As part of the development of the more granular stop behaviour outlined in Section 8.1.2, the functionality of soundable content in a Time Chooser lane in Choosers v2 has been enhanced to include a mute/unmute icon.

The previous design used in Choosers v1, introduced in Section 6.2.5, allowed soundable content to be used in a Time Chooser lane. If this Time Lane forms part of a Full Chooser the soundable content's duration would be used to moderate the duration of the Full Chooser. If the user wanted to use the soundable content for both duration and soundable output the previous design required them to put one copy of the soundable content in a Time Chooser lane for duration, and another copy in a Soundable Chooser lane for playback—see Figure 6.34 for an example.

The new design allows a sample to be used for both duration control and playback without having to duplicate it in both the Soundable and Time Choosers. When soundable content is added to a Time Chooser lane it is muted by default; if selected to moderate duration, the lane's contents will be used to set duration but will not be played. If the user wants the lane's contents to both moderate duration and to play as part of the Chooser's audible output, they can choose to unmute

the lane. Various candidate icons were created; see Section D.2.3 of Appendix D for examples. All the icons use a speaker, which reflects the standard icon for mute/unmute in consumer audio software. Professional audio software typically uses ‘M’ for mute, which in turn has been taken from audio console design.

8.2 Changes to the documentation

The previous chapter (Chapter 7) presented the design and findings from a participant programming walkthrough evaluation of Choosers v1. This chapter has so far used those findings in the design of Choosers v2. The next chapter (Chapter 9) will present a second participant programming walkthrough to evaluate the design of Choosers v2.

In addition to the notation changes in Choosers v2 (listed in Section 8.1), several of the issues identified in the first user study (outlined in Section 7.4) required changes to the supporting materials. These changes are listed and discussed below.

8.2.1 Explaining how infinity is used

Following earlier finding in this chapter, and in preparation for a second user study (Chapter 9), an extended tutorial video was prepared to present the metaphors of priority aeroplane seating and a lottery (see Section 8.1.1.1). The video was designed to use a ticket as an analogy to reinforce precedence; if a lane has a weight of zero it cannot be selected, regardless of the nose cone value. A still from the tutorial video is shown in Figure 8.7. The video presents a frame of reference allowing users to consider the interaction between the nose cone and lane weights. More generally, it provides a frame of reference for the nondeterministic selection of musical actions which may help users conceive of music algorithmically and free up assumptions concerning linearity inherent in phonography (Section 2.3) and DAW design (Section 2.4.3).

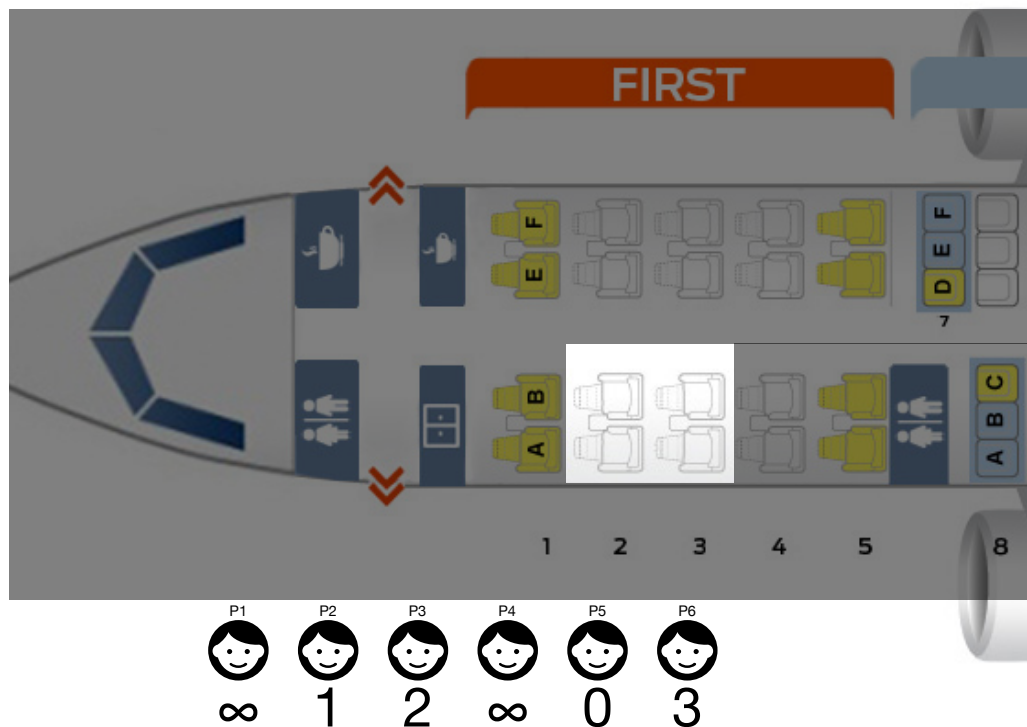


Figure 8.7: A still from the tutorial video showing the aeroplane priority seating/lottery ticket analogy.

8.2.2 Improved scenario content and order

The design of the second participant programming walkthrough, to be presented in the next chapter (Chapter 9), is broadly similar to the design of the first programming walkthrough (see Chapter 7). As well as introducing the updated notation and changing questions and tasks accordingly, the series of scenarios presented to the participants was reordered for two main reasons:

1. The scenario showed musically meaningful uses for Choosers. This was done to show participants how the system could be used and to encourage them to think about music non-deterministically;
2. Time Choosers were introduced in their primary role—moderating the duration of a Soundable Chooser—rather than as a musical rest. While the use of Time Choosers as rests has not changed it is a secondary function; introducing them as rests in the first user study appeared to introduce confusion.

8.2.3 Enhanced tutorial material for hard and soft stops

Choosers v2 introduces a per-lane stop setting, rather than the per-Chooser stop setting used in Chooser v1. The motivations for the change are outlined in Section 8.1.2. Some users had found the concept of a soft stop challenging (see Section 7.5.1) and so the tutorial materials used to introduce hard and soft stops were enhanced. The updated tutorial materials will be shown in the next chapter.

8.2.4 Clarification on vertical alignment

Participants with experience of using DAW software come with a set of expectations (see previous discussions on technological framing in Section 2.5 and the analysis of the first user study in Section 7.5.3), and so the tutorial documentation and ordering of materials needed to be updated to clarify the points at which this framing would otherwise be harmful. One example of this is the use of vertically aligned lanes as allowing nondeterministic choice rather than always referring to synchronous playback.

8.2.5 Clarification on Time Chooser nose cone

As previously discussed in Section 8.1.2.1, there are trade-offs in changing the Time Chooser nose cone from a 0/1 Boolean numeric choice to an on/off button. This process resulted in the choice to maintain the 0/1 number. The tutorial materials were strengthened to make both the similarities and differences clear between Soundable Chooser and Time Chooser nose cones. The most important difference for novice users is the Time Chooser nose cone's limitation to select either 0 or 1 lanes; multiple simultaneous durations do not make sense in the context of a single Chooser.

8.2.6 Clarification on order of priority

The first user test showed that some participants did not readily understand the order of precedence, most markedly between the nose cone and lane weights. The aeroplane analogy, outlined in Section 8.1.1.1 and used in Section 9.2.3, was designed to make the order of priority clear. Once understood, a seemingly contradictory situation in the tentative design can be resolved. For example, if the nose cone is set to 3 but only two lanes have a non-zero weight, then only two lanes can be selected. The analogy drawn is that, although there are three seats, only two passengers have tickets.

8.3 Conclusions

In summary, the following design changes were implemented for the second user study, to be presented in Chapter 9. Table 8.1 itemises the changes with reference to the relevant issues from the first user study, outlined in Chapter 7.

Based on the first user study (see Section 7.5.2 in Chapter 7), infinity was introduced into Choosers v2 (Section 8.1.1) for participant evaluation in the next chapter. Infinity was introduced as a valid weight (Section 8.1.1.1), replacing the previous ‘always play’ option. Infinity was also added as a valid duration for a lane in a Time Chooser (Section 8.1.1.1) and as a nose cone value to play all selectable lanes (i.e. all lanes with a non-zero weight; Section 8.1.1.3). Stop behaviour was moved from a per-Chooser global setting to a per-lane setting (Section 8.1.2), allowing for more complex and musically meaningful behaviour. The use of soundable content in a Time Chooser lane was simplified via a play/mute option (Section 8.1.4). The legibility of the notation was enhanced via refreshed icons and the removal of internal columns (Section 8.1.3).

Turning to the tutorial materials for Choosers v2 (see Section 8.2), changes were made to support new behaviour introduced in Choosers v2, and to better communicate behaviour based on the findings from the first user study. A new metaphor to explain the interplay between lane weights (including infinity) and the nose cone value was developed (Section 8.2.1). The introduction of hard and soft stop behaviour was enhanced to better communicate behaviour (Section 8.2.3). Finally, the content and ordering of the scenarios was updated in order to better introduce key nondeterministic music concepts to novice users (Section 8.2.2).

Table 8.1: Issues in the design of Choosers v1 and supporting tutorial materials, identified in the first user study (see Section 7.5 of Chapter 7), mapped to proposed changes in the design of Choosers v2 and supporting materials.

Issue from the first user study	Change for the second user study
Always play is not consistently understood (Section 7.5.2).	Infinity introduced as the maximum weight setting (Section 8.1.1.1). Infinite weight does not override the nose cone value. The revised design introduces the use of infinity in three distinct places with three distinguishable meanings, motivated by considerations of design parsimony and consistency.
No way to quickly tell a Chooser to ‘play all available lanes’.	Infinity nose cone, with zero weight to remove from selection (Section 8.1.1.3). The use of infinity in the nose cone is a convenience motivated by consistent design; it results in the selection of all non-zero weighted lanes in a Soundable Chooser.
Some users wanted to use ‘always play’ as a duration in a Time Chooser lane (Section 7.5.2).	Choosers v2 allows a Time Chooser lane duration to be set to infinity, which provides consistent design while providing the behaviour some participants expected in the first user study. The functionality is identical to a Soundable Chooser with no Time Chooser, or a Full Chooser with a disabled Time Chooser (Section 8.1.1.2).
Some users felt that arithmetic notation was overused (Section 7.5.5).	The revised icons include a weight icon to visually differentiate all numeric elements. The new self-contained weight icon allows for the removal of internal columns. The removal of internal columns allowed for only selectable icons to be shown, and for a varying number of icons per lane (Section 8.1.3).

Issue from the first user study	Change for the second user study
Some users assumed that vertical alignment referred to sequential playback.	The supporting documentation, consisting of tutorial videos and examples, has been improved (Section 8.2). Particular emphasis has been placed on the priority seating and lottery metaphor for lane weights (Section 8.2.1), clarifying vertical alignment (Section 8.2.4), and an enhanced explanation of hard and soft stops (Section 8.2.3). Improved tutorial order (Section 8.2.2).
The main role of Time Choosers (moderating the duration of a Full Chooser) was not always understood due to their introduction as the mechanism for inserting musical rests (Section 7.5.1). Algorithmic composition was a new area for participants (Section 7.5.1).	Given an assumption that most of the participants will not be familiar with algorithmic composition, the scenarios were developed to introduce nondeterministic composition alongside the notation (Section 8.2.2).
Soft stop behaviour was unclear as it challenged learned stop behaviour (Sections 7.5.1, 7.5.3, 7.5.4).	The previous design, Choosers v1, applied stop behaviour globally per Chooser. The updated design of Choosers v2 provides per-lane stop behaviour, allowing for more granular musical control (Section 8.1.2). Tutorial documentation has been improved (Section 8.2.3).
Hard and soft stop icons were not universally understood.	New stop icons (Section D.2.1.3).
The visual design was seen by some as ‘boring’ (Section 7.5.4). Some users felt that the rules surrounding soundable content in Time Chooser lanes was confusing—consistent but overly complex—an interesting example of consistency and parsimony having a negative impact on usability.	Removal of internal columns and new icons (Section 8.1.3). In order for a sample to be both audible and used for duration moderation the previous design required it to be used twice—once in the Soundable Chooser and once in the Time Chooser. The updated design simplifies this by adding a play/mute switch for any soundable content used in a Time Chooser lane, meaning that it can be used for duration only or for duration and playback (Section 8.1.4).

Chapter 9

Second user study

This chapter presents the design of, and findings from, a participant programming walk-through evaluation of Choosers v2. The findings identify specific design problems and issues which will inform the design of Choosers v3.

This chapter presents the second user study, designed to test the design of Choosers v2 via a programming walkthrough evaluation with pairs of undergraduate Music Technology students as participants. As in the first user study (Chapter 7), the evaluation was designed to test the ability of self-taught music producers without programming skills to use Choosers v2 to carry out a range of rudimentary nondeterministic composition tasks, to identify user experience issues in the current design, and to identify tensions and trade-offs in the interaction design of the system.

The key functionality of Choosers v2 was introduced via short tutorial videos. Participants were asked to carry out musical tasks, solve musical problems, and tackle an open-ended musical challenge by sketching Choosers diagrams on a whiteboard. The facilitator (i.e. the thesis author) played a Wizard of Oz role (outlined in Section 3.6), creating runnable code via a prototype non-graphical prototype version of Choosers. In this way the participants were able to instantly hear their creations and experiment to create a range of outputs.

The findings from the first user study were presented in Section 7.3 and discussed in Section 7.5. These led to the proposed design changes listed in Section 7.6; the resulting revised design was presented in Chapter 8 and summarised in Table 8.1.

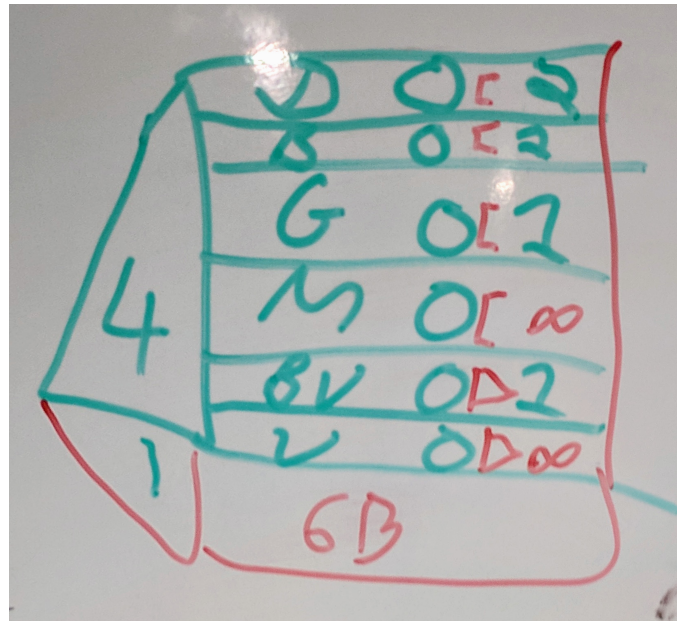
9.1 Method

9.1.1 Participants

Following the protocol outlined in Section 3.5, and as in the first user study (Section 7.2.1), the second user study was undertaken with six pairs of undergraduate Music Technology students as participants. Given that both user tests are designed to introduce Choosers to first-time users, a new cohort was used for the the tests outlined in this chapter. The users were introduced to each element of the graphical programming language via updated tutorial videos (Bellingham, 2020b). Users were then given a range of practical tasks to complete on a whiteboard, and their sketches were played by the facilitator using a set of *SuperCollider* (McCartney, 2002) classes written to implement the musical abstractions behind the system. As with the first user study, the sessions were videoed and transcribed to assist in the analysis presented here. An example of an annotated transcript from this round of user studies is presented in Appendix C.

An example of a participant sketch and the corresponding *SuperCollider* code used by the facilitator is shown in Figure 9.1. This example shows a Full Chooser consisting of a Soundable Chooser with six lanes and a Time Chooser with one lane. The Soundable Chooser's nose cone is set to 4,

and all Soundable Chooser lanes are set to loop. The lane contents have been abbreviated by the participants; 'drums', 'bass', 'guitar', 'marimba', 'bv' (backing vocals), and 'vox'. The 'bv' and 'vox' lanes are set to a soft stop, with the other four lanes set to a hard stop. Lane weights include two lanes set to infinite weight, 'marimba' and 'vox'; this means that they are certain to be two of the four Soundable Chooser lanes selected. The Time Chooser nose cone is set to 1, and the single Time Chooser lane has a duration of 6 bars. Lane weights are not required for this particular Time Chooser as there is only one lane to select from. In order for the facilitator to provide near-instant playback as part of the Wizard of Oz technique, a series of templates had been prepared and were edited to match the participant sketches.



```
~ch1 = Xhooser.new;
~ch1.noseCone_(4);
~ch1.addLane( Lane.new.weight_(2).namedSample(\drums).loopOn);
~ch1.addLane( Lane.new.weight_(2).namedSample(\bass).loopOn);
~ch1.addLane( Lane.new.weight_(1).namedSample(\guitar).loopOn);
~ch1.addLane( Lane.new.weight_(inf).namedSample(\marimba).loopOn);
~ch1.addLane( Lane.new.weight_(1).namedSample(\bv).loopOn.softStopOn);
~ch1.addLane( Lane.new.weight_(inf).namedSample(\vox).loopOn.softStopOn);
~tc1 = TimeChooser.new;
~tc1.noseCone_(1);
~tc1.addLane( TimeLane.new.beats_(6));
~ch1.timeChooser_(~tc1);
```

Figure 9.1: An example of a participant sketch and the corresponding *SuperCollider* template code, edited by the facilitator during the sessions to enable participants to hear the result of their sketched Chooser. Note that the whiteboard pen colours do not have special meaning; the Chooser was drawn collaboratively by two participants, each using a different pen colour.

Of the twelve participants, eleven had some experience in playing a musical instrument at some point. Five participants considered themselves entirely self-taught on their instrument, and seven had undertaken some formal instruction via a GCSE Music qualification. The most common instruments were piano, drums, and guitar. Of the musicians, only two had any experience as performers. Five of the eleven musicians felt they could read common music notation if given sufficient time; none could read and perform simultaneously. Seven could read chord charts, three could read rhythmic notation, and two could read guitar tablature. All participants were familiar with DAW software: the most commonly used software was *Logic Pro* (9 users), followed by *Pro Tools* (5), *Reason* (3), *GarageBand* (2), *Cubase* (1), *FL Studio* (1), *Ableton Live* (1), and *LMMS*

(Linux Multimedia Studio) on Windows (1). All had used common music production hardware; as expected, all twelve participants had experience in using a MIDI controller, followed by hardware drum machines or samplers (7), drum pads/triggers (6), vinyl turntable (3), and a DJ controller (2). Three-quarters of the participants (nine out of twelve) had experience in using *Pure Data* (Puckette, 1997), a visual programming language discussed in Chapter 4. Of these nine participants, six also had some limited experience using *SuperCollider* (McCartney, 2002), one mentioned Excel, and one listed HTML/CSS. Three participants said that they had no programming experience. Finally, ten of the twelve participants said that they did not know what algorithmic music was. The remaining two participants said that they knew what it was, but that they had never created any algorithmic music.

9.1.2 Walkthrough protocol

Following the protocol presented in Section 3.5, and as in the first user study outlined in Chapter 7, this evaluation followed the programming walkthrough method (Bell et al., 1991, 1992). Participants took part in the six scenarios shown in Figures 9.2, 9.3 and repeated for the reader's convenience from Section 9.2. Participants were free to discuss the scenarios and to ask for help or clarification from the administrator of the test. Participants were asked to act as active participants in the research and they helped to categorise issues that were raised. The categorisations that users were asked to use were questions (e.g. why does the loop do that?), problems (e.g. I don't understand what these lanes are for), suggestions (e.g. maybe the cone should be a different shape), and other observations (e.g. I like the fins). Finally, participants were asked if they could think of any other ways in which each scenario could be completed. This prompted a discussion on alternative routes in order to test understanding and to capture user expectations.

The six scenarios presented to the participants, with a brief overview of the results from each scenario, are described in Section 9.2. A more detailed reflection on the design issues follows in Section 9.3.

9.2 Walkthrough scenarios—results

9.2.1 Scenario 1: simple sequence

The participants were shown a video (Bellingham, 2020b) which introduced the handling of audio samples and the design of lanes and sequences. They were then asked to answer the following two questions and to complete a short task.

- Questions
 - What will happen when the sequence shown in Figure 9.4 runs?
 - How could you change the order of the sequence?
- Task
 - Using the whiteboard, update the sequence to play the marimba, then the bass hook, then the marimba again, then stop.

Findings from scenario 1

All groups answered the questions and completed the task without difficulty. Some participants asked questions which showed their desire for more complex functionality; one asked about synchronous playback, and another asked about the possibility of branching arrows. Participants can be seen completing the implementation task in Figures 9.5, 9.6.

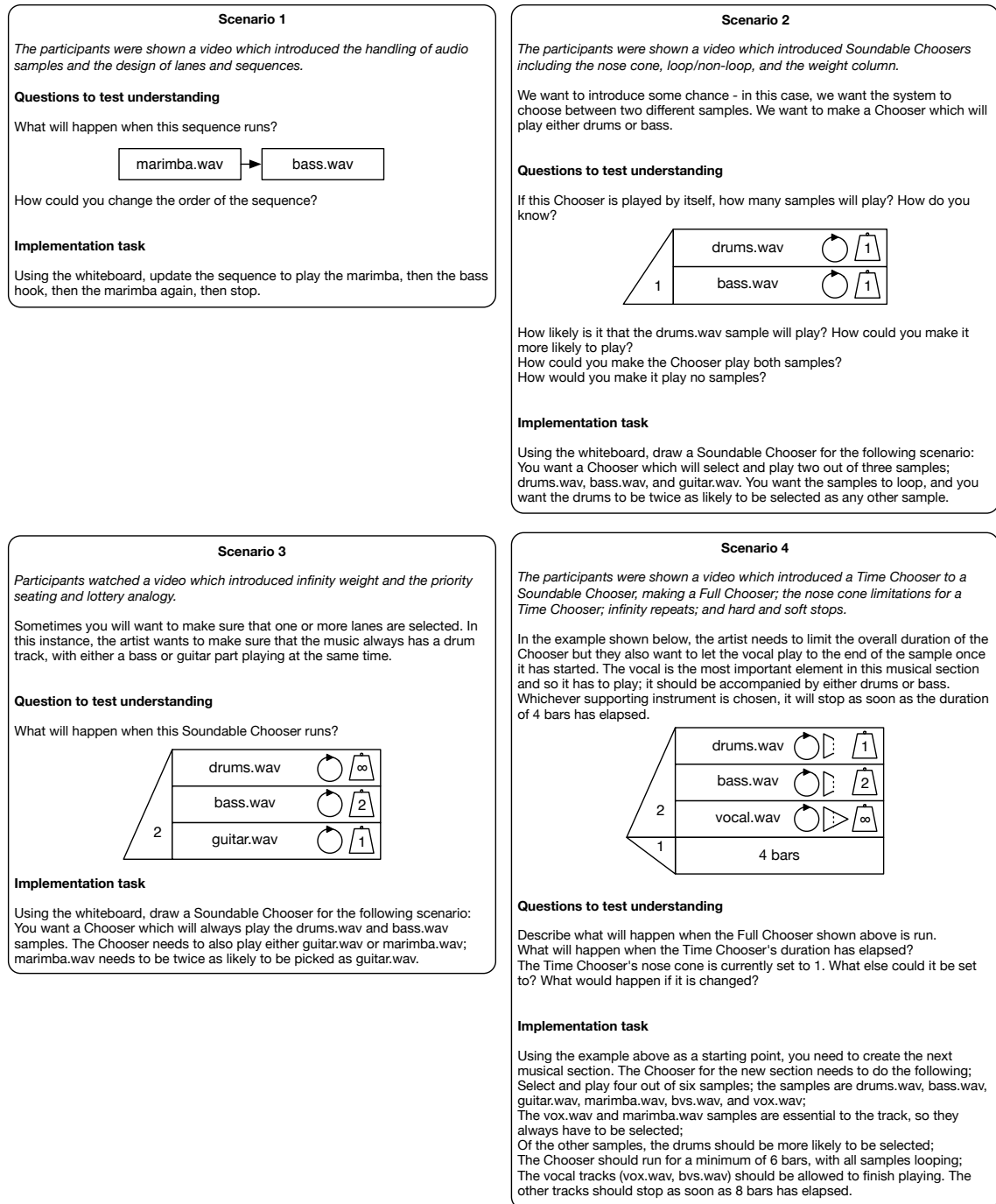
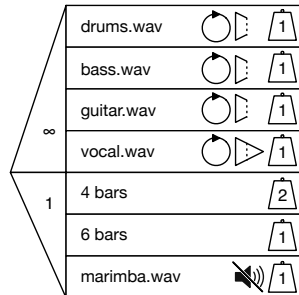


Figure 9.2: Scenarios 1–4 from the second user study.

Scenario 5

Participants watched a video which introduced infinity for the Soundable nose cone, and infinity as a duration in a Time Chooser; multiple lanes in a Time Chooser, including weighted choice and nose cone limitations; and soundable content in a Time Chooser, including mute/non-mute.

In the example shown below, the artist wants to hear all the lanes of the Soundable Chooser, and so uses `~` in the nose cone. All lanes are looping, and the Chooser will have a duration of 4 or 6 bars, or it will use the duration of the `marimba.wav` sample. A 4 bar duration is twice as likely as the other two options. Regardless of the duration, the vocal's soft stop means that it will play in its entirety before stopping. The other samples will stop immediately as soon as the duration has elapsed.



Questions to test understanding

Explain what will happen in the Time Chooser shown above.
What effect will the Time Chooser's selected duration have on the Soundable Chooser lanes?
How could we make the marimba.wav sample audible if it is selected?
In the Soundable Chooser, which lanes will be selected?
What effect will the Soundable nose cone setting have? How could you quickly change the Chooser to play two of the four samples?

Implementation task

Sketch out the previous example on the whiteboard, and make the following musical changes:

- Make it so 2 lanes are chosen; the vocal and one other instrument
- Make it so the Full Chooser has a duration of either 4 or 8 bars, with hard stops on all lanes
- Next, create a second Chooser and sequence it so it plays before this Chooser. The new Chooser should have the following characteristics:
 - Drums and bass should always play, and should loop until hard-stopped;
 - Either the vocal or backing vocals should play, looping until soft-stopped. The choice between them should be equal;
 - The Chooser should have a duration of 4 or 8 bars, with a 4 bar duration three times more likely to be chosen than an 8 bar duration.

Scenario 6

Using the whiteboard and the available samples, 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.

Final questions

Participants were asked the following questions at the end of the second user study:

Can you see anything this would be useful for?

Can you see any ways in which this is similar to other tools you have used?

Is there anything that is made easier by this system? Anything which was not possible made possible/hard and made easier?

Figure 9.3: Scenarios 5 and 6 from the second user study, plus the final questions.

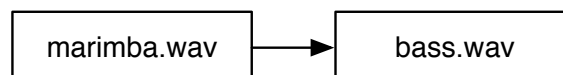


Figure 9.4: The simple sequence example shown to participants in scenario 1 as part of the second user study.



Figure 9.5: A participant creating a sequence on a whiteboard.

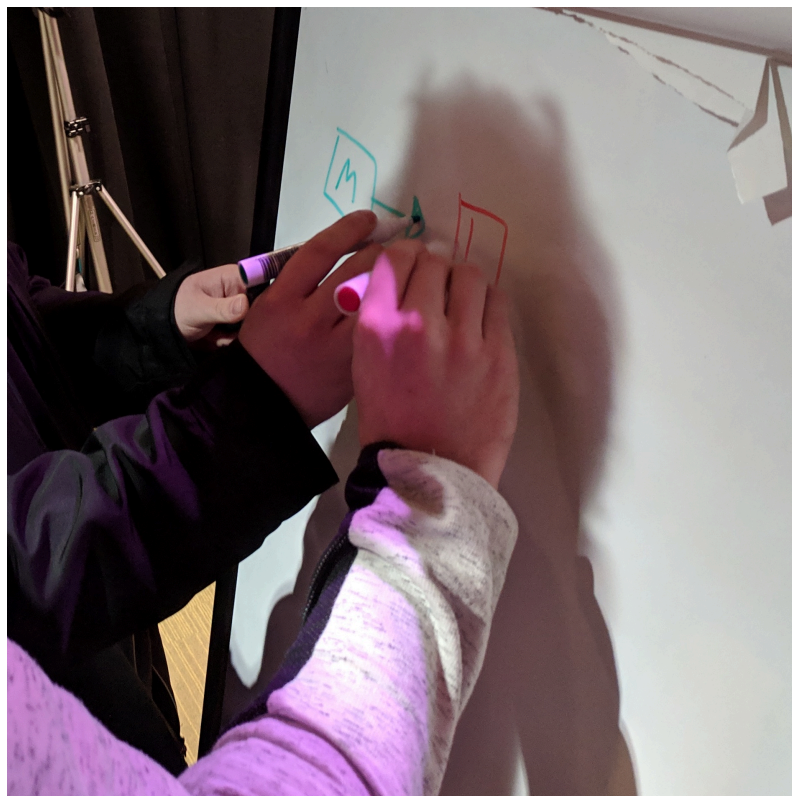


Figure 9.6: Two participants working collaboratively on a sequence.

9.2.2 Scenario 2: Soundable Choosers

The participants were shown a video (Bellingham, 2020b) which introduced Soundable Choosers including the nose cone, loop/non-loop, and the weight column. After a brief aural introduction to the scenario's focus on choosing between two different samples they were then asked to answer a series of questions to test their understanding, followed by a practical task.

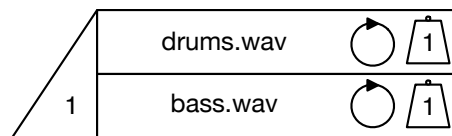


Figure 9.7: An image of a Soundable Chooser, used as part of scenario 2 in the second user study.

- Questions
 - If the Chooser in Figure 9.7 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?
- Task
 - Using the whiteboard, draw a Soundable Chooser for the following scenario:
 - You want a Chooser which will select and play two out of three samples; `drums.wav`, `bass.wav`, and `guitar.wav`. You want the samples to loop, and you want the drums to be twice as likely to be selected as any other sample.

Findings from scenario 2

This scenario introduced nondeterministic choice to the participants for the first time. The participant questionnaire showed that none had created algorithmic music before and so this was their introduction to the core concepts of algorithmic composition. As a result, participants sought the clarification of some of the key terms and behaviour of Choosers. One group used different analogies (lottery tickets, a coin toss) to support their understanding. Two groups needed to rewatch the section of the video which covered nose cone behaviour, and one rewatched the section on lane weight. One participant asked for clarification on whether the lanes represent sequence or synchronous playback. One participant asked if the nondeterministic choice is made afresh each time the Chooser is run. Interestingly, one participant used the word 'priority' when discussing lane weight; this word is introduced in Scenario 3. Importantly, all participants used this scenario to clarify their understanding of nondeterministic playback, and they did not request further clarification after this. One participant drew a parallel with nonlinear video game design, and another suggested that Choosers could be used for oscillator selection in a subtractive synthesiser design.

One participant asked how a lane could be excluded from selection, and their partner was able to correctly identify a solution; in this case, to set the lane weight to zero.

Participant 1: What about if you didn't want it to play one of the samples?

Participant 2: I suppose you could change this [pointing at the lane weight] to zero, could you?

Participant 1: Yeah, true, because obviously they're all going to play at the same time.

The tutorial video did not show the icon for a disengaged loop. A user in the first test identified the omission by asking for an example of the ‘loop off’ icon, and thereafter all subsequent tests showed participants a disengaged loop icon at the start of scenario 2.

9.2.3 Scenario 3: infinite weight

Participants watched a video (Bellingham, 2020b) which introduced infinite weight and the priority seating and lottery analogy. The participants were then presented with the following question and tasks:

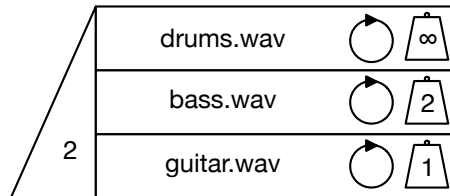


Figure 9.8: The example used in scenario 3 of the second user study, showing the use of infinite weight.

- Question
 - What will happen when the Soundable Chooser in Figure 9.8 runs?
- Tasks
 - Using the whiteboard, draw a Soundable Chooser for the following scenario:
 - * You want a Chooser which will always play the `drums.wav` and `bass.wav` samples. The Chooser needs to also play either `guitar.wav` or `marimba.wav`; `marimba.wav` needs to be twice as likely to be picked as `guitar.wav`.

Findings from scenario 3

Infinite weight was met with an ‘aha!’ from participants in two groups, who had asked about priority in the previous scenario. One of these participants was keen to understand the algorithm and asked specific questions about the limits of the weight parameter. One participant asked how a lane could be excluded from selection, and their partner was able to correctly answer (set the lane weight to zero). Interestingly, the aeroplane priority seating/lottery ticket analogy used in the video (see Figure 9.9) proved effective in communicating the key information insofar as no participants required further clarification on this point and all used the feature correctly.

One participant wanted to be able to specify a certain number of repeats:

The only thing I would say I would have it so it would loop for a certain amount of times – for example, the drums would be infinite, the bass would loop twice, and then once that’s gone it would select the guitar or marimba.

Although this issue had been identified and this feature adopted moving from the v1 design to the v2 design, this specific feature was not shown in this, the second user study.

Two participants in two different groups asked specifically about how the nondeterministic selection could be changed from ‘select sample and loop’ to ‘select sample, play once, then reselect’. This functionality is available via nested Choosers and, while nesting was not part of this study, the solution was discussed with these participants. In the ensuing discussions, all understood the proposed nesting mechanism.

The compositional task, referred to as an ‘implementation task’, was designed to test participant understanding of the interplay between the nose cone and lane weights. Some participants needed

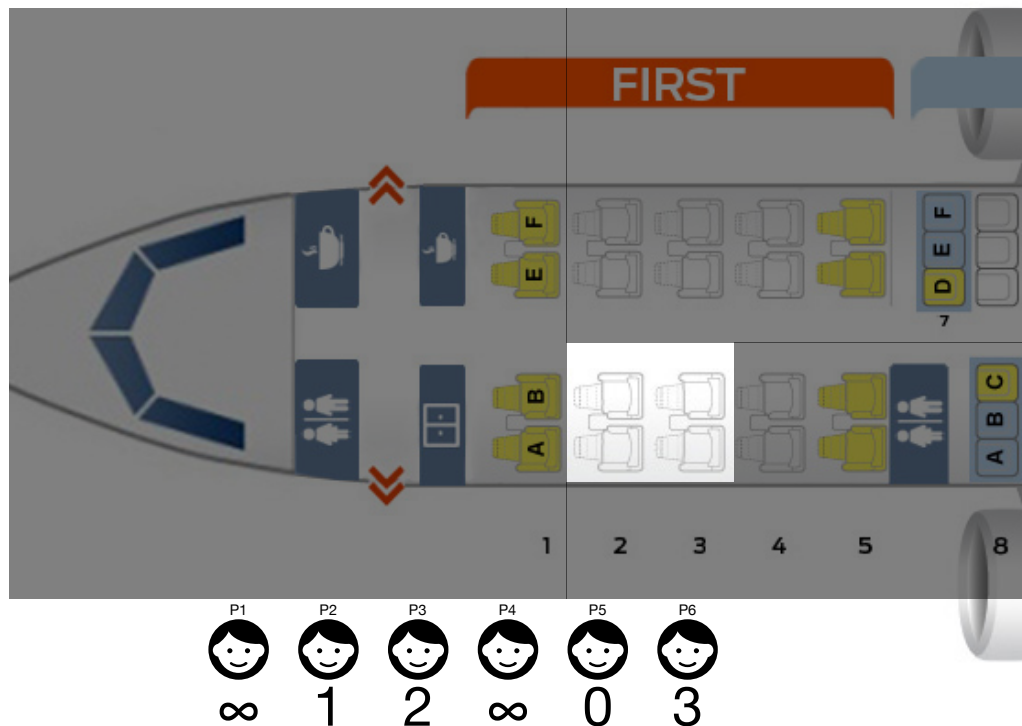


Figure 9.9: A still from the tutorial video showing the aeroplane priority seating/lottery ticket analogy. This image is a duplicate of Figure 8.7 and is presented here for the reader's convenience.

time, but all correctly completed the task. Two participants suggested that some elements of the design reminded them of commercial software (Ableton *Live*; 'Drummer' in *Logic Pro*) but that the commercial software did not allow for nondeterminism.

One participant reflected on nondeterministic playback:

It's quite nice that it goes back to the old way of ... hearing music different every time ... it's a nice element that's sometimes lost in the music we hear nowadays. It all has to be 'perfect' but perfect isn't necessarily the best.

All participant pairs were able, without prompting, to answer the questions and to provide a solution to the implementation task that played correctly when input into the Chooser engine by the facilitator.

9.2.4 Scenario 4: Full Chooser, including hard and soft stops

The participants were shown a video (Bellingham, 2020b) which introduced a Time Chooser to a Soundable Chooser, making a Full Chooser; the nose cone limitations for a Time Chooser; infinity repeats; and hard and soft stops. The participants were then asked to answer the following questions and to complete the following task:

- Questions
 - Describe what will happen when the Full Chooser in Figure 9.10 is run.
 - What will happen when the Time Chooser's duration has elapsed?
 - The Time Chooser's nose cone is currently set to 1. What else could it be set to? What would happen if it is
- Task

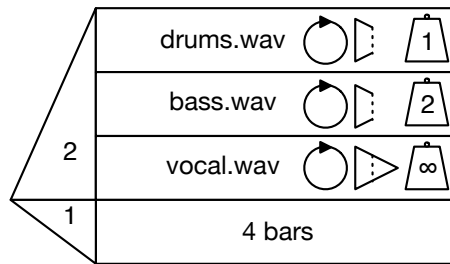


Figure 9.10: The Full Chooser used in scenario 4 of the second user study, introducing Time Choosers and both hard and soft stops.

- Using the Chooser shown in Figure 9.10 as a starting point, you need to create the next musical section. The Chooser for the new section needs to do the following;
- Select and play four out of six samples; the samples are `drums.wav`, `bass.wav`, `guitar.wav`, `marimba.wav`, `bvs.wav`, and `vox.wav`;
- The `vox.wav` and `marimba.wav` samples are essential to the track, so they always have to be selected;
- Of the other samples, the drums should be more likely to be selected;
- The Chooser should run for a minimum of 6 bars, with all samples looping;
- The vocal tracks (`vox.wav`, `bvs.wav`) should be allowed to finish playing. The other tracks should stop as soon as 8 bars has elapsed.

Findings from scenario 4

Participants in three of the six pairs had commented on the desirability of duration control in previous scenarios, and the introduction of Time Choosers in the tutorial video prompted some ‘aha!’ recognition. All participants were confident in using hard stops as they mirror the standard behaviour of DAW and sequencing software. Soft stops required discussion in most groups, and all groups found that listening to and editing Choosers containing soft stops was the most instructive way to consolidate their understanding. One pair seemed to find that conceptualising stops as polite or rude (using language from early in the development of Choosers v1; see Section 6.2.4), rather than soft or hard, allowed them to consider the functionality from a different perspective. The participants explained their assumption that a polite stop would ‘let you finish’, and a rude stop would ‘cut you off’.

By contrast with the first user study, in which a small number of participants found the role of Time Choosers confusing (previously discussed in Section 7.5.1), the collective understanding of the participants in this second user study suggests that the sequencing of concepts in which Time Choosers are introduced in their primary role (duration moderation) before their secondary role (rests) allowed users to better understand their function without the need for explicit additional explanation.

One participant requested a DAW-style repeat visualisation after being introduced to such an image in the tutorial video (shown in Figure 9.11).

The Time Chooser nose cone prompted discussion in two of the six participant pairs. The tutorial videos had not fully explained the functionality of the nose cone in this context and, as a result, two participants in two separate pairs required verbal confirmation of its purpose. This was largely caused by a lack of information and context, and could be addressed by improving the teaching order.

Four of the six groups said that they found the hard and soft stop icons, shown in Figure 9.11, confusing. The soft stop icon—a right-pointing triangle bisected by a vertical dotted line—had been designed to suggest continuing playback after a stop command. However, some participants felt that the triangle looked like a fade or an automation curve. The hard stop icon was misinterpreted

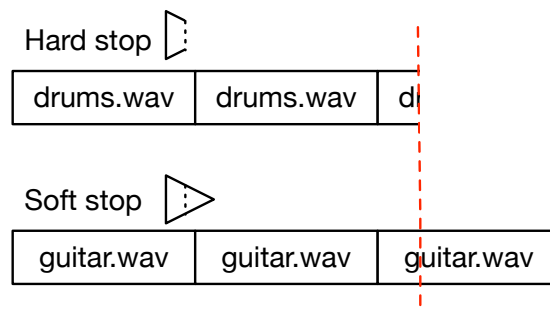


Figure 9.11: A still from the tutorial video showing a visualisation of hard and soft stop behaviour.

by two participants; one thought the sample's duration would be halved, and the other thought the duration set by the Time Chooser would be halved.

P1: I don't quite agree with the icon for the hard stop. It's a silly thing, but that's quite confusing. A square means stop, a triangle means play. I would change the hard stop icon.

P2: I would have thought it was a fade.

P1: It's like automation.

Three participants—all experienced DAW users—correctly identified soft stops as highly dependent on the duration of the sample and the global tempo and time signature.

How do you know how long the marimba sample is?

They requested metadata and global settings; both have featured in branches of the design but not in the versions chosen for evaluation. We discussed progressive disclosure in two groups, and these participants felt that some metadata could be shown as part of an 'expert mode'. One participant felt that the lack of visible metadata forced them to calculate the result which they felt was 'too much work'.

Several participants commented favourably on the results of a nondeterministic music system:

This kind of gives you a sense, if you're not sure what to go with, you put these two [samples] in and then it just messes about. You just grab something you like and then go for it. You can put them [samples] all together and say 'help me out'.

One participant requested multiple time lanes (which are in fact introduced in the next scenario). One user felt that Choosers could be used with a visual patching software design such as Reason. All participants were able to sketch a graphic program that exactly solved the musical problem posed (see Figure 9.12 for an example).

9.2.5 Scenario 5: Multi-lane Time Choosers

Participants watched a video (Bellingham, 2020b) which introduced infinity for the Soundable nose cone, and infinity as a duration in a Time Chooser; multiple lanes in a Time Chooser, including weighted choice and nose cone limitations; and soundable content in a Time Chooser, including mute/non-mute.

- Questions
 - Explain what will happen in the Time Chooser in Figure 9.13.

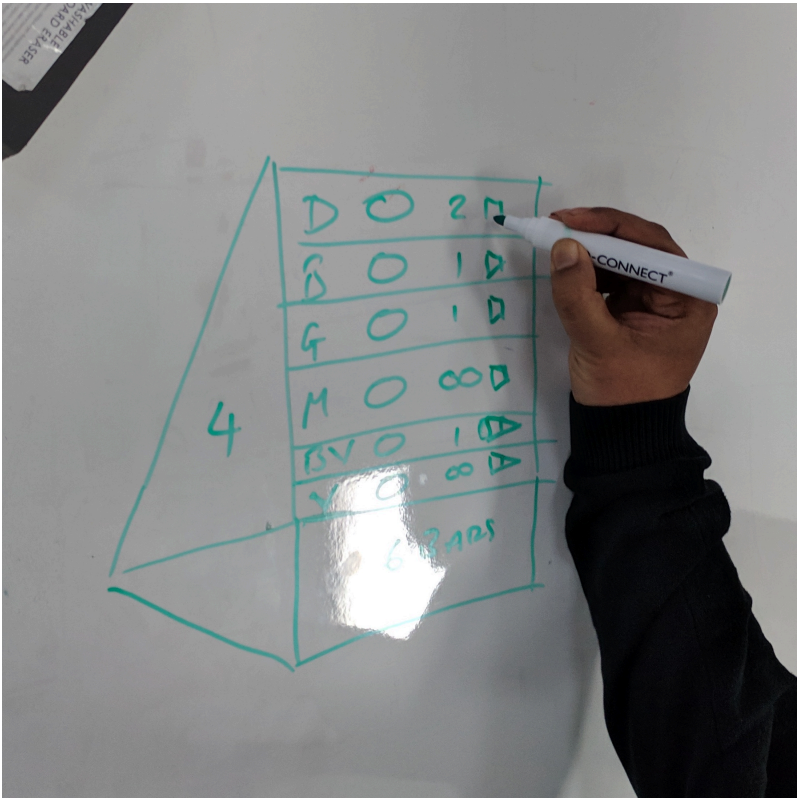


Figure 9.12: A participant working on a Full Chooser design using a whiteboard.

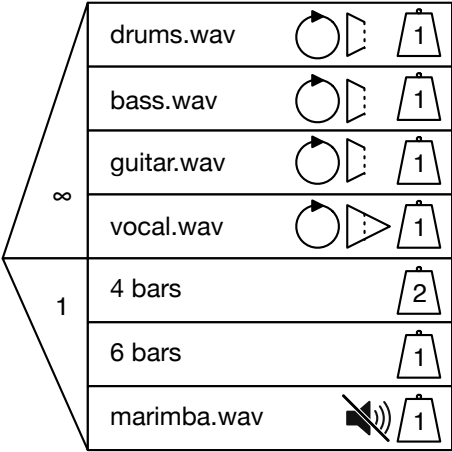


Figure 9.13: The example used in scenario 5 of the second user study, showing infinity in the nose cone of the Soundable Chooser, the user of a multi-lane Time Chooser, and soundable content in a Time lane.

- What effect will the Time Chooser’s selected duration have on the Soundable Chooser lanes?
 - How could we make the `marimba.wav` sample audible if it is selected?
 - In the Soundable Chooser, which lanes will be selected?
 - What effect will the Soundable nose cone setting have? How could you quickly change the Chooser to play two of the four samples?
- Tasks
 - Sketch out the previous example on the whiteboard, and make the following musical changes:
 - ✧ Make it so 2 lanes are chosen; the vocal and one other instrument;
 - ✧ Make it so the Full Chooser has a duration of either 4 or 8 bars, with hard stops on all lanes.
 - Next, create a second Chooser and sequence it so it plays before this Chooser. The new Chooser should have the following characteristics:
 - ✧ Drums and bass should always play, and should loop until hard-stopped;
 - ✧ Either the vocal or backing vocals should play, looping until soft-stopped. The choice between them should be equal;
 - ✧ The Chooser should have a duration of 4 or 8 bars, with a 4 bar duration three times more likely to be chosen than an 8 bar duration.

Findings from scenario 5

Scenario 5 introduces Choosers with multiple time lines. This more complex functionality of the Time Chooser required some discussion in a number of groups. Some participants found the Chooser shown in Figure 9.13 to be initially confusing. However, having sketched one or more graphic programs and compared the played result with their expectations, all participants were able to answer questions and carry out tasks correctly. On the basis of their performance in this and the upcoming scenario all users seemed to have formed an accurate mental model of the behaviour of multiple time lanes. Two groups asked questions to clarify the role of soundable content in a time lane.

All participants were able to quickly, confidently, and accurately answer the questions and complete the programming task. The facilitator made an error when entering a participant’s Chooser into SuperCollider, and the participant had sufficient understanding to recognise the error.

In one pair, one participant responded to the zero/one limitation of the Time Chooser nose cone by suggesting that it could be replaced by an on/off icon; the other participant in the pair disagreed and felt that it should remain as zero/one to match the behaviour of Soundable Choosers. In an unconnected comment, the same participant suggested that a Time Chooser be greyed out when the nose cone is set to zero and the Time Chooser is not active.

One participant requested the ability to name a Chooser for reuse. As noted in Chapter 8, this is indeed a core feature of the design, since, for example, it allows for the naming and subsequent referencing of a Chooser inside a box or another Chooser (see Section 6.1.6). Another user asked for a way to specify a given finite number of repetitions rather than a Boolean loop on/off switch, which was considered as a feature of Choosers v2 (see Section 8.1.3, detailed further in Section D.2.1.2 of Appendix D). However, these features were not presented as part of the present study.

Three participants, in three separate pairs, commented favourably on the readability and understandability of the notation.

It looks simple, but it explains a lot. For someone with no music experience, you’re going to automatically say ‘what’s 6 bars? What’s 4 bars?’ This can be taught to younger people who can get into music really quickly.

9.2.6 Scenario 6: Playground

Participants were given the following task:

- Using the whiteboard and the available samples, 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.

Findings from scenario 6

All participants were able to create a short piece of music. The process was seen as enjoyable and novel, and participants seemed particularly pleased to have completed a task that they could not have undertaken before the user test. Most, but not all, participants used the compositional scenario to test their understanding of Choosers rather than focussing on the structure of the music. Two groups worked top-down, creating an arrangement and then populating it. The other four groups worked bottom-up, creating individual Choosers and then assembling them into a sequence. One group created multiple copies of the same Chooser (imagining they were working on a computer which would make duplication simple) and introduced differences between them via zero lane weights and differing nose cone values. In this way they were able to quickly create a thematic arrangement.

9.2.7 Final formative questions

Participants were asked the following formative questions at the end of the second user study:

- Can you see anything this would be useful for?
- Can you see any ways in which this is similar to other tools you have used?
- Is there anything that is made easier by this system? Anything which was not possible made possible/hard and made easier?

Findings from the final formative questions

Can you see anything this would be useful for?

Most participants felt that Choosers could be used for musical inspiration. None were familiar with the area of computer-aided algorithmic composition (CAAC; see Section 2.6), but when asked all asserted that Choosers could create raw material for a composer to choose from. Participants used phrases such as ‘inspiration’ and ‘random arrangements’ to explain their desire for the software to create options for a human to select and curate.

Two participants suggested that Choosers could be used for nondeterministic performance. One participant considered the needs of a DJ, and the other was interested in a recording which has nonlinear elements, leading to the possibility of a different performance or a different piece each time.

Three participants identified novice users as benefitting from the design of Choosers. One participant suggested that it would be useful for novice music producers, and the second participant considered the top-down arrangement and sequencing capabilities to be useful for novice songwriters. The third participant suggested that the simplicity of Choosers would make it suitable as an educational tool for children.

Can you see any ways in which this is similar to other tools you have used?

One pair of participants thought Choosers would be useful as a plug-in for a DAW. They felt that an end user could experiment and then output a MIDI file from the Chooser plug-in into the host DAW for further editing.

Two participants considered the easy arrangement changes afforded by top-down composition to be similar in approach to both Ableton *Live* and *Superior Drummer*. Both participants noted that only Choosers allowed for nondeterministic output.

The focus on the needs of a novice user led five participants to suggest three pieces of software which are designed for quick and easy use—Apple’s *GarageBand*, *FL Studio*, and Logic Pro’s *Drummer* functionality.

Two participants suggested that the ability to program, including referencing named Choosers, drew parallels to both *Max for Live* and *Massive* by Native Instruments.

Is there anything that is made easier by this system? Anything which was not possible made possible/hard and made easier?

None of the participants had previously attempted any algorithmic music-making and so all non-deterministic elements seemed novel to them.

I haven’t come across any algorithmic software before, so everything is new!

Several participants felt that the design seemed complex at first but, once learned, it is easily understandable and readable. The stop functionality was seen as offering musical opportunities not easily accessed using other tools.

[Stops are] like cutting without actually cutting it.

The ability to create and edit an arrangement via a top-down approach was seen as musically useful, as was the ability to introduce nondeterminism across a range of musically-relevant parameters.

While it was not a specific question, some participants were interested in other potential uses for Choosers, such as for oscillator control in a synthesiser, or as a nondeterministic way to augment visual patching software.

9.3 Reflection/discussion

The findings from the second user study, outlined above, highlight a range of issues used to help refine the next iteration of Choosers, as described in the next chapter.

9.3.1 Surfacing musical moves

9.3.1.1 Choosers as introduction to algorithmic music

As in the first user study, and previously discussed in Section 7.5.1, the participants in the second user study were not previously familiar with algorithmic composition. The tasks they undertook in the study were the first time they had engaged with nondeterministic music and, as such, it was the first and only tool that they had used to create nondeterministic music. Rather than running a separate introduction to nondeterministic music before the user tests, the user study was specifically designed to introduce key concepts in a musically-meaningful order. The design of Choosers proved to be effective in allowing novice programmers to explore weighted nondeterministic choice. The behaviour and comments of the participants suggested that a quick iterative cycle—making a speculative change and immediately hearing the result—was a powerful learning tool. The impact of immediate aural feedback has links to Leman’s Action-Reaction cycle (Leman, 2008) and Tanimoto’s ‘liveness’ (Tanimoto, 1990), both introduced in Chapter 2. Almost all conceptual problems encountered by the participants using Choosers were addressed by hearing examples and allowing participants to explore and experiment. This mirrors Nash’s finding that an accelerated edit-audition cycle (shortening the feedback cycle by editing short excerpts of music) leads to a higher

level of liveness when compared with the linear timeline interactions typically used in many DAWs (Nash, 2012, 2015).

As outlined in Section 8.1, the design of Choosers for the second user study was changed in order to allow easier access to a range of musical moves or transformations. The uses of infinity as a lane weight ('make this always play'), a nose cone value ('play all legal lanes'), and a duration ('play forever') were all understood and correctly used in the creation of short pieces of music. Infinity as lane weight replaced the 'A' ('always play') notation from the first user test, and represents the only way in which a lane can be set to always be selected. However, the use of infinity as both nose cone and duration values provides the user with multiple ways to achieve a desired result. Infinity as a nose cone value is a convenience designed to reduce the mental work asked of the user. Infinity as a duration offers a third way of providing infinite playback (alongside a Soundable Chooser with no Time Chooser, and a Full Chooser with an inactive Time Chooser).

Infinity provided affordances in all three circumstances and there were minimal instances of misunderstanding seen in behaviour, questions, or comments. The use of infinity as a metaphor is discussed in detail in section Section 9.3.2.

All participants had experience in using music creation software and so a hard stop ('stop immediately') required no further explanation as it is the standard behaviour of such tools. A soft stop ('stop when you have finished') required more explanation in this test as it did in the first user study (see Section 7.5.1). However, it appears that the enhanced documentation and use of metaphors, discussed in Section 9.3.2, may have reduced the number of misunderstandings regarding stop behaviour. Changing stop behaviour from global to a Chooser to a per-lane setting has significantly widened the musical opportunities with no evidence of harming understanding. Indeed, all participants expected to be able to control the stop behaviour of each lane.

Several participants asked about more advanced functionality; in some cases they asked about functionality which would be introduced later in the user test (such as the addition of infinite weight, or duration control), and in other cases their questions required more advanced functionality which was out-of-scope of the test (such as nested Choosers for re-choice, or the referencing of named Choosers). In all cases the participants were effectively working in the same conceptual space as Choosers; their questions were based on their understanding of the notation and they were able to intuit musical nondeterministic possibilities. Importantly, all requested functionality existed either in the user test scenarios or in the larger design of Choosers. The use of nested Choosers is an example of progressive disclosure, with the most common musical move available on the surface and a different behaviour easily available. For example, a single Soundable Chooser can select one audio file and play it repeatedly. A different behaviour—select a file, play it once, then reselect—is available by nesting a Soundable Chooser with lanes set to not loop inside the lane of another Soundable Chooser which is set to loop. The user tests suggest that this design is effective.

The use of soundable content in a Time Chooser's lane was accepted and quickly understood. The use of a widely used icon for play/mute meant that participants understood its function without the need for explanation.

A small number of the more musically sophisticated participants were aware of the need for global control over key parameters (tempo, time signature) as well as file-specific metadata (in this case, sample duration).

9.3.2 Identifying and implementing suitable metaphors for non-programmers

Metaphor is widely used in music production software, and the findings from the first user study (see Section 7.5.4) led to greater use of metaphor in the documentation/teaching stage for Choosers v2 in order to better communicate concepts which were new to the participants.

As outlined in Section 9.3.1, infinity was introduced to allow users to easily issue three commands: always play a given lane (infinite weight); play all available lanes (infinity nose cone); and play forever (infinity as duration). The use of zero to make a lane unavailable (zero weight) or to deactivate a Chooser (zero nose cone—'pick none') was introduced after infinite weight was intro-

duced. The tutorial video which used a plane seating metaphor showed infinite weight as ‘priority seating’, and later equated a weight of zero with travellers without a ticket. This proved to be an effective metaphor, with all participants effectively fully understanding not only this application of zero but also the use of zero elsewhere in the notation. When compared to the design of Choosers v1, which used ‘A’ (‘always play’) instead of infinite weight (see Sections 6.2.2, 8.1.1.1), the use of infinity weight in Choosers v2 might be considered to have better parsimony, thematic unity, and consistency.

The tutorial video which introduced hard and soft stops in the second user study included a visualisation of the effect of hard and soft stops (see Figure 9.11). This proved effective, with fewer questions and clarifications required when compared with the first user study. Within some participant groups we discussed a car race metaphor when designing the stop icons—a car stopping immediately, or a car which is flagged and then comes in to pit on the next lap.

The second user study showed a reduction in participant feedback on non-essential design elements (such as icon design) and an increase in practical use, experimentation, and requests for more advanced functionality. It is possible that this is due, in part, to more efficient and effective learning due to the use of appropriate metaphors to communicate concepts which are novel to the participants. It is also likely that the design of Choosers v2 is more consistent than the v1 design used in the first user study.

9.3.3 User expectations

As outlined in Section 2.5, users bring frames of reference with them and apply them to a new interface. In this instance, the participants were not only familiar with music playback devices following a phonography format (Section 2.3), they were also familiar with DAW design (Section 2.4.3). As a result, as outlined in Section 9.3.1, they understood a hard stop immediately, as that is the default behaviour of nearly all linear playback and creation tools from *Spotify* to *Pro Tools*. The same frames of reference led to some confusion over the updated icon designs, as the hard and soft stop icons reminded some participants of DAW functions. In particular, the triangular stop icons were reminiscent to some of automation curves, and the dotted line in the stop icons hints at editing tools on some platforms. The design of some icons had unwittingly moved into the DAW icon space and, given that Choosers are a music creation tool, users will understandably try to apply prior learning to a new interface. A related expectation was that Choosers would run as a plug-in with a DAW host, rather than being an alternative tool.

9.3.4 Possible improvements for design and implementation

A small number of participants requested some sort of visualisation, such as a trace or a map. Another visualisation approach would be for Choosers and lanes to highlight or flash to show when they are active; this could be useful when understanding the impact of compositional changes. Related to participant comments on computer-aided algorithmic composition, the provision of a post-run trace would aid understanding, show the ‘choices’ made by the software, and allow for the reuse of material. One participant suggested that a deactivated Time Chooser should be greyed out. One user requested greater control over the loop behaviour. Specifically, they wanted to be able to specify the number of repeats, rather than use a Boolean on/off loop control. While this feature was not presented to participants for this study it is a feature of Choosers v2, and it offers a consistent use of integers inside icons to control significant parameters (e.g. nose cone, lane weight).

A small number of participants seemed to find the Time Chooser’s nose cone confusing when introduced. The behaviour and comments of the users suggested that this was because there was one lane and so only one available option, making the nose cone seem redundant as the other options (nose cone of zero; multiple time lanes) had not yet been introduced. This appears to be a teaching order issue, and could be rectified by briefly introducing the uses for the nose cone when the

Time Chooser is introduced. However, it should be noted that this element of the design requires explanation.

Several of the participant pairs appeared to fully understand the functionality of Choosers and wanted to know if and how they could achieve more advanced behaviour. An example of this was one pair who wanted to run a Chooser with an infinite duration that would keep selecting samples once they had finished, rather than selecting and looping one sample. In effect, the participants had understood the design and were working in the same conceptual space; they wanted to combine and leverage the same tools to create more complex musical fragments, and their expectation was that they could do so. This suggests that the design principle of the use of small tools and combinatorial power has been understood. While some participants in the first user study wanted to explore more advanced behaviour, most discussion was focussed on icon design and matching the behaviour of the DAW software with which the participants were familiar. The participants in the second user study had better understood the concept, were happier to accept it as distinct from DAW design, and were better able to create more complex nondeterministic musical structures.

The one participant with the most experience as both a musician and a DAW user was the person who felt that Choosers did not offer him the tools he wanted to make music. He is a professional Bhangra keyboard player. He was happy to accept Choosers as part of a CAAC-based compositional process, but he strongly felt that a final piece should be linear/deterministic and so would not want to use Choosers in his own work.

9.4 Conclusions from the second user study

Section 7.4 outlined the findings from the first user study (Chapter 7) and Chapter 8 documented the development of Choosers v2. The changes made to both the notation and supporting documentation are explored in detail in Chapter 8, and summarised in Table 8.1.

This chapter has presented a participant programming walkthrough evaluation of Choosers v2. Overall, participants worked quickly and accurately, and their work showed both a high level of understanding and an ability to create musically interesting pieces in a short time frame. This suggests that the changes to both the notation and documentation had been effective in better presenting the system and conveying the underlying concepts.

The introduction of infinity was effective as it is in the same conceptual space as other numeric values and allowed participants access to the maximum value. Alongside this, the ‘priority seating/lottery’ metaphor used when introducing infinity as a weight (see Section 8.1.1.1) proved to be an unexpectedly useful tool. The metaphor clearly communicates the levels of priority between the nose cone and lane weights, with zero weight equated to the lack of a lottery ticket and therefore no chance of selection. Once this was stated the participants were able to navigate various other instances of zero (such as zero in a nose cone) with relatively few clarifications required. Other than infinity as a weight, infinity was also introduced as a legal setting for the nose cone in a Soundable Chooser and for a Time Chooser duration. Both these uses are conveniences, allowing users another way to achieve a given goal. This use of consistent design increases the flexibility of the system, reinforces prior learning, and allows users to develop a preferred way of working. Participants quickly understood these uses of infinity operationally and were able to use them without issue. Note that the use of zero in the nose cone should be introduced before Time Choosers so that the functionality of the nose cone is understood.

The design change in stop behaviour—from global per Chooser to a per-lane setting—significantly increased the musical possibilities available to the user. Interestingly, this change did not seem to harm the participant’s understanding; indeed, the change seemed to match user expectations once they understood the distinction between hard and soft stops. The removal of internal columns in Choosers v2, and the refreshed icon designs, proved to be positive changes. The move away from columns improved the progressive disclosure of icons as it removed the need to fill each column, and the play/mute icon used for soundable content in a Time Chooser was

quickly understood. The icon designs were mostly successful, but the stop icons included two design elements which clashed with the visual language of DAWs. The triangular element of the soft stop icon reminded some participants of an automation fade, and the dotted line was similar to some edit tool icons. Some participants requested DAW-style visualisations.

The improvements made to the documentation were effective in better communicating new concepts and techniques, and the scenario order worked well in introducing relevant content and then making it accessible to the participants. The ability for users to iteratively build, experiment, audition, and make changes was particularly powerful. Once participants were confident in navigating the conceptual space, a number of users applied their understanding to both correctly guess more advanced functionality (such as nesting) and to request advanced functionality available via combinatory usage (such as choose once and loop vs. play once and re-choose). This suggests that participants understood the system and felt confident in leveraging the consistency of the design to find new combinations and interactions.

Chapter 10

Design exploration in preparation for the final user study

This chapter presents the development of Choosers v3, informed by a combination of speculative design changes and the findings from the second user study presented in the previous chapter. Next, the chapter presents two annotated musical examples to test the features implemented in Choosers v3, to be used in the final user study.

10.1 Methodological considerations

This chapter is a design study in preparation for the third and final user study (to be presented in the succeeding chapter). The final user study will be different from the previous ones since it takes the form of interviews with musical experts; in keeping with this forthcoming development, the present chapter also has a different emphasis. In particular, only the first part of the present chapter, Section 10.2, focuses on design iterations resulting from the previous user study. This material is supplemented by a design study, to be shown in Section 10.3, exploring ways of extending the expressivity of Choosers. Finally, two fully-notated examples of more complex musical pieces will be presented in Section 10.4. It is important to note that these musical examples do not make use of the speculative extensions introduced in this chapter; only currently implemented features of Choosers are used.

10.2 Design iterations led by issues identified in user testing

The following design iterations were motivated by findings from the second user study, previously presented in Section 9.3 of Chapter 9 and summarised in Section 9.4. An example of the final design of Choosers v3, showing the use of a more granular loop control (Section 10.2.1), a blank lane (Section 10.3.1), and new stop icons (Section 10.2.2), is shown in Figure 10.1.

10.2.1 Increasing the granularity of the loop control

While testing Choosers v2, one participant expressed their desire to individually specify the number of repeats per lane. The design of the Choosers v2 loop icon had included a consideration of this exact issue via the use of integers inside the loop icon to denote the number of times the material hosted in the lane will play (see Section D.2.1.2 of Appendix D). However, this was not implemented in the final design of Choosers v2, which instead offered a simple Boolean on/off loop setting due to its unambiguous legibility.

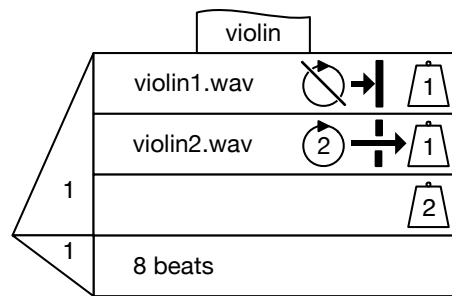


Figure 10.1: An example of the final design of Choosers v3, showing the use of an integer inside the loop icon to specify the number of times the material in a lane should be played, a blank lane to allow for a musical rest for the Time Chooser’s selected duration (in this case, 8 beats), and updated hard (top) and soft (bottom) stop icons.

The design of Choosers v3 reconsidered the option of specifying the number of times the contents of a lane will play, with the following motivations and limitations. The use of integers inside icons is consistent across the design of Choosers (e.g. integers inside the nose cone and weight icons). There are clear circumstances in which the user may want to specify the number of repeats of the content of a lane, and circumstances in which a simple ‘play once’ or ‘play until you are stopped’ are too inexpressive. The existing loop on/off option can remain the default, with the new behaviour (specifying the number of repeats) used as an option, as shown in Figure 10.2. So, the use of numbers in the loop icon in Choosers v3 adds a new capability without introducing visual clutter when it is not used, thus making use of progressive disclosure. Optionally, the user may use 0 and ∞ in the loop icon. ‘Zero loops’ means that the lane can be selected but the contents of the lane cannot play, resulting in silence; this makes the lane functionally equivalent to a blank lane, a new addition to the notation to be introduced below in Section 10.3.1. ‘ ∞ loops’ is equivalent to the Boolean ‘loop on’. While seemingly redundant, these options reduce viscosity if the user is changing to or from an existing value in the loop icon, and provide consistency with the use of 0 and ∞ elsewhere (see Section 8.1.1). An example of the finite looping control adopted in Choosers v3, including 0 and ∞ , is shown in Figure 10.2.

10.2.2 New stop icons

The improved tutorial materials for hard and soft stops, introduced in Section 8.2.3 and used in the second user study, seemed to be effective in use (see Section 9.4). However, the stop icons used in both Choosers v1 and v2 proved problematic for participants in both the first (Section 7.5.4) and second (Section 9.3.3) user studies. As a result, a series of candidate icons, shown in Figure D.25 of Appendix D, were drafted with the aim of more effectively communicating stop behaviour using a left-to-right depiction of time familiar to users of DAW software. In these draft designs, playback is shown as a left-to-right arrow, with the stop message from the Time Chooser depicted as a vertical line. The hard stop icon shows the arrow stopping at the line—playback stops as soon as the stop message is received. In contrast, the soft stop icon shows the arrow continuing past the line. The icons were designed to be legible at small sizes; aligning the vertical ‘stop’ line produces a clear distinction between hard and soft stops. The final designs for hard and soft stops used in Choosers v3 can be seen in Figure 10.1, with the candidate designs shown in Figure D.25 of Appendix D.

10.2.3 Metadata visibility in lanes

As previously outlined in Section 9.3.1, a small number of participants in the second user study requested easy access to the metadata of Soundable Chooser lane contents (e.g. sample duration in bars and/or beats). The calculation of musical duration also requires access to global tempo and time signature information. For Choosers v3, an overlay can be optionally added to show the

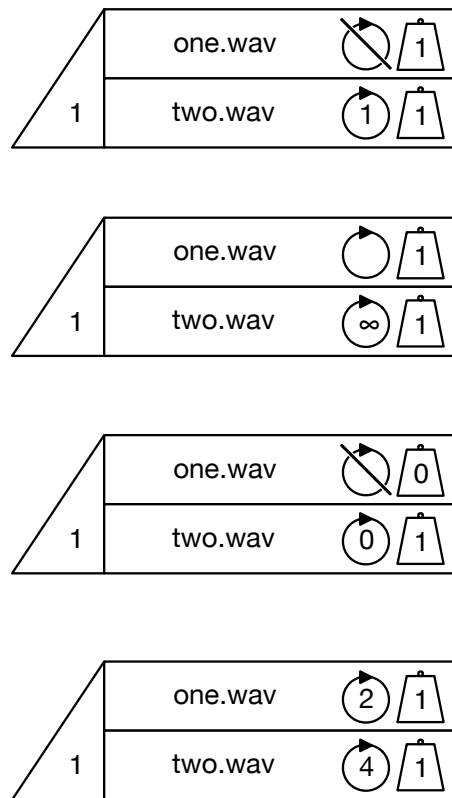


Figure 10.2: Finite looping control in Choosers v3. The uppermost Chooser shows 'loop off' in the first lane and 'play once' in the second lane; these are functionally identical. The second Chooser shows 'loop on' in the first lane and 'loop infinitely' in the second lane; these too are functionally identical. The third Chooser shows examples of both 'zero weight' (first lane) and 'zero loops' (second lane). The first lane's zero weight means that the lane cannot be selected for playback. While the second lane's positive weight means that it can be selected, the zero loop setting means that the lane contents will not play and the lane will act as a blank lane (to be introduced in Section 10.3.1). Finally, the lowermost Chooser shows an example of the new functionality that the addition allows; if selected, the top lane will play twice before stopping, and the bottom lane will play four times before stopping.

duration of file contained in a lane. The unit used for duration can be musical (i.e. bar, beat), in which case it would be calculated using the sample length and global tempo and time signature settings. Alternatively, the duration of an audio file can be shown in samples. The duration unit of a Soundable Chooser will default to bars unless a smaller unit allows for an integer duration (e.g. 3 beats rather than 0.75 bars). In a Full Chooser, the duration unit will update to match the duration units used in the Time Chooser, if applicable. If there is no Time Chooser, or if there are multiple duration units used in the Time Chooser, or if the user wishes to override the default, the metadata menu shown in Figure 10.5 could be used to set the desired duration unit.

Figure 10.3 shows three candidate variations of visible sample length metadata in Soundable Chooser lanes. A sketch of the global control over tempo and time signature, required to calculate musical duration, is shown in Figure 10.4.

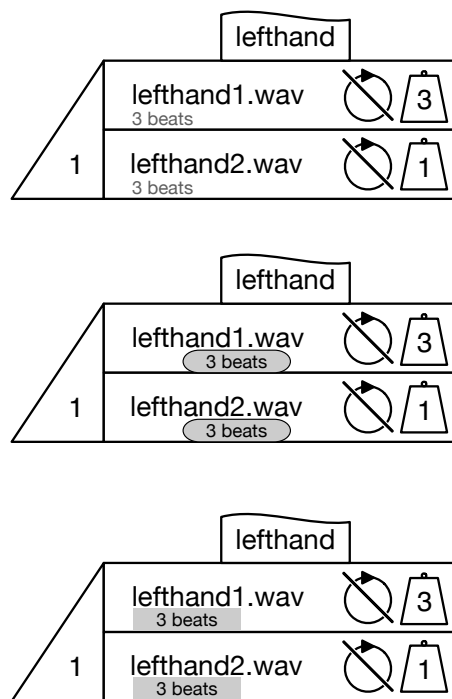


Figure 10.3: Three alternative visualisations showing the musical duration of each lane in a Soundable Chooser.

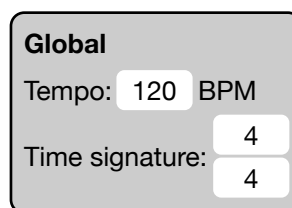


Figure 10.4: Global controls for tempo and time signature.

An alternative or additional way of showing and editing duration information in a Chooser lane would be to add a duration display. Figure 10.5 shows a test sketch in which duration is added to the upper left of the menu. The user is able to override the default duration unit if they wish: the duration unit defaults to the unit used in Time Chooser lanes, if present; if there is no Time Chooser it defaults to whichever duration unit allows for an integer value (i.e. it will default to 2 beats rather than 0.5 bars). Additionally, Figure 10.5 shows the addition of a 'percussion' toggle. This control mirrors the use of 'one-shot' or 'percussion' sample playback in samplers, as outlined in Section 6.2.3. When selected, this parameter results in the playback of the entire sample regardless

of input.

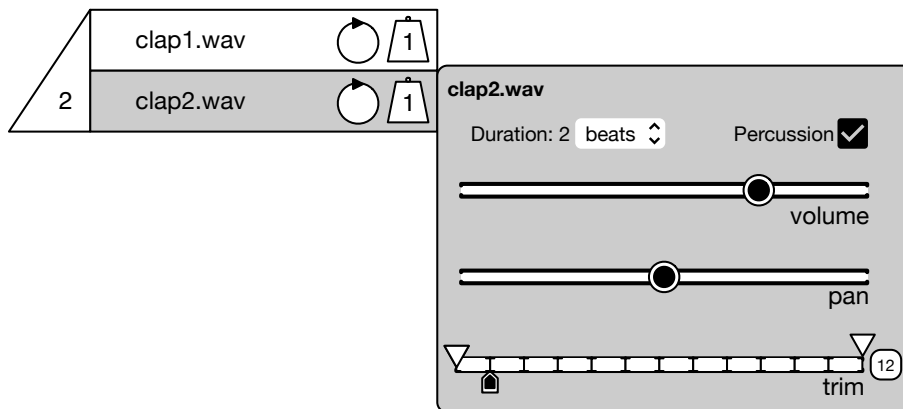


Figure 10.5: An alternative visualisation to that shown in Figure 10.3: here the user can choose the unit used to display duration, and can opt to play the entire sample ('percussion'). This design is a development of the panel shown in Figure D.36 of Appendix D.

10.2.4 Visualisation and post-run trace

The findings from the second user study (see Section 9.3.4) included a suggestion to introduce visualisation via highlighting/flashing chosen elements. The same findings outlined the usefulness of a post-run trace, both to help the user understand their work and to enable a particularly pleasing result to be used in the user's future work. While both visualisation and traces offer benefits they are not required for the third user study, and so have not been adopted as part of Choosers v3.

10.3 Speculative design changes

Section 10.2 considered issues identified as part of the second user study (previously outlined in Chapter 9). As well as addressing user issues, the design process also included a process of reflection, a consideration of the underlying design issues, and a re-examination of some assumptions made early in the design of Choosers. This led to a series of speculative design changes, outlined below, to explore ways of making Choosers more expressive.

10.3.1 Blank lane

The design principles pursued in Choosers include consistency, simplicity and parsimony. However, these principles do not mandate that the notation and semantics should be designed in such a way that there is only ever one single way to achieve any given outcome. Indeed, alternative routes to outcomes can facilitate progressive disclosure, help support user diversity, and avoid viscosity in a wide range of situations. With this in mind, Choosers v3 introduces the concept of a blank lane. Put simply, a blank lane allows 'nothing' to be a valid selection. Consider the following simple musical command: 'either play this sample or skip it and move on'. This musical move can already be achieved by using a variable in the nose cone of a Soundable Chooser, as shown in Figure 10.6. However, a blank lane, shown in Figure 10.7, is an alternative to the use of a Variable Chooser in Figure 10.6.

Figure 10.7 shows a Soundable Chooser which will select one of two lanes; the upper lane contains an audio file, *melody1.wav*, and the lower lane is blank. There is an equal chance of the selection of either lane. The upper lane will result in playback; the lower lane will result in no output and, if used as part of a sequence, the Chooser would be skipped. The same example is expanded in Fig-

ure 10.8, which adds repeats via the 'x8' multiplier. This will result in eight repeats of the Soundable Chooser, each of which will result in an equal likelihood of either playback or no playback.

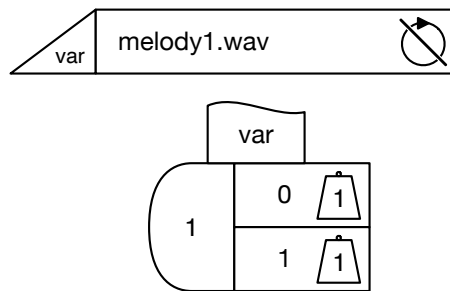


Figure 10.6: A single-lane Soundable Chooser with the output of a Variable Chooser used as the nose cone value. This will result in the `melody1.wav` sample being played if the output of variable `var` is a 1. If `var` outputs a 0 then the sample will not play. Note that the nose cone value of a Variable Chooser is always 1 and cannot be changed (see Section 6.1.7 in Chapter 6).

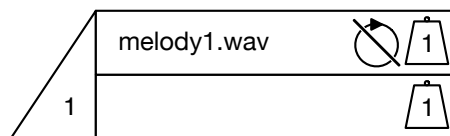


Figure 10.7: An alternative to Figure 10.6, this time using a blank lane. There is an equal chance of the selection of either lane. The upper lane will result in playback; the lower lane will result in no output and, if used as part of a sequence, the Chooser would be skipped.

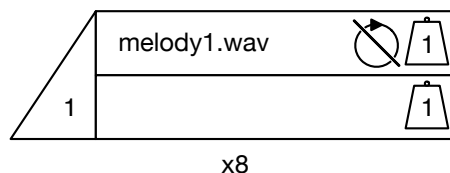


Figure 10.8: This example is identical to Figure 10.7 with the addition of repeats via the 'x8' multiplier. This will result in 8 repeats, each of which will result in an equal likelihood of either playback or no playback.

Blank lanes can be used in either Soundable and/or Time Choosers, as described in Figure 10.12. The following examples illustrate some uses for blank lanes.

Figure 10.9 shows the addition of a Time Chooser to the Soundable Chooser from Figure 10.8. The Time Chooser will moderate duration to a length of 4 bars. If the `melody1.wav` lane is selected it will have a hard stop; it does not make sense for the blank lane to exhibit stop behaviour as there is no file to stop, so the stop icon is not shown. If the sample is selected it will play for 4 bars; if the blank lane is selected it will produce a 4 bar rest. The Chooser will play eight times, each time selecting afresh. A blank lane can also be added to a Time Chooser, as shown in Figure 10.10. If the blank lane is selected, the Full Chooser will run as though it were a Soundable Chooser without a Time Chooser to control its duration. If the `melody1.wav` sample is longer than 4 bars, and if the blank Time Chooser lane is selected, then the sample will play in its entirety once before stopping. If the lane containing the sample were set to loop, and the blank Time Chooser lane was selected, the Chooser would play infinitely. The user can opt to include blank lanes in both Soundable and Time Choosers at once, as shown in Figure 10.11. If the Time Chooser selects the '4 bars' duration then the Chooser will play either the sample or silence for 4 bars. If the Time Chooser selects the blank lane the Chooser will play the sample for its full duration or, if both blank lanes are selected, the Chooser will be skipped.

Given that the above sketches suggest that blank lanes do not harm existing functionality while improving progressive disclosure and offering an alternative route to a given outcome, the blank lane design was adopted as part of Choosers v3.



Figure 10.9: A Full Chooser with a blank lane in the Soundable Chooser.

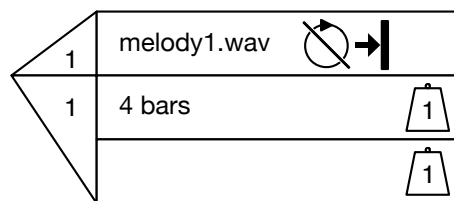


Figure 10.10: A Full Chooser with a blank lane in the Time Chooser.

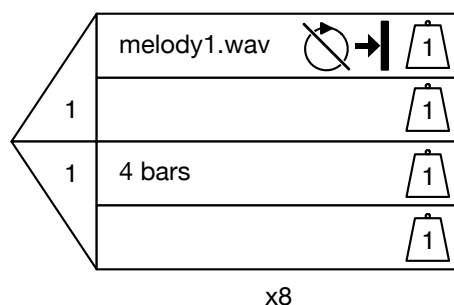


Figure 10.11: A Full Chooser with blank lanes in both Soundable and Time Choosers.

10.4 Musical examples

The next chapter, Chapter 11, will present the results of a series of interviews with domain specialists in the areas of music composition, music production, and music computing education. In order to explore the expressivity of Choosers v3, and to provide musical examples for the interviewees to manipulate, two original musical examples will now be presented. The examples are fully notated, and playable audio files of each example are available online (Bellingham, 2020a). These examples are not intended to test the expressivity of Choosers, but rather to support the discussions with experts outlined in the next chapter. Both examples are written without the speculative extensions introduced in this chapter.

10.4.1 Piano and violin example

When there is one or more levels of nesting, considerably more complex structures and modifications become possible. The next example, shown in Figure 10.14, 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

	Soundable Chooser lane with content	Blank Soundable Chooser lane
Time Chooser lane with duration	Playback with moderated duration The Soundable Chooser content plays and will be moderated by the duration set in the Time Chooser lane if it is longer than the Time Chooser duration or if the Soundable Chooser lane is set to loop.	Rest The chosen duration elapses with no playback, resulting in a musical rest.
Blank Time Chooser lane	Unbounded playback The Soundable Chooser content plays and is not moderated by the Time Chooser; the Soundable Chooser runs as though there is no Time Chooser.	Skip Neither Soundable nor Time Chooser content is selected, so the Chooser is skipped in its entirety.

Figure 10.12: The musical output resulting from the selection of blank lanes in Soundable and/or Time Choosers.

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. Focussing on lefthand, 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 10.14 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.

A central aspect of music is timing. By adding Time Choosers to the nested structure shown in Figure 10.14, time can be manipulated at different levels; an example is shown in Figure 10.15. Here it can be seen 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.

As shown, Figure 10.15 builds on Figure 10.14 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;

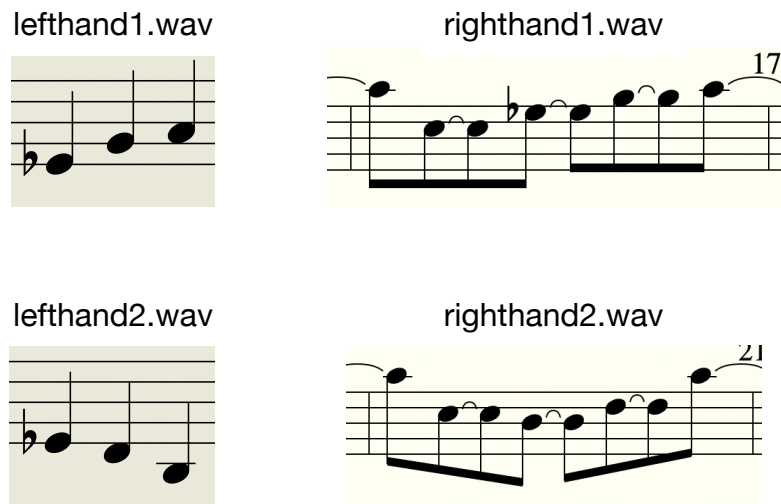


Figure 10.13: The four short piano phrases used in the piano and violin piece shown in Figure 10.14 and Figure 10.15. Note that the left hand phrases are three beats long, and the right hand phrases are four beats long. The right hand phrase starts halfway through a note, and is syncopated when compared to the left hand phrases.

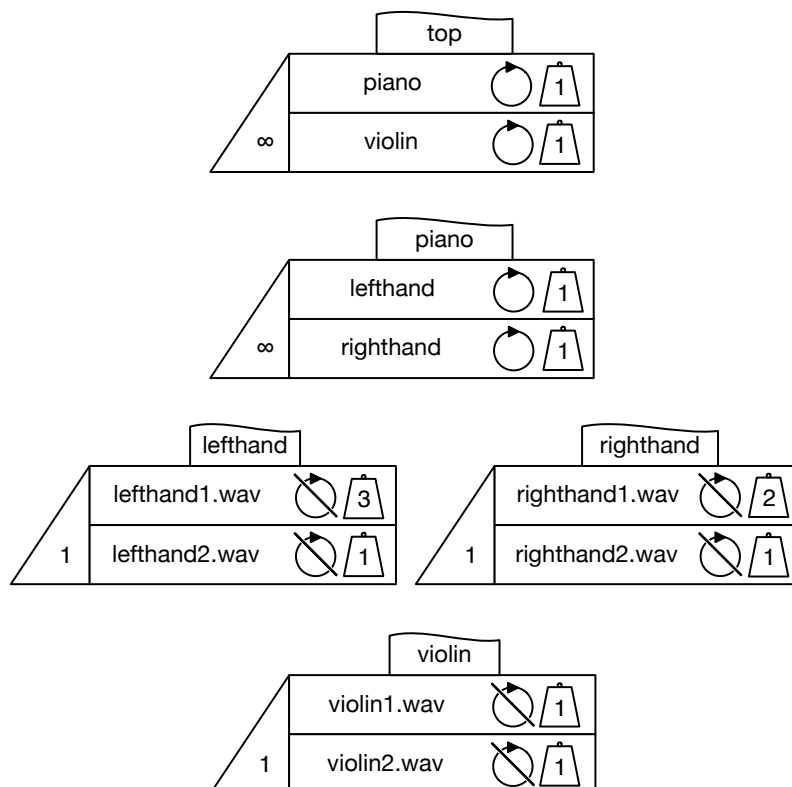


Figure 10.14: Piano and violin example 1. The left hand pattern is three beats long and the right hand pattern is four beats long, meaning that they will align every twelve beats. The violin sample is eight beats long. Audio of this example is available online (Bellingham, 2020a).

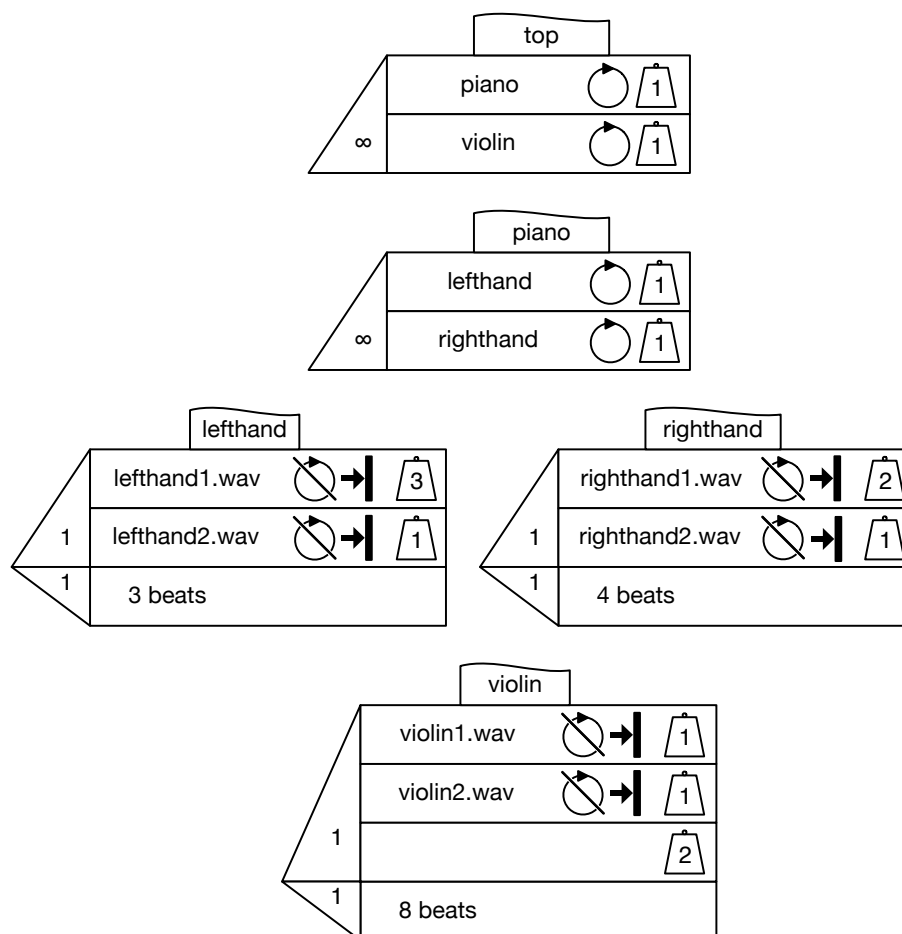


Figure 10.15: Piano and violin example 2, which takes the piano and violin example shown in Figure 10.14 and adds Time Choosers to moderate duration. Note the use of a blank lane in the violin Chooser; if selected this lane will result in a rest (silence) for the duration of the Chooser. Audio of this example is available online (Bellingham, 2020a).

- 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; this is shown in Figure 10.16. A duration of 3.05 beats, shown in Figure 10.17, creates a slower, tape-phase-style shift.

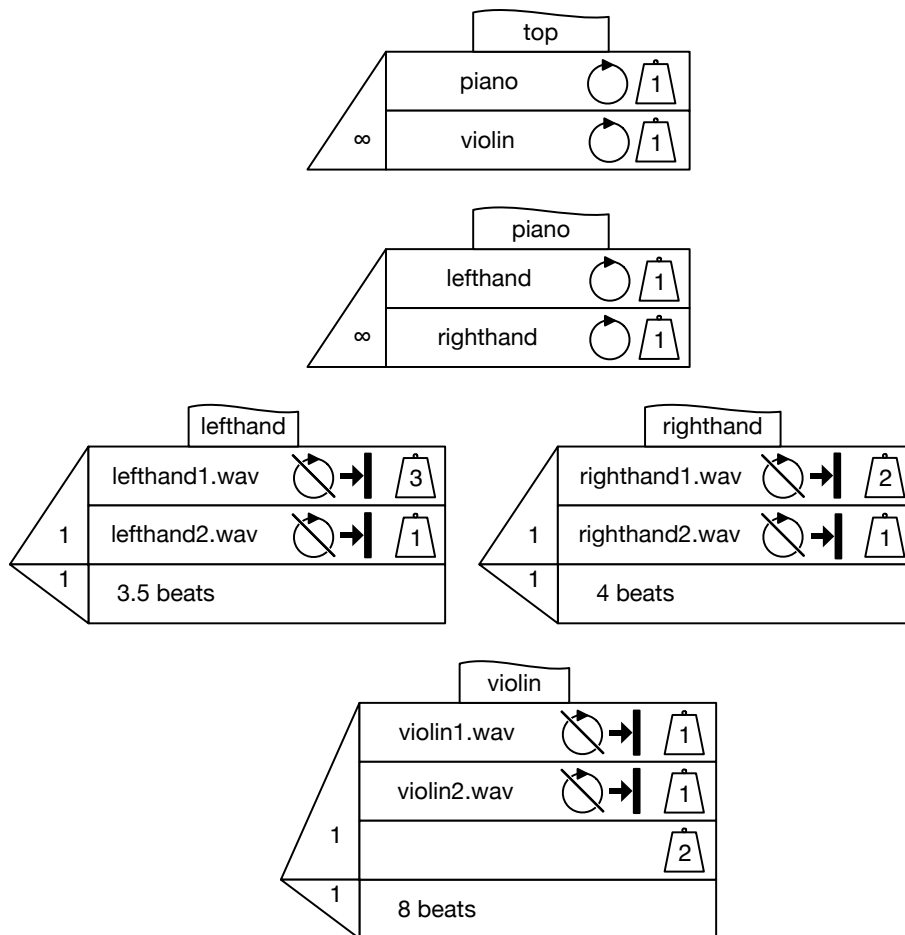


Figure 10.16: Piano and violin example 3. This is the same as the example in Figure 10.15 but with the duration of `lefthand` increased from 3 to 3.5 beats. Given that the audio samples in the ‘lefthand’ Chooser are both 3 beats long, this introduces a half-beat rest at the end of each phrase which creates an alternating syncopated/synchronised pattern when played alongside the 4-beat ‘righthand’ Chooser patterns. Audio of this example is available online (Bellingham, 2020a).

10.4.2 Drum machine example

The final example, shown in Figure 10.18, includes two doubly-nested Choosers. In order to clearly show the nesting hierarchy the Choosers are presented 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 10.18 shows beat-level durations in the child Choosers of the `perc` Chooser (upper row).

Consider the `hats` Chooser shown in the upper right of Figure 10.18; when run it will select and play one of three samples without looping and, after a duration of 1, 2, or 4 beats, will be triggered again by the looping lane in its parent Chooser (percussion). 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 16 or 24 beats. Finally, the `glockenspiel` Chooser is a

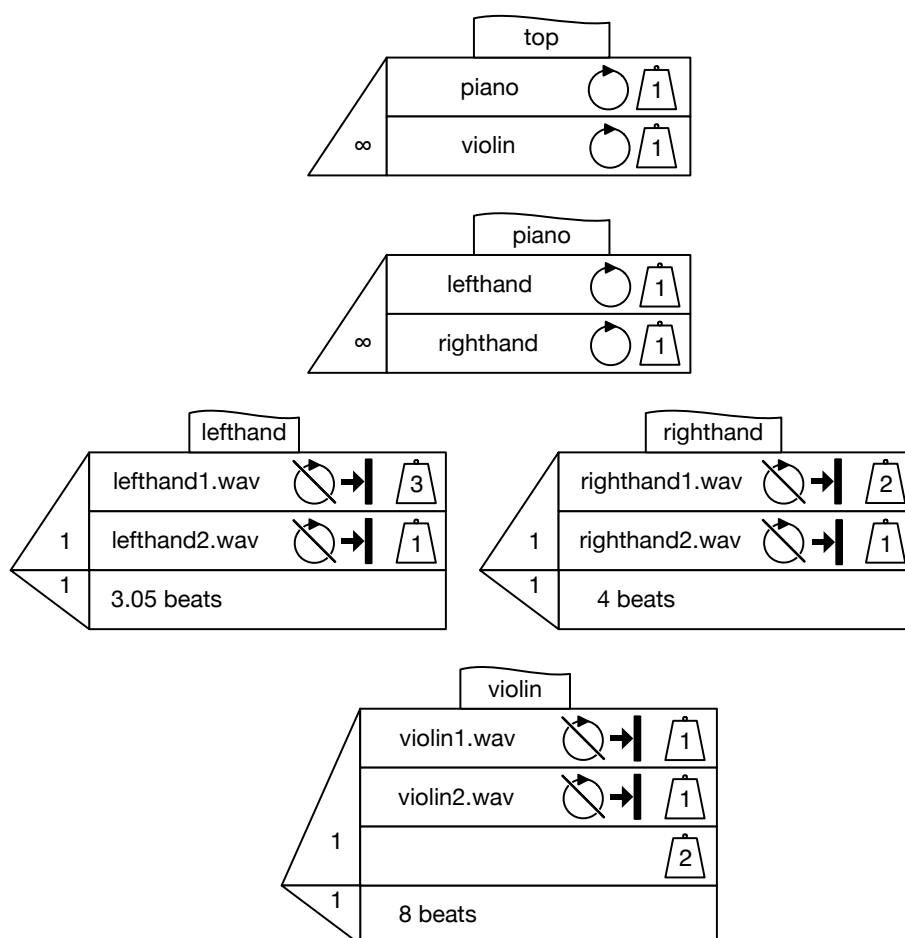


Figure 10.17: Piano and violin example 4. This replaces the 3.5 beat duration of the 'lefthand' Chooser with a duration of 3.05 beats, creating a slower phase shift between the 'lefthand' and 'righthand' patterns. Audio of this example is available online (Bellingham, 2020a).

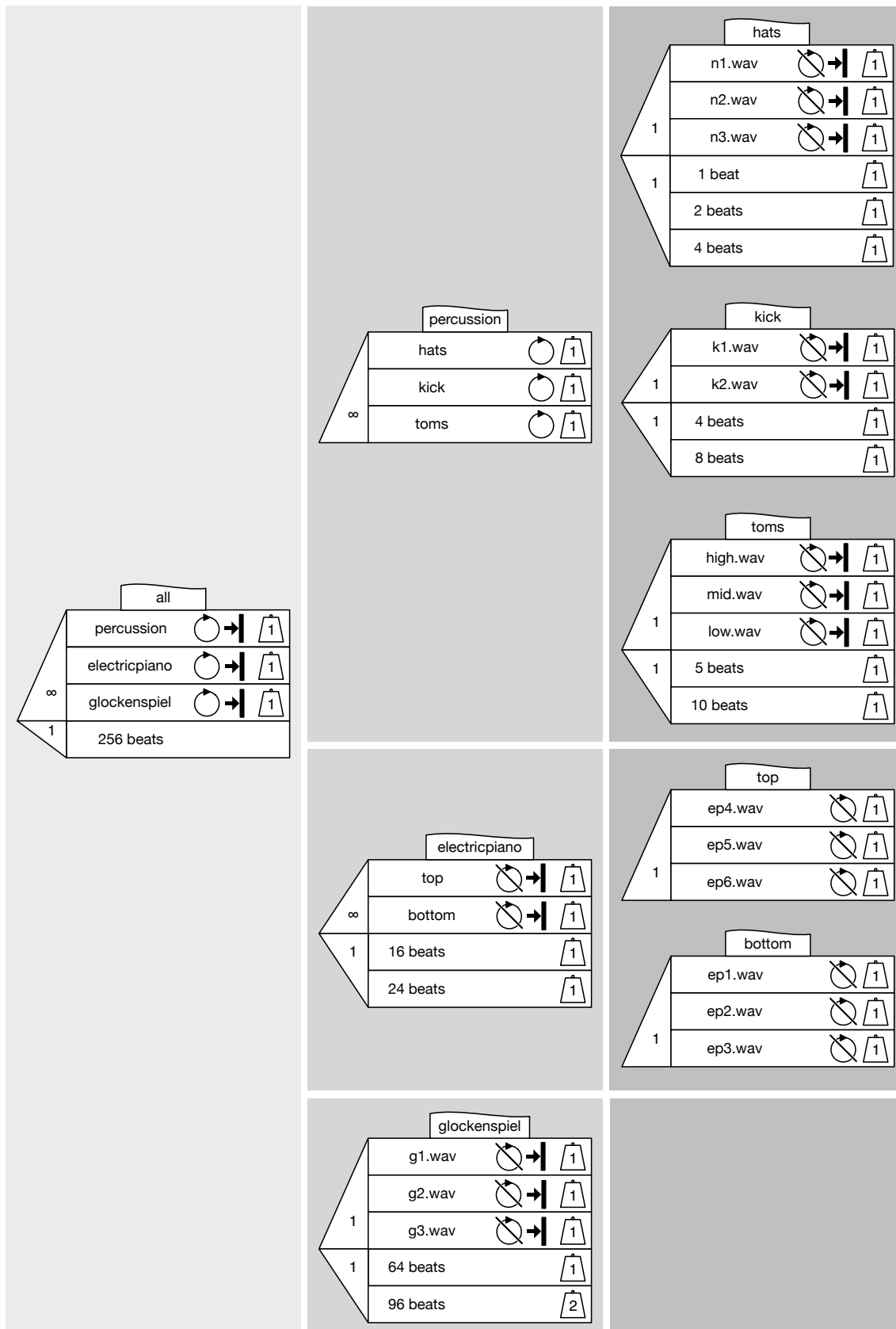


Figure 10.18: An example showing doubly-nested Choosers. The nesting hierarchy has been visualised in columns and rows for the reader's benefit; in fact, Choosers and sequences may be laid out freely. Audio of this example, named 'Drum machine example.wav', is available online (Bellingham, 2020a).

singly-nested Chooser which will play one of three glockenspiel samples every 64 or 96 beats. The parent Chooser, `all`, has a total duration of 256 beats, after which a hard stop message will immediately stop all lanes. The structure of the piece can be changed as 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 percussion 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 percussion 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 10.18; 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 percussion to infinity, resulting in the selection of percussion plus one other lane. If desired, the weight of glockenspiel 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 10.18. If the user wanted to remove toms from selection they could do so in one of four ways; the toms lane in the percussion 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 glockenspiel lane in the `all` Chooser could be set to not loop, and the lanes in the glockenspiel Chooser set to loop.

The `all` Chooser has a Time Chooser with a duration of 256 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`.

10.5 Conclusions

This chapter has outlined the design work undertaken in response to the findings from second user study. In order to allow for greater control over iteration, the ability to add a number to the loop icon was added to Choosers v3 (Section 10.2.1); as well as positive integers, 'zero loops' can be used to inhibit playback, and 'infinity loops' can be used as an alternative to the standard 'loop on' behaviour. The stop icons were updated for Choosers v3 (Section 10.2.2) to improve the readability of the notation. Metadata visibility, visualisation, and post-run tracing were also considered (Section 10.2.3) in response to the findings from the second user study; while there is potential for future development in all three areas, they were not implemented in Choosers v3.

In addition, the chapter has presented a number of speculative design changes, including the ability to use a blank lane (introduced in Section 10.3.1) in either a Soundable or Time Chooser, allowing 'nothing' to be a valid choice. This allows for a range of nondeterministic outputs including silence for a given duration, unbounded playback, or the skipping of a Chooser in a sequence. The blank lane design was implemented in Choosers v3. Alternative weight icons (Section D.3.1 of Appendix D) and sequencing via a 'big arrow' (Section D.3.2 of Appendix D) were both considered before being rejected in favour of the existing Choosers designs. A series of speculative design sketches (outlined in Section D.3.3 of Appendix D) were used to develop a drawer-based design for programmatic parameter control; while the design shows promise it was not implemented in

Choosers v3.

Finally, the design of Choosers v3 was tested via two musical examples. The piano and violin example (Section 10.4.1) showed an example of how a blank lane can be used to introduce musical rests nondeterministically (see Figure 10.15). The same example showed how Time Choosers can be used to create structural change by extending the duration of a Chooser beyond the length of the non-looping sample(s) contained in the Chooser; this introduces a period of silence (i.e. a musical rest) at the end of the Chooser's duration and was used to create various phase music effects (see Figures 10.16, 10.17). Both musical examples show the use of nested Choosers in the creation of complex structures. Lane looping in either parent or child Chooser allows for both 'select once and loop' or 'continuous reselection' behaviours, which can be applied to musical phrases of differing lengths. The example shown in Figure 10.14 demonstrates the use of nondeterministic selection of longer, multi-bar musical phrases, whereas the example in Figure 10.18 shows the continuous reselection of single percussive hits with nondeterministic duration selection.

Chapter 11

Third user study

In order to better understand how Choosers might be used by a variety of music practitioners, this chapter presents the findings from a series of ethnographically-inspired, semi-structured interviews with domain specialists in three areas: music composition, music production, and music computing education.

In Chapter 7 and Chapter 9 we outlined the first two user studies, designed to test the ability of self-taught music producers without programming skills to use Choosers to carry out a range of nondeterministic composition tasks, to identify user experience issues, and to identify tensions and trade-offs in the design of Choosers. These user studies consisted of programming walkthrough evaluations delivered using the Wizard of Oz technique. Each study led to a number of refinements to the notation which were, in turn, evaluated.

The third and final user study, presented in this chapter, was designed to better understand how Choosers may be used by practitioners in three relevant domains. In order to do so, a series of interviews were held with domain specialists in which the notation was introduced, interrogated via manipulable compositions, and used as a prompt for discussion. The participants were asked to give feedback on the potential uses for such a language, and what barriers to adoption—both from within and without the notation—may exist.

11.1 Design of the third user study

The interviews were designed as a probe to understand potential uses and barriers in three key areas; in music composition, in music production, and in music computing education. With this in mind, and as outlined in Section 3.7, the third user study took the form of ethnographically-inspired, semi-structured interviews with domain experts with experience in these three areas. The broad structure for the interviews is outlined below (Sections 11.1.1, 11.1.2, 11.1.3).

11.1.1 Introducing the interviewees to Choosers

The interviews began with a brief overview of the functionality of Choosers, delivered aurally by the interviewer, with examples played using a series of bespoke *SuperCollider* classes and pre-prepared templates to allow for near-instant playback as part of the Wizard of Oz technique. Participants were free to ask clarifying questions and to manipulate and modify the examples in order to explore the design.

The aural overview of Choosers, supported by Chooser images and playable and editable examples, covered the following points. References to relevant sections of the thesis are included for the reader's convenience.

- Adding samples as boxes and creating a sequence (Section 6.1.1), including arrangement (Section 6.1.5) and repeats via multiplication notation (Section 6.2.2);
- Soundable Chooser creation; weighted nondeterministic choice via the nose cone and lane weights (Section 6.1.2); looping (Sections 6.1.3, 10.2.1); infinity as a value for the nose cone, lane weights, and duration (Section 8.1.1);
- Full Choosers, including using the use of Time Choosers for both deterministic and non-deterministic duration (Section 6.1.4.2); stop behaviour (Section 8.1.2), musical rests (Section 6.1.4.1); and soundable content in a Time Chooser lane (Section 8.1.4);
- Naming and referencing Choosers (Section 6.1.6).

11.1.2 Musical examples for use in the interviews

After introducing the basic functionality of Choosers (see previous section), diagrams and code for three musical pieces were made available for the participants to read, play, and modify. An example of the *SuperCollider* code can be seen in Figure 11.1. The facilitator (i.e. the thesis author), using the Wizard of Oz technique (see Section 3.6), was able to quickly change parameters as requested by the interviewees. The participants were free to ask clarifying questions, make suggestions, and to modify the examples in any way. Given the ethnographically-inspired approach, and the fact that each participant represented a different type of user, there was no attempt to keep the direction of questioning identical for each user.

Example 1 A rock music piece, shown in Figure 11.2. This example shows the use of a sequenced arrangement populated by Choosers, and allows for significant musical changes via the manipulation of lane weights and nose cone values;

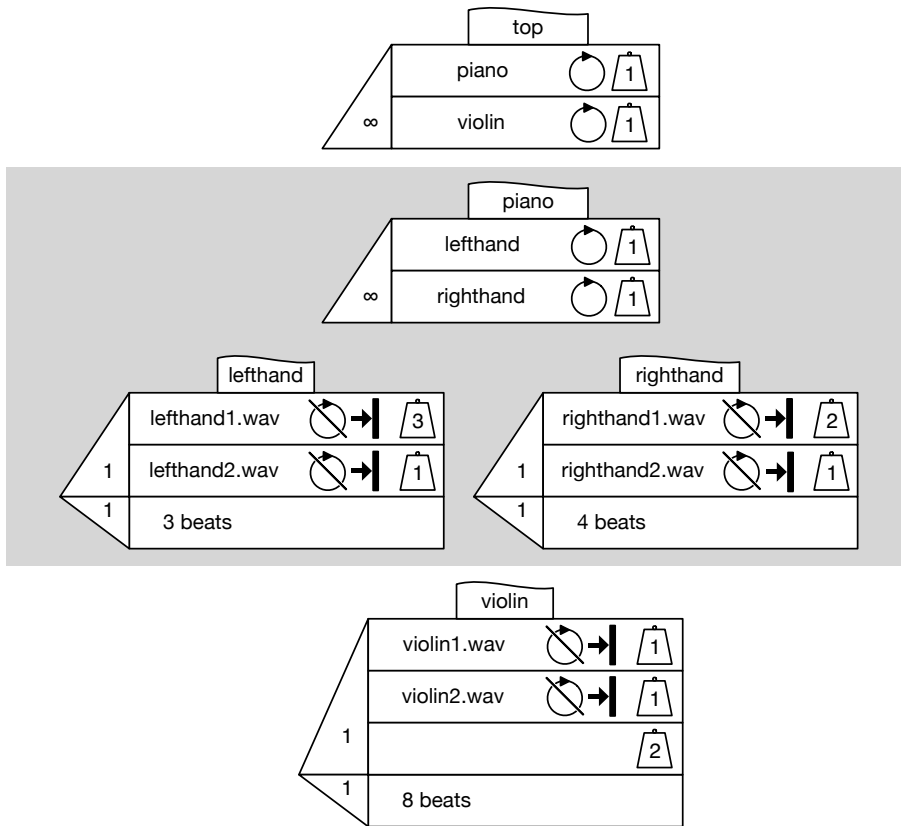
Example 2 Piano and violin, previously described in detail in Section 10.4.1 of Chapter 10. Two versions of this example were used; the first (see Figure 11.3) uses only Soundable Choosers, and demonstrates nesting. As with Example 1, changing lane weights and nose cone values creates significant variability in the output. In addition, the user can explore the impact of looping in the parent lane or in the child Chooser. The second version (see Figure 11.4) adds Time Choosers to three Soundable Choosers to make Full Choosers, giving the user the ability to experiment with changing the durations of each musical phrase, thereby creating the potential for syncopation, synchronisation, and phase music;

Example 3 Drum machine and electric piano, shown in Figure 11.5. This example, previously outlined in detail in Section 10.4.2 of Chapter 10, allows the user to explore the interrelationships offered in doubly-nested Choosers. Note that the nesting hierarchy has been visualised using columns and rows for the reader's benefit; the shading and white-ruled barriers are not part of the notation, and Choosers and sequences may be laid out freely.

11.1.3 Questions for participants

After reviewing the system and examples, each participant was asked the following questions. These questions were designed to initiate discussions on the potential users, uses, and barriers for the notation.

- What kind of user would Choosers work for?
- What can't Choosers do? What is Choosers not useful for?
- Are there interesting or useful comparisons with other tools that you use? Would Choosers replace or augment them?
- Can you see uses for Choosers? How could you use Choosers as part of your research/education/performance practice?



```

~lefthand = Xhooser.new;
~lefthand.name_("lefthand");
~lefthand.noseCone_(1);
~lefthand.addLane(Lane.new.namedSample(~lefthand).loopOff.hardStopOn.weight_(3));
~lefthand.addLane(Lane.new.namedSample(~lefthand2).loopOff.hardStopOn.weight_(1));
~lefttime = TimeChooser.new;
~lefttime.noseCone_(1);
~lefttime.addLane(TimeLane.new.beats_(3)); // sample is 3 beats long
~lefthand.timeChooser_(~lefttime);

~righthand = Xhooser.new;
~righthand.name_("righthand");
~righthand.noseCone_(1);
~righthand.addLane(Lane.new.namedSample(~righthand).loopOff.hardStopOn.weight_(2));
~righthand.addLane(Lane.new.namedSample(~righthand2).loopOff.hardStopOn.weight_(1));
~righttime = TimeChooser.new;
~righttime.noseCone_(1);
~righttime.addLane(TimeLane.new.beats_(4)); // sample is 4 beats long
~righthand.timeChooser_(~righttime);

~piano = Xhooser.new;
~piano.name_("piano");
~piano.noseCone_(inf);
~piano.addLane(Lane.new.nest(~lefthand).loopOn.hardStopOn.weight_(1));
~piano.addLane(Lane.new.nest(~righthand).loopOn.hardStopOn.weight_(1));

```

Figure 11.1: An example of a pre-prepared musical example. The figure shows the complete Chooser diagram as shown to the participants (top). The section shaded in grey corresponds to the *SuperCollider* template code (bottom), edited by the facilitator in response to modifications requested by the interviewees.

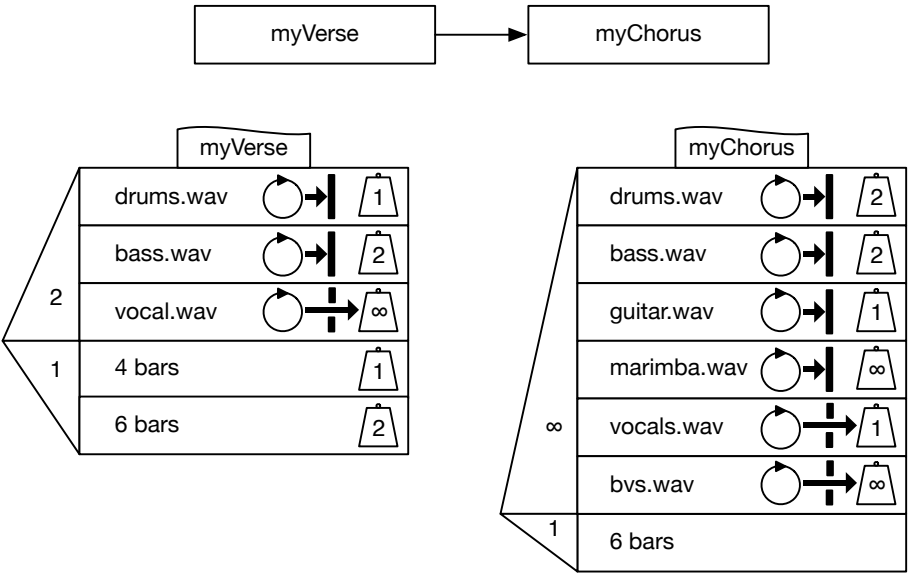


Figure 11.2: Example 1—rock music, showing the use of sequencing and named Choosers and offering significant scope for change via the manipulation of lane weights and nose cone values.

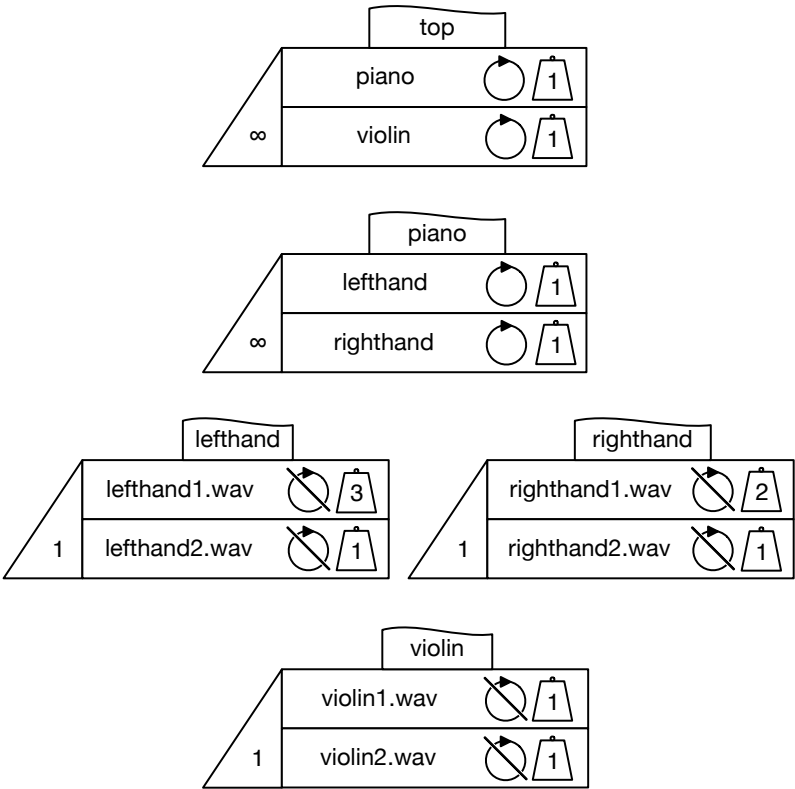


Figure 11.3: Example 2, version 1—piano and violin piece, demonstrating nested Choosers and the interplay between looping parent and child Chooser lanes. This example was described in detail in Section 10.4.1 in Chapter 10 and is duplicated here for the reader’s convenience.

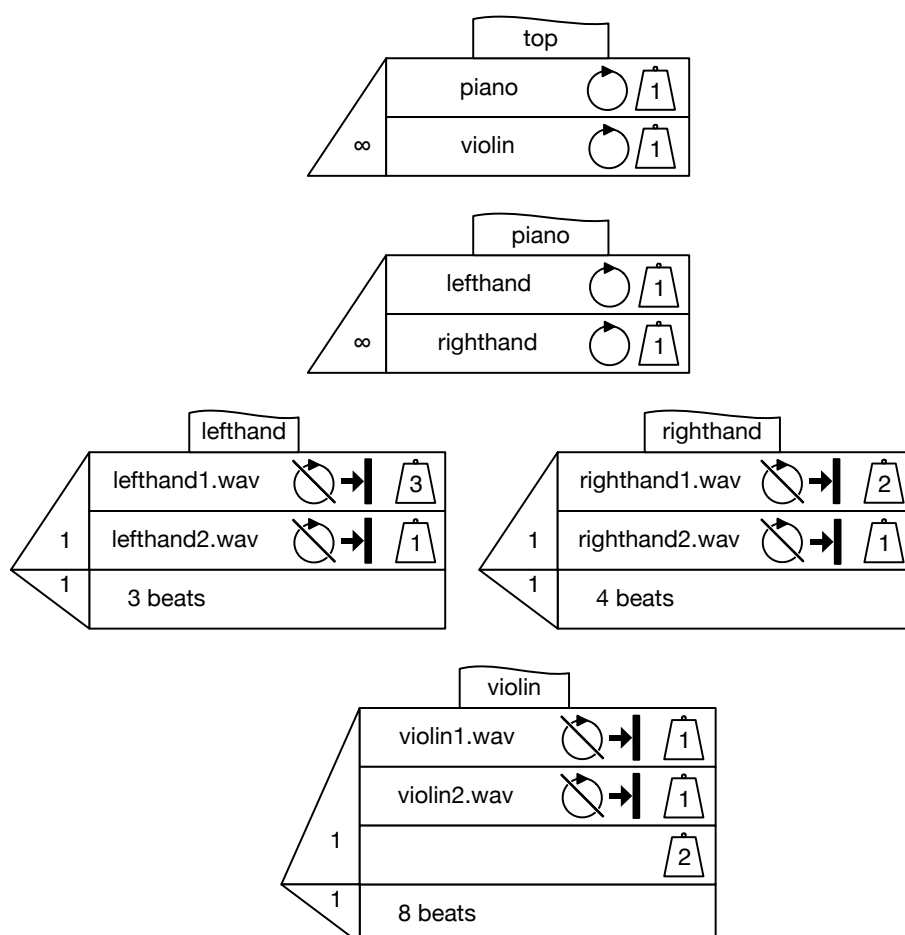


Figure 11.4: Example 2, version 2—this enhances the piece shown in Figure 11.3 by adding Time Choosers to the lefthand, righthand, and violin Soundable Choosers to create Full Choosers. This change allows for the durations of each Chooser to be overridden, offering access to syncopation, synchronisation, and phase music. The violin Chooser also contains a blank lane which will play silently for the duration of the Chooser if selected. As in Figure 11.3, this example was described in detail in Section 10.4.1 in Chapter 10, and is duplicated here for the reader's convenience.

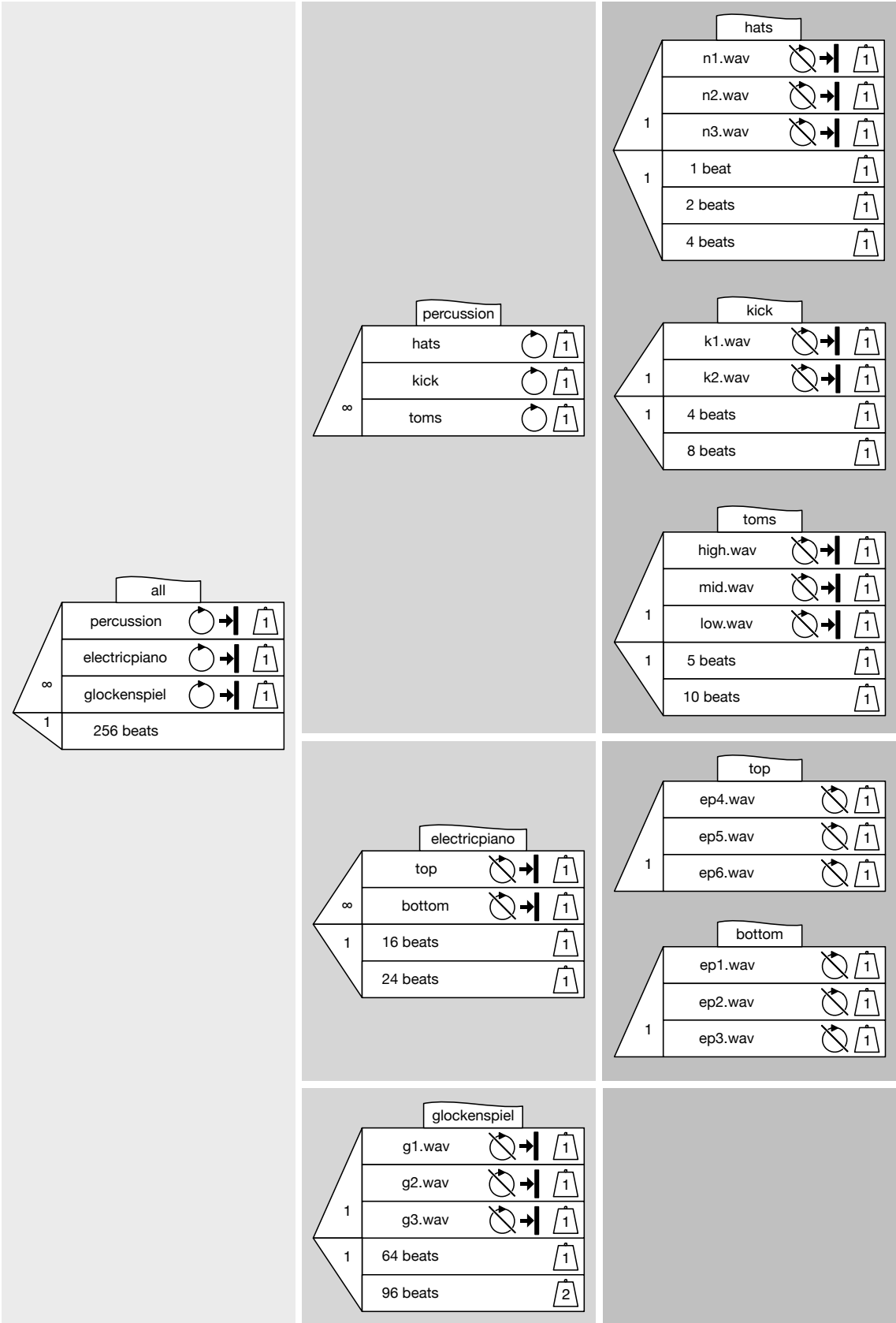


Figure 11.5: Example 3—drum machine, electric piano, and glockenspiel. This example is more complex than the previous pieces, and includes double-nested Choosers and the use of beat-duration (rather than bar-duration) Time Chooser lanes. The nesting hierarchy has been visualised in columns and rows for the reader's benefit. This example was described in Section 10.4.2 in Chapter 10 and is duplicated in this chapter for the reader's convenience.

- What barriers¹ would you need to be overcome for that to happen?

11.2 Participants

Given that music composition, music production, and music computing education are key areas of focus, this third and final user study was designed to further test and explore the listed issues with expert participants who can offer perspectives on each area of application. In practice, the three identified focal areas are not separate but are intimately linked. Each participant has significant experience in all three areas. All three participants were recruited opportunistically via professional bodies and contacts. They are not employees of either the University of Wolverhampton or of the Open University, and I have not previously collaborated with them.

BG is a record producer, engineer, songwriter, and musician, producing major chart artists as well as signing a major record deal with their own band. BG has worked with software companies on the design of two of the most widely-used pieces of music production software. They teach music production at a number of universities in the UK and Europe.

VM is a Senior Lecturer at a UK university. They are the course leader of undergraduate programmes in the area of music computing. VM is an active musician, composer, and audio engineer. VM's research interests include algorithmic composition.

YT is a composer and lecturer. YT has studied with a variety of composers, and has specialised in indeterminate and aleatoric compositions which typically do not involve the use of computers. YT uses the word 'indeterminacy' to describe both his own work and the output of Choosers. Section 1.3 of Chapter 1 defined indeterminacy as changes made by chance or by human agency, while nondeterministic selection was defined as the result of a probabilistic computer algorithm, made without human agency. Indeterminacy in music was previously considered in Section 2.6 of Chapter 2.

11.3 Changes made to code by the participants/interviewees

As part of the interview process the participants were free to make changes to each of the musical examples. VM made relatively minor changes to each example; their changes consisted of changing parameters (e.g. nose cone, lane weight, stop behaviour) and requesting playback to check their understanding. BG made similar parameter changes, and also rearranged the sequence in the rock example (Figure 11.2) to explore the interplay between stop behaviour and sequenced Choosers. YT made the majority of their changes in the piano and violin example shown in Figure 11.4; they made multiple changes to the durations of the lefthand and righthand Choosers, and explored the interactions in looping and stop behaviour between parent and child Choosers.

11.4 Data analysis

The interviews were recorded to video (BG, VM) or audio (YT), and later transcribed. Each transcript was analysed and divided into individual responses which were then broadly categorised. These categorised responses were iteratively developed into a series of broad themes, shown in the following section.

11.5 Findings from the expert interviews

Several themes emerged from the discussions prompted by the questions in Section 11.1.3. Here, each of the key themes is loosely categorised while providing an overview of the discussions.

¹With a particular focus on external barriers; for example, the curriculum; a reluctance or lack of confidence in users, etc.

11.5.1 Reflections on linearity

YT's comments during the introduction of Choosers suggested an interest in the impact of phonography on the conception of the work:

YT: It brings into question the whole idea about having a work. Do we like the idea of having a work? 'We know where we stand. Beethoven's Fifth Symphony, I know what that is'. Every time you listen to it it *is* different because the performance is different. I'm coming at it from that point of view; every performance is going to be different ... How do performers make choices in executing what they think?

After being introduced to nondeterministic choice of durations (via a multi-lane Time Chooser), BG needed clarification on this aspect of Chooser design; their lack of familiarity with nondeterministic processes may be due to their previous professional focus on the creation of nondeterministic linear recordings. However, after an aural clarification, they were able to predict how the multi-lane Time Choosers in Figure 11.2 and Figure 11.5 would play, and their subsequent performance suggested that they encountered no further issues.

11.5.2 Design consistency

While being introduced to the basic design of Choosers (Section 11.1.1), the participants were asked to predict the behaviour of a series of examples leading up to the examples shown in Section 11.1.2. The behaviour and comments of the participants suggested that they were able to learn the notation quickly with few issues. In part, this is due to the frames of reference that they brought to bear. For example, BG anticipated a DAW-like mechanism for specifying the number of loops/repeats in a Soundable Chooser lane, and correctly assumed that adding an integer in the loop icon, initially outlined in Section 10.2.1, would be the mechanism for this. Another example was VM's expectation that there would be a distinction between an independent copy of an element and a linked copy (an *alias*), similar to the behaviour of Apple's *Logic Pro* (2021).

The loop mechanism echoes the use of integers in the weight icon, and VM correctly assumed that copying an object and changing its name would allow for the creation of an unlinked copy:

VM: I like how, once you build those up, you can just refer to them and build up musical structures pretty quickly ... from that, if you've done that and you've got another myVerse and if you want to tweak the second one you just rename that and make a few edits to it ...

VM was also correct in assuming that sequences can be packaged, named, and referenced in the same way as Choosers.

When being introduced to multi-lane Time Choosers as part of the aural introduction outlined in Section 11.1.1, YT expressed their dissatisfaction at the inability to select more than one duration in a single Time Chooser:

Interviewer, introducing a multi-lane Time Chooser: You can't choose multiple durations.

YT: I can!

YT's work frequently uses polyrhythms and phasing and he seems to have been pushing back against a perceived limitation that would prevent him from including sounds of multiple durations or multiple time signatures. The power of combinatorial usage, and of nesting Choosers, was not apparent in some of the early tutorial examples. As a result, YT initially felt that there were unwelcome restrictions with regard to the selection of multiple durations. The possibility of allowing the

selection of multiple durations within a single Time Chooser was considered early in the development of Choosers, but the semantics seemed likely to cause confusion—see Section 6.2.4 of Chapter 6 for a comprehensive explanation. As a result, the use of multiple simultaneous durations was implemented via nested Choosers, as seen in the examples at the start of this chapter. YT returned to this perceived limitation later in the interview once nested Choosers had been introduced and after YT had experimented with the example shown in Figure 11.4. YT felt that, while nested Choosers offered the functionality they were initially seeking, the capability had not been clear to them:

YT: What strikes me is that you would get to that level of understanding, but not immediately. You would get that level of understanding after you’ve used it—it’s one of the subtleties.

YT was also interested in the relative merits of an on/off button as an alternative to an integer, previously discussed as part of the design of Choosers v2 in Section 8.1.2.1. YT felt that an integer was more consistent, as well as offering a clearer mental model (‘pick *n* of the following’).

BG made a guess, informed by his knowledge of common music notation, when encountering the hard stop icon for the first time:

BG: [Referring to the hard stop icon] ‘Is it a coda?’

Figure 11.6 shows a coda symbol next to the icons for both hard and soft stops; there are some arguable similarities in design, particularly in the use of vertical and horizontal lines, which could result in confusion. After a brief clarifying discussion, BG was satisfied that the icons were sufficiently distinct and they did not encounter any further issues.

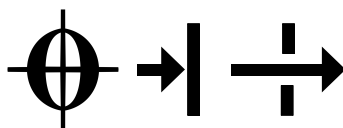


Figure 11.6: A coda symbol (left), compared to the hard stop (centre) and soft stop (right) icons.

Similarly, YT felt that the ‘loop off’ icon, introduced in Section 8.1.3, looked like a mute icon. This may be due to the use of a diagonal line to ‘cross out’ the loop icon, much like the ‘crossed out speaker’ used in Apple operating systems. The other two participants—both very familiar with music production software—had no issues with the icon.

11.5.3 Free placement of objects

As previously outlined in Section 6.1.1, Choosers deliberately allows the user to freely place objects. This allows the user to impose their own secondary notation—for example, a piece could read from left to right, top to bottom, bottom to top, etc. However, this freedom caused an issue with the legibility of the piano and violin piece shown in Figure 11.3. VM was initially confused by the layout of the piece:

VM: I’m not sure what order that they would play in but I can understand what’s happening and in each of the individual Choosers ... I can see they’re nested.

There is a potential trade-off between user freedom and legibility, particularly when a piece is read by someone who is not the author. There may be an argument for consistency when writing a piece of music; this could take the form of a ‘house style’, agreed on at the start of a collaborative process.

YT, a composer whose principle notation device is paper, assumed that the vertical alignment of Choosers in the piano and violin example shown in Figure 11.3 was significant. Interestingly,

they had a similar presumption of a left-to-right representation of time to the participants in earlier tests whose frame of reference was DAW software (see Sections 2.4.3, 7.5.3, 9.3.3). In this case, YT assumed that nudging a Chooser left or right would change the relative start times of the material. Once clarified, this did not pose any particular issues.

11.5.4 Visibility of metadata and global controls

For the system to be practically useful, participants expected access to global controls such as BPM and time signature, as well as to metadata such as sample length. The lack of surface-level visibility led to some clarifying questions from both VM and YT; VM could not immediately see the recorded tempo or relative lengths of each audio sample, and YT wanted to know how long each sample was. The current design of Choosers v3 does not clearly communicate some key information (e.g. the relative length of samples in a Chooser) without opening secondary windows.

11.5.5 Similarities to other systems

As part of the interview process, the participants were asked to consider similarities between Choosers and other systems. As may be expected given their different backgrounds and specialisms, each participant made unique suggestions.

VM is used to expressive music programming environments such as *Max*, *Pure Data*, and *SuperCollider*. As a result, they are very familiar with tools which enable nondeterministic music creation. VM was not, however, aware of a tool which makes the same trade-offs as Choosers:

Interviewer: Are you aware of any existing tools which are similar?

VM: No, not really ... everything else I'm familiar with [has] got more bells and whistles, or distractions, as well. There's a couple more steps to be able to realise the same result.

BG is very familiar with music production software such as *Pro Tools*, *Logic Pro*, and *Reason*. They are also highly experienced in using a range of hardware devices, including mixing consoles, hardware effects/processors, samplers, and sequencers. While not familiar with software which allows for nondeterministic selection, BG felt that there was a clear link between Choosers and the Akai MPC range of samplers/sequencers, exemplified by the MPC60, designed by Roger Linn [1988; in Exarchos (2019)] and shown in Figure 11.7. BG compared the MPC's ability to interact with patterns in real-time to Ableton *Live*'s functionality:

BG: Yes, an Akai MPC. It definitely has connections with the way an MPC works because everyone goes on about Ableton and ... it wasn't the first thing to do that. Actually, I used to be able to do that on an Akai MPC where ... I would just put a whole pattern on a pad and if I then hit another pad it would wait for the end of a bar ... there was a chance element to using an MPC.

The comparison made by BG to a sampler is interesting; the design of Choosers was influenced by the looping and 'one-shot' functionality of many samplers when designing the soft stop feature (see Section 6.2.3), so it is pleasing to see BG intuit this link. Ableton *Live* (shown in Figure 11.8) is software which was developed, in part, to offer functionality for live performance (Knowles and Hewitt, 2012). In particular, BG was referring to the ability of both MPC devices and Ableton *Live* to 'wait' for a particular file or cycle to finish. This functionality is similar to the use of a soft stop lane in a Chooser; the Chooser can only be released once the file has finished playing.

YT, a composer whose process is largely paper-based, does not routinely make use of software:

YT: I deal mostly in acoustic music, although I do use digital software for acoustic reasons.



Figure 11.7: The Akai MPC60, released in 1988 (Exarchos, 2019).

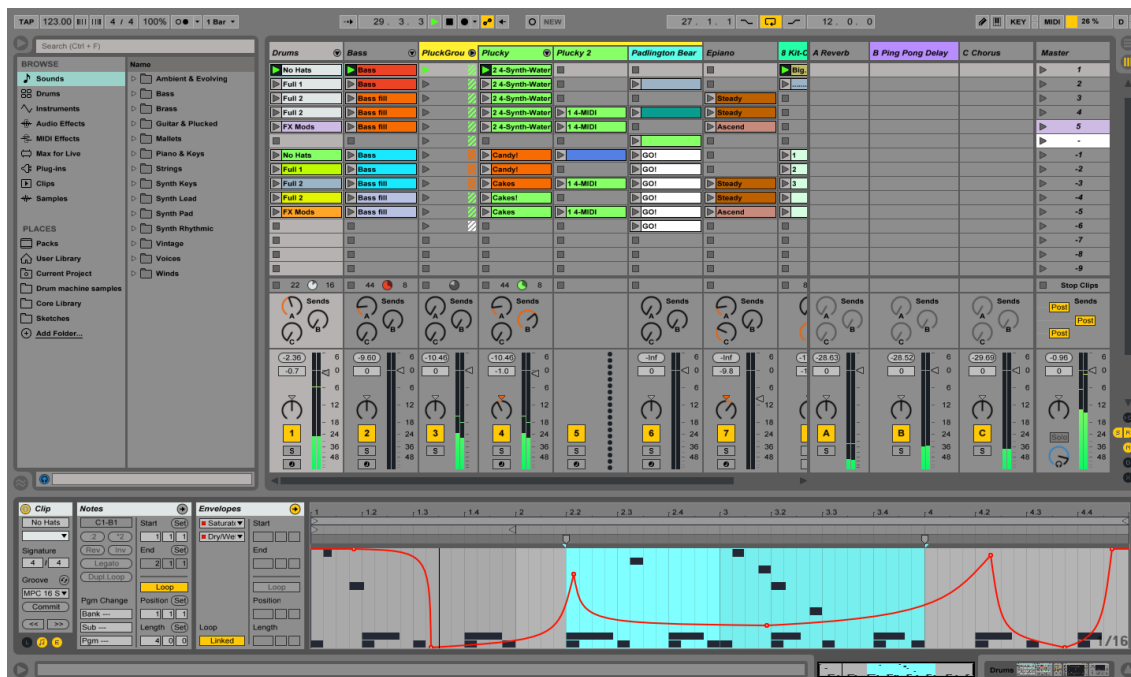


Figure 11.8: Ableton Live (Ableton, 2021).

As a result YT's frame of reference is not software. Instead, YT related the design of Choosers to the design of a piece of aleatoric music that was performed in the theatre where the interview took place, in which musicians played a new note only when their previous note had decayed until it was no longer audible. The soft stop option in Choosers reminded YT of this same musical device:

YT: What you're introducing is texture—the amount of stuff that's going on in any particular space, which is quite interesting. It's the thing I was doing acoustically with the piece we did in there [gesturing to the theatre] which is to let sounds decay. You do get these really interesting melodic lines. That's something that would happen with [Choosers] if you had a whole range of instruments. It's certainly something I would find quite interesting. You've introduced texture almost without consciously talking about it; it's built-in to what you're doing.

11.5.6 Who might want to use this?

11.5.6.1 Students

All three participants felt that Choosers could be useful in an education environment, with VM and YT specifically considering implementation in a curriculum. Each considered the requirements in their own specialist area, but there was a focus on the use of Choosers to explore the creative potential of nondeterministic composition.

VM considered the requirements of undergraduate students who would benefit from a system that allowed for conceptual and musical exploration and experimentation without having to engage with more complex systems:

VM: We teach with *Max*, *Pd*, and some *SuperCollider*, and some students thrive with that but it can be a bit of a Marmite experience ... some students do just actually get scared of it ... terrified of the language. It's something that can become a bit of a mental block for them so I think something that was easily accessible with some of the parameters moving to the back end and an easier drag-and-drop interface would be good for them so we could explore ideas and concepts before moving on to something that's more customisable. I can definitely think of a group of students who are in that category.

Upon questioning, VM considered the potential role of a simple notation in the development of an undergraduate student's understanding:

VM: I think it's good to have something that's really accessible [so students] can explore ideas quickly without having to worry too much about connecting things together in the right way, or having connected to the wrong input and something's not working, so they can explore ideas quickly without having to fuss around ... I suppose what we want them to do next is to explore a bit more and be able to deconstruct things.

YT considered the needs of composition students, rather than those studying music computing. YT felt that students may benefit from an accessible tool which enabled them to independently test ideas and explore serendipity, and which could be used to reinforce relevant concepts:

YT: There might be a possibility as a teaching tool to learn about the importance of serendipity in creating work; to maybe look at how this works in relation to a painting by Picasso which started as one thing and was converted to something else. [Picasso's painting] was a working tool to demonstrate slippage in thinking.

Given VM's focus on music computing students, they were asked to consider any potential downsides to the use of Choosers:

Interviewer: Would you have any pedagogic or andragogic concerns in the use of such a tool with undergraduate music computing students?

VM: Potentially. We're teaching ideas and [want students to be] thinking algorithmically, breaking complex ideas down themselves, and looking at how to solve those problems individually. We'd want to make sure we had that next step where they were actually able to build the tools themselves. I think as a sort of introductory step this could work quite well for some students, but I wouldn't see it as replacing that sort of process.

In summary, the participants felt that Choosers could enable students to focus on creative nondeterministic work rather than on implementation details, but that the expressivity ceiling of Choosers is likely to require a more expressive tool for some work.

11.5.6.2 Composers

As with education, all three participants felt that composers and songwriters could potentially benefit from access to an easy-to-use nondeterministic composition tool.

Early in the interview, YT was keen to understand how serendipity may be possible in the system; they felt this was important in a creative tool, and it is one reason why YT does not routinely use software in their composition process:

YT: Can you make mistakes with it? Can you do something that will be ... unforeseen and might generate an idea that you hadn't originally conceived?

Similarly, at the start of the interview, YT was curious about the range of control afforded to the user, and the effect of this on the user's creative ownership of the results:

YT: What about a sense of ownership? Purely from the user point of view, rather than a philosophical discussion on it. What sense of ownership does one have over any sense of creating work using this?

After an initial introduction to Choosers, YT began to interrogate the design with respect how sections are structured and the content of the sections. YT referred to the 'narrative' of a piece—the macro structure—and initially felt that Choosers may overly focus on structure:

YT: The system seems to be all about construction, and there seems to be little in the way of creating the source ... I'm sure you could find your own files to use, but form and structure is only one element of making a creative piece. Sometimes you're stuck with certain sounds but you can do certain things to them—you can put things inside a piano, you can find harmonics, glissando ... Spectralism is sound as colour rather than constructing a formal structure. At the moment, the impression I'm getting is that this is about generating structure—narrative—what follows what.

Later examples showed YT that Choosers is capable of manipulating content—see YT's comments on texture in Section 11.5.5.

YT has written a number of indeterministic compositions, and so has considered some of the same tensions that Choosers has encountered. As such, YT recognised that the types of control over nondeterminate actions that Choosers offers are important in a creative process:

YT: The thing about indeterminacy is that, at some point, someone has to determine something. So, whose responsibility is it?

VM has written algorithmic music using a variety of software, and has also assisted composers in constructing software systems to realise their work. As a result, VM feels that composers and musicians are often less concerned about implementation details than technical users:

VM: In terms of musicians or composers their requirements would be different; they are not necessarily interested in the way the tool is working.

This, VM felt, could mean that Choosers may have uses for composers.

11.5.6.3 Producers

BG considered the ways in which music producers and songwriters might make use of Choosers. BG felt that the trade-offs made in the design of Choosers—allowing for the creation of expressive music without requiring code—matched a type of user sometimes characterised as exhibiting characteristics partway between professional and amateur users:

BG: We talk a lot in FAST² about this group of people that sit in between amateurs and experts, and we call them Pro-Ams. Pro-Ams are people that want a professional result but don't want to know anything about the science or the process ... they're almost there to be entertained. They do know about masking and they do know about equal loudness contours, but they don't actually want to bring them into the workplace.

While BG felt that nondeterministic music is 'sadly lacking from most music', their sense was that a new system would be unlikely to be embraced by many professional users; BG's impression was that professional users are highly unlikely to move away from the tools and techniques that are proven to be both reliable and functional. Amateur users, or new users who are yet to choose tools and form habits, may be more likely to adopt a system like Choosers:

BG: ... certainly for the hobbyist or the amateur or the consumer ... the professional tends to be a little bit narrow-minded. The weird thing with technology is that the professionals are the last people to embrace new technology. They do stuff in an established way and, come hell or high water, they will stick with it.

BG felt that the soft stop functionality allowed for easy access to 'drops' (Solberg, 2014) and 'dropouts', a production technique used in the rock example (shown in Figure 11.2) in which the backing instrumentation is muted for a short time before a transition:

BG: It's really useful for doing dropouts actually, or for doing drops, so when you're mixing you might, when you come into a chorus sometimes you'll mute everything for half a bar apart from the vocal or something, just to make more of the chorus that's about to come in.

11.5.6.4 Listeners

Both BG and YT considered the possibility of nondeterministic domestic playback, with Choosers offering a potential way for listeners to interact with professionally produced music:

BG: There's a term that they use for it—personalisation. What I understand by that is where, in the future, consumers and musicians alike want to pursue so it could be ... like when you hear a remix you think 'oh, I would love to have dropped out with snare drum for those four bars there but I can't do it'.

²The *Fusing Audio Semantic Technologies for Intelligent Music Production and Consumption* project at Queen Mary University of London (FAST IMPACt Project, 2015).

This focus on phonography was particularly interesting given the focus on phonography throughout the project. Both BG and YT reflected on the work of Glenn Gould and his desire to allow the listener scope to change the recorded performance:

YT: One of the things that Glenn Gould used to talk about back in the '70s was giving the listener choice over speed etc., which is an odd concept. Did he really mean it? I think he probably did, but not everybody has that frame of mind.

BG directly considered how Choosers might be used to create alternative 'performances' by creating a nondeterministic output via weight changes.

BG: ... thinking almost along the Glenn Gould route, where he was interested in giving listeners the tools to assemble a performance at home or to assemble a different number of performances. The idea would be, using Choosers, for the user to be able to change the weights.

While discussing the ramifications of creating a mainstream nondeterministic playback tool, YT reflected on the impact of recording on the act of performance, and the ways in which the recording can be seen as the canonical work:

YT: Early recording musicians just used to play and be recorded playing, rather than 'doing' recordings; they weren't really interested. If you listen to one of the early pianists, such as [Alfred] Cortot, they don't even bother with 'right' notes—they go on and play, and that's it. Recording in some ways inhibited creativity in performance, so you get these clinically perfect performances and you think, 'where's the music gone?'

Similarly, BG considered the aesthetic effect of introducing nondeterministic imperfections or changes to a performance:

BG: The problem with generative, if you get into sort of Terry Riley or Steve Reich or suchlike, you don't quite know what's going to happen. Which is nice—it can be nice for certain Eno-esque [music]—but I do think there is a market for something here because people are... The comments I'm hearing back from musicians now is that the human ear is tuned in on imperfections—it notices imperfections—far more than it notices when things are all perfect, and if something changes they're more likely to pick up on that and it could ultimately make the music more interesting and more engaging.

11.5.7 Who would not want to use it?

When asked which users may not opt to use Choosers, VM felt that power users, whether in music composition or computing, would choose to use a more expressive tool:

VM: I suppose power users in both senses—composition and computing—of wanting the flexibility [that results from] creating everything themselves.

Interviewer: Would a Max user find any value in using this system?

VM: No, I think you could do everything in *Max* that you can do here. I don't think many of the more proficient users of *Max* would be tempted to use [Choosers] as they would be able to 'roll their own' which would exactly match their current task.

We then discussed the potential benefits of giving users a relatively simple way to create and manipulate a nondeterministic system. We talked about the experiential effect of understanding the output from the system, and the musical opportunities it offers.

VM: I can really understand that because those students that do struggle with it, or have a mental block with it, it's because they see it as something separate from their creative practice. We've worked hard to encourage them to see the creative possibilities of it but sometimes the technical demands of it just end up being too much of an ordeal for them. Giving them some immediate ways to creatively and conceptually explore which remove those technical boundaries would be quite valuable.

Conversely, BG felt that more advanced users would be able to better interrogate and manipulate the system, drawing a parallel between Choosers and mature and powerful software which uses presets for accessibility to novice users:

BG: An observation that I've made, rightly or wrongly, is that a lot of the tools that are designed [with presets] are ironically much safer in the hands of a professional than the person that it was designed for ... because of one word, interrogation. They have the ability to interrogate it.

YT also felt that, rather than presenting a problem, the limitations of Choosers are well matched to composition, and may prove both useful and inspiring:

YT: As a composer I often come up against limitations which are themselves generators for creative thinking. So if I have a system, say a piano, which is a very simple computer with all the pitches tuned in, that is the limitation I have. How do I find a creative way out of making something with it? So, for me, blocks are not necessarily blocks to creativity ... Having a completely open and free system doesn't necessarily lead to better or more creative work.

BG suggested that Choosers, or a tool like it, could be a useful tool when exploring options when composing. They envisioned a scenario in which the user explored various possible arrangement options by listening to the output of a nondeterministic system:

BG: One of the things that really troubles me about artificial intelligence (in music) is AI is very much ... concerned with averages ... but creativity [exists in] accidents. If you think of it statistically it lies in the outliers or exceptions. Creativity often lies in the exceptions and it seems to me that a lot of new technologies are very preoccupied with doing stuff for you where they should be assistive and maybe taking a lot of the kind of donkey work out and making suggestions.

In response to a clarifying question, BG agreed that computer-aided algorithmic composition, as defined by Fernández and Vico (2013) and introduced in Section 2.6, matches their characterisation of assistive AI.

11.5.8 Would you use it?

Once they had reviewed the design, all three participants were asked if they would be interested in using Choosers. BG considered the trade-offs in the design:

BG: Yes, most definitely ... I kind of come from this school anyway. I started using a BBC computer with Pro 16 and a Fairlight and they used to use a similar system ... It always occurred to me that it became a lot more complicated than it needed to be, so the idea of a pattern now in a conventional DAW is very fixed whereas a lot of the sequences I grew up using were never fixed; you could run stuff in counterpoint.

In contrast, VM was more hesitant given their background in using more expressive systems. Choosers was designed to abstract some complexity away from the user, whereas VM's teaching requires students to access a higher degree of expressivity. While VM accepted the trade-offs in the design of Choosers, these trade-offs introduced too many limitations for their use case of teaching. When asked what additional functionality they would like to see in Choosers, VM suggested transformations of the type used in DAWs such as *Cubase*:

VM: I think I'd like the flexibility to do some transformations on the sounds, or generate parts in real time, or be able to chop them up in different ways.

While experienced in writing music with aleatoric and algorithmic processes, YT had not used software as part of this process before. They considered how they might approach it:

YT: As a composition tool I might choose to import my own audio samples [and use the software to] reorganise the structure. As a composer if I was to use your software it would be for construction purposes—the narrative, mixing that narrative, and adding an indeterminate element to the playback of the sound.

Interviewer: Would you want a tool to do that?

YT: I would certainly try it. I would certainly use it to see what the possibilities are.

After reviewing nesting in Choosers, and satisfying themselves that multiple simultaneous processes are possible, YT expressed more enthusiasm:

YT: I would be quite interested to make something in it [Choosers]. I quite like the way you've set it up, and I like your approach.

BG reflected on the range of control over nondeterministic elements, from the fully deterministic output typical of phonographic recording to more radical variation:

BG: This is good because it bridges the gap ... the problem [is] you're either attached to a traditional studio paradigm or you're into generative [music]. This bridges the gap between generative and tradition for me because you have a little more control here, don't you?

Similarly, YT's previous work on indeterminate processes had led them to consider various types of indeterminate instruction, which they referred to as 'structured indeterminacy':

YT: I look at indeterminacy and a range of possibilities. We have more than one choice; there's more than just 'do what you want' or 'follow this exactly'; there's a whole range of possibilities. You can separate the musical parameters; pitch, duration, timbre, dynamic, attack, and all of the other things you could possibly think of. Take pitch, for example; you have tessitura, so I can tell you to play a C but you choose where on the instrument you play the note. Or, I tell you what notes to play but I leave where to play the note up to you. That is a type of indeterminacy. Or, I could say 'play something at the highest register of your instrument', or 'play 5 notes but you choose the pitches', and that is just scratching the surface.

Building on the same idea, YT described the problems with unstructured, or total, indeterminism in music. The composer may want indeterministic processes in the music but may want to impose some restrictions in the creation of a 'field of possibilities':

YT: For me personally, the reason I would use this would be in terms of creating interesting constructions and structures, and creating a field of possibilities. So, if it was me, I would want to have ownership over a fair amount of what was going on because I still think that's important. Introducing the idea of indeterminacy would be what is interesting, but what would be indeterminate would be from the known field—the judgement or the taste or whatever you want to call it would be at the sample stage and then using these tools to find some sort of interesting arrangement with indeterminate outcomes. What I would fear would be trying different numbers, and I would probably want to avoid that. I would want it to be more considered—considered indeterminacy. The interesting thing about indeterminacy, there are quite a few of us working in this field and we're doing it tangentially, the father of all of this was John Cage. Michael Finnissy, who worked with John Cage on a piece called *Europerras*, said he [Cage] wasn't as indeterminate as all that. If he didn't get what he wanted he would let you know about it. This idea of being completely open and free is not really what we want; we want a sense of ownership for whatever reason. So, what would be interesting for me is a sense of structured indeterminacy.

YT was interested in the focus in Choosers on making changes and auditioning the output. They reflected on how important this is for some types of creative practice:

YT: How do you find what it is you're trying to do? Does it come to you in a flash like it did for Mozart—which it didn't—or do you actually find out by doing it? I think mostly you find out by doing it. What was the term you used? Auditioning.

11.5.9 Enhancements

Both VM and YT suggested adding visualisation to the operation of Choosers. VM was interested in visual indications to show the selections made during playback:

VM: Is there a sort of visual indication of which ones it's chosen from?

YT felt that the use of infinity in a Soundable Chooser's nose cone was a useful feature, but that non-zero lane weights should then be temporarily greyed out to show that they do not affect lane selection.

YT was also keen for the design of the stop icons to line up with the rightmost edge of the Time Chooser lane(s); they felt that this would visually indicate that the Time Chooser triggers the Soundable Chooser lane stops.

VM felt that Choosers should ideally allow for editing of lane content (such as audio file editing, or a MIDI piano roll) within the software, rather than using a different piece of software.

Finally, YT asked to use common music notation rests in a sequence. We discussed the possible trade-offs in offering such notation as a 'musician' option.

11.5.10 Ecosystem

Both BG and VM felt that Choosers would benefit from being implemented as a plug-in within a linear DAW such as *Logic Pro* or *Pro Tools*. In this situation, the host software would provide mature and detailed control over linear playback while Choosers could provide nondeterministic output:

VM: I could see something like this working as a tool within *Logic Pro* ... I don't think it's going to replace human creativity, but it's just going to be another set of tools that we use. A lot of DAWs at the moment don't really have much [algorithmic] facility—Ableton has got some randomisation tools but they're more constrained [than Choosers]. I could definitely see there being a market for Choosers as a plug-in within existing DAWs, offering another way to experiment.

BG reflected on the workflow of many music producers, in which they use a number of DAWs in order to benefit from their varying strengths and affordances. For example, it is common in some genres for producers to create electronic material in *Reason* or *Ableton Live* while using *Logic Pro* or *Pro Tools* for linear recording and editing. Each piece of software encourages different working practices which can be creatively useful. BG felt that a hybrid system, such as *Logic Pro* and *Choosers*, would offer complementary affordances to the user:

BG: I see a lot of people these days using multiple DAWs because *Ableton* works in a certain way, and *Pro Tools* works in a certain way. *Pro Tools* uses the traditional paradigm a lot more than *Ableton* does, but people like things about both systems. It's the same as using maybe an Akai MPC with a traditional DAW because they both have a completely different workflow and you're sometimes thinking 'I wish these were both in the same place'. So, if I wanted to be able to be slightly generative and use something [like] 15/8 over the top of 3/4 but then I wanted the rest of the piece to be ... attached to the traditional tape-based metaphor, so absolutely a hybrid system could work ... I don't see them as being directly diametrically opposed to each other. I think they can be quite useful ... they can work together.

Given their independent suggestion of a plug-in and/or synchronised system, both VM and BG were asked if they would want to take the nondeterministic output of *Choosers* and 'freeze' it as one or more audio or MIDI files in the linear DAW and, if so, at what point in the process they would do so. VM suggested that the user should be able to choose whether they convert the output of *Choosers* to a linear file or if they maintain nondeterminism given a suitable playback format:

VM: It's an interesting question; as Brian Eno's famous quote of how people will look back on how we listen to music now and be surprised that we want to listen to the same thing all the time³. I suppose I can see people wanting the option to be able to freeze [the output of *Choosers*] but we might be consuming music in different formats, and there are times when we don't want things to play back exactly the same all the time, so I think having the option to be able to [play nondeterministically] would be useful.

BG reflected on how their professional practice differs from students that they work with. BG learned to make records on multitrack tape with synthesisers and samplers triggered via MIDI. The MIDI was programmed on a separate sequencer which was synchronised to the tape machine via SMPTE timecode. As a result of limited tape tracks, many producers would opt to run the synthetic and sampled content 'live' from the sequencer, rather than committing the parts to tape. As a result, the sequenced parts were editable right up to the point where the final mix was created. BG compared their own practice to that of students who have learned to record on DAWs:

BG: In order to complete I think I probably would. It's interesting how a lot of my students commit to audio in a way that I never used to do. I would leave stuff in MIDI forever; I'm a child of the late 80s and early 90s [and we would commit live MIDI to audio] right at the end. Whereas, I see students now [and] they commit to audio early but then use a different set of tools to manipulate that audio.

Both BG and VM felt that audio and MIDI files would be the most useful output formats for *Choosers*. In discussion, VM agreed that a protocol which allows for software to both synchronise playback and stream audio between them, such as *ReWire* (Propellerhead Software, 2019) or *Jack* (Davis and Letz, 2019), may offer an interesting solution. In this way, *Choosers* could be synchronised with a linear DAW, allowing them to be used together while maintaining the nondeterministic output of *Choosers* throughout.

³I really think it is possible that our grandchildren will look at us in wonder and say: 'You mean you used to listen to exactly the same thing over and over again?' (Eno, 1996)

Interestingly, YT considered the linear ‘capture’ of the nondeterministic output of Choosers to be a potential misuse of the tool:

YT: This is the problem when you roll it out, people will make use of it as they choose. It has a life of its own at that point ... So, you might have people who take a snapshot and then that becomes a deterministic thing.

11.5.11 Scope of the software and initial design assumptions

Once they had experimented with the piano and violin example shown in Figure 11.3, YT questioned whether many users in popular music would want to use software which offers a large range of options:

YT: A lot of users want to create very straightforward stuff. Who would want to use a system that gives you different options all of the time? Who is the actual end user? It seems to me that, in its [current] developmental stage, it's looking to be a more complex piece of software rather than a simple one. It's got potential for other things.

Interestingly, in suggesting that Choosers may be more complex than many users will need, YT offers a partial counterargument to VM's thought that Choosers is not expressive enough for ‘power users’.

YT also questioned whether there is a market for nonlinear audio:

YT: Is there a musical community who would be interested in using those kinds of samples with variables in terms of playback?

We briefly discussed Algorave and video game music as two different contemporary avenues for nonlinear music. We also talked about the development of electronic dance music and how certain key pieces of technology—such as the Roland TR-808 and TB-303—were used in ways that had not been anticipated by their designers (Shapiro and Lee, 2000).

11.6 Conclusions

Using as a focal point a stylistically varied set of musical examples available for manipulation in Choosers, three professional practitioners in the areas of music composition, music production, and music computing education were interviewed in depth to gather feedback on the potential uses for Choosers and on barriers to adoption.

The behaviour and comments of the participants suggested that they learned much of the notation quickly as they were able to leverage their knowledge of other systems. The design consistency of Choosers seemed to be largely effective in allowing the participants to learn an operation and then reuse the same logic in multiple different ways. Some of the functionality of Choosers was not always intuitive, especially in relation to the use of multiple durations; the expert interviewees found that Choosers still has a learning overhead. Accessing advanced functionality via nesting and variables was initially clear to the music computing expert, potentially due to their frames of reference, whereas the music producer and composer required additional input to reach the same level of competence. One of the trade-offs with nesting is between legibility and user freedom, and two of the participants initially found more complex examples hard to read. The composer expected to read the example from left-to-right, as they would read a score, whereas the design of Choosers allows for the free placement of widgets. A possible solution is a ‘house style’ for a collaborative project, in which the users may agree to place Choosers in a particular way e.g. left-to-right or top-to-bottom. All participants agreed that metadata (e.g. the duration of a given sample) and global controls should be made visible.

The music computing education expert had some concerns regarding the limited expressivity of Choosers, pointing out that sufficiently knowledgeable users could build their own bespoke tools using existing audio programming languages. They felt that Choosers may be instructive for those taking their first steps in nondeterministic music, but that a more expressive tool would eventually be required. They felt that, while several tools are capable of algorithmic music composition, none offer the same set of trade-offs; that there are a number of more expressive tools but they present a higher barrier to novice users. The participant felt that Choosers may enable novice users to focus on the aesthetic benefits of nondeterministic music without having to learn the syntax required by a more complex software tool. However, given that one of the aims of music computing education is the development of a deep understanding of the creation of bespoke tools, they felt that Choosers may be most useful as an introductory tool rather than a replacement for more expressive platforms; those who can already program may find Choosers unnecessarily limiting.

The music producer saw parallels between Choosers and pattern-based sequencing and live performance tools such as the Akai MPC60 and Ableton *Live*, although these tools are not, by default, capable of nondeterministic output. They suggested that Pro-Ams—amateur users who want a professional result without necessarily understanding the processes required—would be a possible target market. The participant felt that professional users would be unlikely to adopt a new system into an existing and mature workflow. They felt that a number of production and arrangement moves, such as creating ‘drops’ before a chorus, were easy to create in Choosers.

The composer participant has long used non-computer indeterminate processes in their work, and felt that Choosers may offer a welcome range of affordances that are not easily accessible in linear music software tools. For example, they were particularly interested in creating pieces in which serendipity may occur, resulting in unforeseen outputs. They had not previously used software which allowed for nesting, and the second (Figures 11.3, 11.4) and third (Figure 11.5) examples were instrumental in developing their understanding of how Choosers can be used to create note-level content as well as controlling macro-level structure. The composer felt that many musicians and composers do not care how a tool works, and they valued the default choices and trade-offs made in the design of Choosers. They felt that the limitations of a system, such as the constrained range or performance characteristics of an instrument, is essential in creative work, and they welcomed the constraints presented by Choosers.

The composer and music producer independently considered the effect on listeners of introducing nondeterminism, either by offering no end-user input or offering partial listener control. They did not think that there is a market for a domestic nondeterministic playback system, even though both found the concept interesting.

The participants suggested a number of improvements, such as visual indications to show selections during playback, and the greying out of unused parameters. The music computing expert wished to see in-place editing of lane content (e.g. via a MIDI piano-roll editor), and the composer asked for musical rests to be allowed in the notation. The music computing and music production experts considered the trade-offs between Choosers running as a DAW plug-in and of Choosers synchronising with a DAW. The participants did not agree about when the output of Choosers should be rendered to a linear audio file, if at all.

Chapter 12

Conclusions

This chapter summarises research contributions, outlines limitations, and points to future work.

This thesis has presented the design and evaluation of Choosers, a prototype algorithmic programming system designed to allow non-programmers access to nondeterministic music composition methods. The research was motivated by the following research questions:

How can nondeterministic composition tools be made accessible to non-programmers?

What methods would support the design and evaluation of nondeterministic composition tools for non-programmers?

What are the implications for music production, music computing education, and music composition?

This chapter will consider each of these questions in turn before considering limitations and presenting future directions for research.

12.1 Answers to the research questions

How can nondeterministic composition tools be made accessible to non-programmers?

This thesis has argued that many of the existing tools which are capable of algorithmic or non-deterministic music provide potential barriers to non-programmers. Some software requires users to be competent in the use of visual or text-oriented programming languages; several tools make use of domain-specific metaphors, such as patch cables, common music notation, or mixing desks; and users are sometimes presented with an interface which imposes a specific workflow, and some software resists changes to musical structure.

In order to maximise usability for novice users, Choosers creates a visual template which provides cues to show both the current musical structure and the parameters which can be adjusted. The primary motivation in this decision was to remove the burden on the user of learning a complex syntax while still having access to the capabilities discussed below.

By definition, nondeterministic algorithmic composition requires *nondeterminism*, and this is achieved in Choosers at essentially all levels of structure: an entire composition of arbitrary length and complexity can be contained in a single nondeterministic Chooser, or an entire piece might be fixed apart from fine points of percussion accompaniment, or determinism might be incorporated at all levels in between. The system is such that nondeterminism at any of these levels can be used and adjusted readily without using complex programming constructs.

Fundamentally, music involves one thing happening after another in time. One way to control this in Choosers is with *sequences*. Because sequences in Choosers never branch it might appear that this is a point of particular inflexibility in Choosers. However, the nose cones on Time Choosers make it easy to omit any item in a sequence, and Choosers themselves allow for branching at any point and at any level.

Music composition requires access to *parallelism*—the ability to play multiple parts synchronously. But during both composition and performance it can also be useful to nimbly pick out just one voice, or perhaps two at a time, or to have silence during the relevant period, or to be able to have the choice of which parts to play in parallel taken by the system and varied every time—perhaps in accordance with weightings. Choosers enables parallelism via Soundable Chooser lanes, with deterministic and weighted nondeterministic selections made possible via the interplay between lane weights and nose cone values.

Arguably fundamental to music is the capability to keep good *time*. Choosers allows durations at all levels of structure to be controlled tightly but flexibly via minimal UI changes. For example, using a Chooser (containing any level of internal complexity) either in isolation or together with a Time Chooser, the following results may readily be explored and re-explored:

- Skip the Chooser altogether;
- Have the duration depend on the choice of Soundable Chooser lane;
- Force the Chooser to have a fixed specified duration regardless of choice of Soundable Chooser lane;
- Force the Chooser to have to choose between a set of specified durations;
- Have a fixed duration overridden by lanes being allowed to finish their last loop;
- Have the duration depend on some fixed or indefinite numbers of repeats in a loop.

Music composition is a task that can be undertaken in multiple ways and so *compositional flexibility* may be beneficial. Choosers allows for the creation of a macro structure before populating each section (a top-down approach), or for the generation of musical fragments which are then linked together in sequences (a bottom-up approach). Music composition can benefit from iterative feedback as a way of galvanising the thought process (Collins, 2005); Choosers encourages the user to make quick changes without destructive changes to the source material which can be immediately auditioned to make an aesthetic judgement.

When comparing the representation of hierarchical musical information in Petri Nets with Choosers, it can be seen that Petri Nets retain readability when representing very simple musical constructs; in comparison, simple Choosers are more visually complex than PNs as they surface musical parameters for manipulation. However, Petri Nets become more viscous and harder to write, read, and manipulate when the musical content becomes more complex; relatively speaking, Choosers continue to surface musically meaningful parameters for the user to operate.

It has been shown through a sequence of studies that novice programmers could undertake a range of nondeterministic composition tasks, beginning with reading and understanding the notation and ending with the creation of a unique piece of music (including weighted nondeterministic choice, sequencing, and looping) with little training. While the majority of participants were able to learn and use Choosers with few problems, a small number of participants found the nondeterministic behaviour more difficult to understand, potentially due to the lack of a suitable frame of reference or a presumption that Choosers would follow the expectations set by DAWs and MIDI sequencers. Metaphors from outside music seemed to be useful in reinforcing the internal logic of Choosers (e.g. whether lane weight or nose cone values take priority).

What methods would support the design and evaluation of nondeterministic composition tools for non-programmers?

There are many issues involved when designing nondeterministic composition tools for non-programmers, with no one clear process to adopt. However, given the project's goal, a reasonable

approach is to begin with a review a range of existing tools using suitable design principles. The Cognitive Dimensions of Notations (CDN) was used as a ‘lens’ to help in identifying the trade-offs and decisions embedded in a representative range of software which is capable of algorithmic music composition. The CDN proved useful in identifying the trade-offs and design issues in various approaches to algorithmic composition software, namely:

- Expressive software typically requires significant user knowledge;
- Metaphor is often used;
- Domain-specific knowledge is often required;
- Working practices are sometimes imposed on the user;
- Some software is not aware of musical structure;
- Complex visual design can lead to low visibility and juxtaposability.

Once complete, the CDN review was used to develop initial heuristics which were used to inform the initial design of Choosers.

Based on ideas synthesised from the diverse approaches reviewed, a novel approach to visual programming for nondeterministic music composition by beginners was iteratively prototyped and refined. Programming walkthroughs were conducted, using a Wizard of Oz simulated front-end and a fully-implemented back-end, to rapidly gather actionable empirical data—specifically questions, problems, suggestions, and observations—from multiple participants. ‘Messy’ non-computational input methods (paper or a whiteboard) were found to be useful in communicating the unfinished and malleable nature of the design to the participants; given that the users were invited to be informants in the design process, the chosen input methods facilitated quick sketching and sharing of ideas. The design of Choosers lent itself to paper/whiteboard input due to the use of templates and a constrained number of interface elements; a more graphically complex design may require a different approach.

The third and final user study was designed to understand how Choosers could be used in the areas of music production, music composition, and music computing education. Experts from each area took part in ethnographically-inspired, semi-structured interviews in which they were introduced to Choosers before being given musical examples to consider and manipulate. From a methodological perspective, talking to the experts allowed us to explore uses in those areas by proxy; the experts enabled us to gather suggestive evidence and likely use cases without running studies in each area.

The two programming walkthrough studies were performed with undergraduate students at the University of Wolverhampton who I had not worked with at the time of the studies. I did not have existing relationships with the expert interviewees, who were recruited via professional bodies. The programming walkthroughs were performed according to the protocol outlined in Section 3.4, and made use of a complete implementation of Choosers run as a back-end to a Wizard of Oz front-end. This implementation did not offer any operational agency, and the Choosers created by users on paper or whiteboard were entered into and played by the back-end exactly and without any modification. Each user study followed the scenarios and scripts outlined in Chapter 7 and Chapter 9, and all user tests and interviews were video recorded and later transcribed to ensure accuracy. Both walkthroughs were designed to build participant understanding of Choosers, and both ended with a simple but open compositional task.

What are the implications for music production, music computing education, and music composition?

Independent domain experts in three key areas (music production; music computing education; and music composition) were interviewed in order to probe potential applications for Choosers. The participants were all experts in using domain-specific tools, and approached Choosers with differing frames of reference. The behaviour and comments of all three participants suggested

that they quickly understood the basic syntax, and each participant leveraged their existing knowledge of other systems. The music producer related Choosers to pattern-based sequencers—devices which allow for real-time tempo-aligned live performance. The music producer was also interested in the nondeterministic nature of Choosers, which is unusual in mainstream music production. The music computing educator quickly understood the possibilities available via nesting and variables due to their existing knowledge of programming languages. The composer approached Choosers as an alternative to common music notation, and needed time to explore the system before understanding the possibilities in nesting. Their frames of reference led both the composer and the music producer to assume a left-to-right representation of time; Choosers allows the user to create a sequence in any spatial direction, leading to the suggestion of a ‘house style’ when collaborating to standardise and to minimise confusion. More complex functionality, such as the use of multiple durations, proved initially problematic for the composer (outlined in Section 11.5.2); this suggests that Choosers requires learning, rather than being fully intuitive, and that the addition of behaviours that extend the expressivity have had a negative impact on usability.

The music producer’s comments and behaviour suggested that Choosers are capable of generating complex and evolving output without requiring programming knowledge. The discussion with the producer suggested a potential use case for Choosers as a form of computer-aided algorithmic composition (CAAC); they suggested that Choosers could be used for idea generation before rendering the outputs to linear files (e.g. PCM audio and MIDI files) for use in a DAW. Alternatively, Choosers could be synchronised with a DAW to allow the user to maintain the nonlinear output of Choosers alongside linear production tools. The music producer interviewee felt that Pro-Ams—amateur users who typically desire high-quality results without wanting to fully understand the processes required—would be a suitable market. Such users would benefit from the template-based design and musically-meaningful default behaviour of Choosers. Choosers allows for a number of common music production techniques, such as bar-aligned ‘drops’, making the system suitable for several styles of popular music.

Composers could use Choosers to create nondeterministic music without having to learn complex syntax. The composer interviewee was particularly happy to see that music written using Choosers is capable of serendipitous and unforeseen output, which they felt could be utilised as a teaching, as well as a compositional, tool.

Discussions in the interview with the music computing education expert led to the suggestion that Choosers could act as an introductory tool to explore nondeterministic music before the introduction of a more expressive language. Undergraduate students in the subject have a variety of previous experience; while some consider themselves programmers many enter their courses with a musical background. As a result, some students feel so overwhelmed by the requirements of programming that they are unable to engage with the aesthetic process of composition. Choosers surface musical operations via a relatively simple interface, allowing for experiential learning without the burden of learning complex textual syntax. Choosers could be used to demonstrate the musical viability of nondeterministic processes to such students, and to encourage them to continue to explore the possibilities offered by more expressive tools. The music computing educator had some concerns regarding the expressivity ceiling of Choosers, pointing out that sufficiently knowledgeable users may choose to use a more expressive tool over which they have a greater degree of finely-grained programmatic control.

12.2 Limitations

Simulated graphic front-end required the presence of a facilitator

The user studies presented in this thesis make use of the Wizard of Oz technique; participants worked on paper or a whiteboard and the results of their work were rapidly translated into runnable code by a facilitator. The lack of a fully implemented piece of software meant that any study

required a facilitator, thereby limiting the number of studies, their geographic location, and the duration of each study. The author acted as the facilitator and interviewer for the user studies presented in this thesis, which potentially introduced a risk of bias. Participants were aware that Choosers was the author's design and they may have moderated any criticism or self-censored, but criticism was explicitly invited and normalised in the sessions given that the methodological focus was to identify questions, problems, suggestions, and observations. As previously outlined in Table 3.1 in Chapter 3, a number of mitigations to the above issues were implemented. All participants were students who I had not worked with at the time of the walkthroughs, or practitioners from other universities with whom I did not have an existing professional relationship. All walkthroughs adhered strictly to the protocol outlined in Section 3.4, in which all user-generated content was entered without modification into a fully deterministic back-end which offered no scope for interpretation. All user studies were filmed and later exhaustively transcribed. In addition, all participant-generated Choosers were documented via photographs of the paper templates or whiteboard sketches, and by saving the edited *SuperCollider* templates used to play them.

Studies were short term

Due to the facilitator limitation listed above, the user studies undertaken were all relatively short; each user study pair worked together for between one and two hours. As a result, the studies did not explore the types of work that participants may undertake over a longer time period. For example, users may compose much more complex music over a longer time, which in turn could raise new and unforeseen issues and could suggest new design patterns. Such issues may require further development of the speculative textual language shown in Section D.3.3.5 of Appendix D. A longer-term study may be useful in understanding the ways in which an understanding of nondeterminism is linked to expectations framed by prior experience (e.g. use of DAWs or MIDI sequencers).

The studies were directed with only limited opportunity for creativity

The user studies, presented in Chapter 7 and Chapter 9, required the participants to work through a number of set scenarios. While the final scenario in each user study offered participants the opportunity to create a short piece of music, the task was constrained both by time and by the participant's limited knowledge and experience in using Choosers. A longer-term study would allow participants to more fully engage with Choosers and to use them in different contexts. As outlined in Section 1.3, this thesis has deliberately focussed on a subset of all possible nondeterministic music composition techniques, and a longitudinal study could explore the use of Choosers beyond the control of musical form via, for example, the use of shorter duration events (i.e. note level or shorter) in Choosers.

Choosers does not currently offer dynamic visualisation of execution

Graphical music software (such as MIDI sequencers, trackers, DAWs, synthesisers, and signal processors) typically use execution visualisation to represent various elements such as the software's current state, the control/data/signal flow, duration, and playback position. Participants in the user tests requested visual confirmation of Chooser selections (i.e. highlighting or flashing selected lanes) and the greying-out of unused parameters. DAW-style visualisation is complicated by the nondeterministic design of Choosers; for example, one Chooser's duration may depend on the nondeterministic selection from a second, nested Chooser; the duration therefore cannot be predicted.

12.3 Future work

Implement a front end for Choosers to allow for autonomous user studies

Choosers currently exists as a language design and a partial implementation via a set of custom *SuperCollider* classes. A full implementation of the design, including a graphical front end, will enable user studies which do not require a Wizard of Oz-style simulation.

Longer studies and observing Chooser use ‘in the wild’

Over the long term, and once the front-end of Choosers has been implemented, longer-term collaborative projects could be run to observe the uses of Choosers in a variety of situations with expert users in the areas of music composition, music production, live performance, and education. Initial work could focus on supporting collaborators in becoming experts in using Choosers; guiding the entry of their musical ideas into Choosers; facilitating their exploration of nondeterministic processes; and collaboratively refining their ideas to make use of the nondeterministic nature of the tool. Chooser use could be observed in a production environment for a specific project, such as an album or a module/unit on a course. In addition to the domain areas addressed in this thesis, future studies could directly address the use of Choosers in live performance. Similarities with existing tools used by DJs and live electronic musicians could be identified; for example, software such as Ableton *Live* and *Tidal*, and beat-aware hardware such as the Akai MPC series. I am keen to further explore the opportunities in which composers can use Choosers to create pieces in which musicians extemporaneously react to a nondeterministic output. My hope is that the lack of syntactic complexity in Choosers, coupled with the visual nature of the notation, can support communication between collaborators. It will be possible to gather both qualitative and quantitative data from the use of Choosers; users can reflect on their use of the tool and how it has affected their working practice, and the number and type of variations can be catalogued.

Develop textual commands

It can be anticipated that longer user studies with expert users may result in new, unforeseen issues. These new issues may require the development of new design patterns and behaviours. One way in which new behaviour could be implemented is via a more expressive set of textual commands, as outlined in Section D.3.3.5 of Appendix D. New commands could be implemented to extend functionality while maintaining core usability. This is due to the expert nature of the textual commands, which make use of progressive disclosure by offering expanded functionality only to those users who wish to make use of it.

Explore opportunities offered by various temporal granularities

The examples used in the thesis were focussed on sound objects and so used musical durations measured in bars or beats and lasting between a fraction of a second to several seconds. However, there is no reason why Time Choosers should not make use of any reasonable time scale and at any temporal granularity. For example, music software commonly uses BBT (bars, beats, MIDI ticks), SMPTE (hours:minutes:seconds:frames), and/or samples. Time Choosers could accept any combination of these time scales, including the use of common music notation durations. Time Choosers could optionally make use of any common time scale as distinguished by Roads (2004), ranging from supra (years, months) through macro (days, hours, minutes) and meso (minutes, seconds) to micro (ms, μ s). The use of micro-musical durations in Time Choosers would enable a single Chooser to output a grain of sound, the microacoustic event that serves as the basic ‘building block’ of granular synthesis (Xenakis, 1971; Roads, 1988). The combination of a programmatically-controllable anchor point and micro-musical durations would allow for the repeated nondeterministic selection

of grain content and duration from a single audio sample; even more complex output would be available by adding nondeterministic sample selection to the same Chooser.

Wider range of Soundable Chooser content

All Soundable Chooser lane content in this thesis has consisted of audio samples. However, and as the name suggests, the contents of Soundable Chooser lanes could include anything which makes a sound. For example, the lane could host a MIDI file which is sonified via a software synthesiser or sampler.

Improved SuperCollider implementation

The current back end implementation in *SuperCollider* is functional and was sufficient for the user tests presented in the thesis. However, it would be beneficial to rewrite at least some parts of the code to make use of classes such as `Pbind` for separating and combining multiple streams, patterns, and events (Wilson et al., 2011).

Visualisation, traces, and editing tools

While working on paper or a whiteboard, participants suggested ways in which a graphical front end might support their work; this included highlighting active Choosers and active lanes within Choosers, and accessing metadata for the contents in Soundable Chooser lanes. The potential addition of post-run traces to allow users to analyse and/or replay specific events was identified during development.

The design exercises presented of Appendix D offer possible enhancements to facilitate the editing of lane contents. The Soundable Chooser lane overlay outlined in Section D.2.4 and enhanced in Section D.3.3 gives the user control over a limited number of non-destructive playback parameters such as volume, trim, anchor point, and pan/balance. Similar control could be given to a wider range of parameters, including real-time or offline signal processing (e.g. normalisation, or reversing the audio file). If support is added for MIDI files in Soundable Chooser lanes, in-place editing could be implemented via one or more suitable editors (e.g. piano roll, list, notation, etc.).

12.4 Concluding remarks

Choosers was designed for novice programmers, and evidence was found of potential for novices in the domains of music composition, music production, and music computing. While novices in music computing may want or need to move on to a more expressive tool, Choosers offer a useful set of trade-offs for users who wish to explore nondeterministic music without having to learn a more complex formalism. Indeed, users for whom computer science is not their key focus may never need to move on to a more expressive tool if they do not need to move beyond the capabilities of Choosers. Such users are not seeking domain expertise in computing but are instead using the computer as a tool; they may remain relative computing novices while developing domain expertise in other areas. It is hoped that Choosers may support novice programmers in exploring the rich creative potential of nondeterministic music-making.

References

- Aaron, S. et al. (2016) The development of sonic pi and its use in educational partnerships: Co-creating pedagogies for learning computer programming. *Journal of Music, Technology and Education*. 9 (1), 75–94.
- Aaron, S. & Blackwell, A. F. (2013) ‘From sonic pi to overtone: Creative musical experiences with domain-specific and functional languages’, in *Proceedings of the first ACM SIGPLAN workshop on functional art, music, modeling & design*. FARM ’13. September 2013 New York, NY, USA: Association for Computing Machinery. pp. 35–46.
- Ableton (2021) *Ableton Live 11*. [online]. Available from: <https://www.ableton.com/en/live/> (Accessed 16 April 2021).
- Ableton (2020) *Max for live*. [online]. Available from: <https://www.ableton.com/en/live/max-for-live/> (Accessed 29 March 2021).
- Adkisson, H. P. (2005) Breadcrumb Navigation. Web Design Practices [online]. Available from: <http://www.webdesignpractices.com/navigation/breadcrumb.html> (Accessed 12 August 2018).
- Alpern, A. (1995) *Techniques for algorithmic composition of music* [online]. Available from: <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.23.9364&rep=rep1&type=pdf> (Accessed 7 November 2021).
- Anacleto, J. & Fels, S. (2013) ‘Adoption and appropriation: A design process from HCI research at a brazilian neurological hospital’, in *IFIP conference on human-computer interaction*. 2013 Springer. pp. 356–363.
- Anders, T. (2012) *Strasheela*. [online]. Available from: <http://strasheela.sourceforge.net/strasheela/doc/index.html> (Accessed 10 June 2020).
- Angelis, V. et al. (2013) Testing a computational model of rhythm perception using polyrhythmic stimuli. *Journal of New Music Research*. 42 (1), 47–60.
- Apel, W. (1961) *The notation of polyphonic music, 900-1600*. Medieval Academy of America.
- Apple Inc. (2021) *Logic Pro X User Guide*. [online]. Available from: <https://support.apple.com/en-gb/guide/logicpro/welcome/mac> (Accessed 7 November 2021).
- Ariza, C. (2011) *athenaCL*. [online]. Available from: <http://www.flexatone.org/article/athenaCLMain> (Accessed 30 September 2018).
- Assayag, G. et al. (1999) Computer-assisted composition at IRCAM: From PatchWork to Open-Music. *Comput. Music J.* 23 (3), 59–72.
- Auslander, P. (1999) *Liveness: Performance in a Mediatized Culture*. London: Routledge.
- Avid Technology, Inc (2021) *Pro Tools Reference Guide*. Version 2021.3. [online]. Available from: https://resources.avid.com/SupportFiles/PT/Pro_Tools_Reference_Guide_2021.3.pdf (Accessed 7 November 2021).
- Baggi, D. L. & Haus, G. M. (2013) *Music Navigation with Symbols and Layers: Toward Content Browsing with IEEE 1599 XML Encoding*. John Wiley & Sons.
- Ball, P. (2011) Schoenberg, serialism and cognition: Whose fault if no one listens? *Interdisciplinary Science Reviews*. 36 (1), 24–41.
- Baratè, A. et al. (2005) ‘Music Analysis and Modeling Through Petri Nets’, in *Computer Music Modeling and Retrieval*. Lecture notes in computer science. [Online]. Springer Berlin Heidelberg. pp. 201–218. [online]. Available from: http://link.springer.com/chapter/10.1007/11751069_19.

- Baratè, A. (2008) *Music Description and Processing: An Approach Based on Petri Nets and XML*. INTECH Open Access Publisher. [online]. Available from: http://www.researchgate.net/profile/Adriano_Barate/publication/221786820_Music_Description_and_Processing_An_Approach_Based_on_Petri_Nets_and_XML/links/004635268b80f75f6b000000.pdf.
- Baratè, A. (2009) Real-time Interaction with Music Structures in IEEE 1599. *Journal of Multimedia*. [Online] 4 (1), 15–18.
- Bardzell, J. et al. (2015) ‘Immodest proposals: Research through design and knowledge’, in *Proceedings of the 33rd annual ACM conference on human factors in computing systems*. CHI ’15. April 2015 New York, NY, USA: Association for Computing Machinery. pp. 2093–2102.
- Bell, B. et al. (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.
- Bell, B. et al. (1991) ‘Usability Testing of a Graphical Programming System: Things We Missed in a Programming Walkthrough’, in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’91. [Online]. 1991 New York, NY, USA: ACM. pp. 7–12. [online]. Available from: <http://doi.acm.org/10.1145/108844.108846>.
- Bell, B. et al. (1994) Using the programming walkthrough to aid in programming language design. *Software: Practice and Experience*. 24 (1), 1–25.
- Bellingham, M. et al. (2014a) ‘A Cognitive Dimensions analysis of interaction design for algorithmic composition software’, in Benedict du Boulay & Judith Good (eds.) *Proceedings of Psychology of Programming Interest Group Annual Conference 2014*. 2014 University of Sussex. pp. 135–140. [online]. Available from: http://www.sussex.ac.uk/Users/bend/ppig2014/15ppig2014_submission_10.pdf.
- Bellingham, M. et al. (2014b) *An analysis of algorithmic composition interaction design with reference to cognitive dimensions*. [online]. Available from: <http://computing-reports.open.ac.uk/2014/TR2014-04.pdf>.
- Bellingham, M. et al. (2017) Choosers: designing a highly expressive algorithmic music composition system for non-programmers. *2nd Conference on Computer Simulation of Musical Creativity*. [online]. Available from: <http://hdl.handle.net/2436/621151>.
- Bellingham, M. et al. (2018) ‘Choosers: The design and evaluation of a visual algorithmic music composition language for non-programmers’, in Luke Church & Antranig Basman (eds.) *Proceedings of the 29th annual workshop of the psychology of programming interest group - PPIG 2018*. 2018 [online]. Available from: <http://www.ppig.org/sites/ppig.org/files/PPIG-2018-proceedings.pdf>.
- Bellingham, M. et al. (2016) ‘Designing a Highly Expressive Algorithmic Music Composition System for Non-Programmers’, in *DMRN+11: Digital Music Research Network One-Day Workshop 2016*. Digital music research network one-day workshop. December 2016 [online]. Available from: <http://oro.open.ac.uk/52732/>.
- Bellingham, M. (2020a) *Musical examples referenced in chapter 11*. [online]. Available from: <https://doi.org/10.21954/ou.rd.12667037> (Accessed 17 July 2020).
- Bellingham, M. et al. (2019) ‘Toward meaningful algorithmic music-making for non-programmers’, in Luke Church et al. (eds.) *Proceedings of the 30th annual workshop of the psychology of programming interest group - PPIG 2019*. 2019 [online]. Available from: <https://www.ppig.org/files/2019-PPIG-30th-bellingham.pdf>.
- Bellingham, M. (2020b) *Tutorial videos referenced in chapter 10*. [online]. Available from: <https://doi.org/10.21954/ou.rd.12310010> (Accessed 11 June 2020).
- Bellingham, M. (2020c) *Tutorial videos referenced in chapter 8*. [online]. Available from: <https://doi.org/10.21954/ou.rd.12309989> (Accessed 11 June 2020).
- Berg, P. (2014) *Algorithmic Composition Toolbox*. [online]. Available from: <https://www.actoolbox.net/> (Accessed 18 March 2019).
- Bernardo, F. et al. (2020) Designing and evaluating the usability of a machine learning API for rapid

- prototyping music technology. *Frontiers in Artificial Intelligence and Applications*. 313.
- Birchman, J. J. & Tanimoto, S. L. (1992) 'An implementation of the VIVA visual language on the NeXT computer', in *Proceedings IEEE workshop on visual languages*. September 1992 pp. 177–183.
- Blackwell, A. F. et al. (2000) 'Cognitive dimensions and musical notation systems', in *Proceedings of international computer music conference*. 2000
- Blackwell, A. F. et al. (2001) Cognitive Factors in Programming with Diagrams. *Artificial Intelligence Review*. [Online] 15 (1-2), 95–114.
- Blackwell, A. F. et al. (2019) Fifty years of the psychology of programming. *International Journal of Human-Computer Studies*.
- Blackwell, A. F. & Green, T. R. G. (2000) 'A Cognitive Dimensions Questionnaire Optimised for Users', in *12th Workshop of the Psychology of Programming Interest Group*. April 2000 Cozenza, Italy:
- Blackwell, A. & Collins, N. (2005) 'The Programming Language as a Musical Instrument', in *Proceedings of PPIG05 (Psychology of Programming Interest Group)*. 2005 pp. 120–130.
- Blackwell, A. & Green, T. (2003) 'HCI Models, Theories, and Frameworks: Toward a Multidisciplinary Science', in John M Carroll (ed.) San Francisco: Morgan Kaufmann. pp. 103–134.
- Bresson, J. et al. (2011) 'OpenMusic: Visual programming environment for music composition, analysis and research', in *Proceedings of the 19th ACM international conference on multimedia*. MM '11. November 2011 New York, NY, USA: Association for Computing Machinery. pp. 743–746.
- Browning, T. R. (2001) Applying the design structure matrix to system decomposition and integration problems: A review and new directions. *IEEE Transactions on Engineering management*. 48 (3), 292–306.
- Bullock, J. et al. (2011) Integra Live: a new graphical user interface for live electronic music. *International Conference on New Interfaces for Musical Expression*. [online]. Available from: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.449.533&rep=rep1&type=pdf>.
- Buttram, T. (2003) *DirectX 9 Audio Exposed: Interactive Audio Development*, chap. *Beyond Games: Bringing DirectMusic into the Living Room*.
- Cage, J. & Tudor, D. (1959) *Indeterminacy: New aspect of form in instrumental and electronic music; ninety stories by john cage, with music*. Smithsonian/Folkways Recordings.
- Cardoso, M. (2014) *SuperCollider quark for Thr44 (contains Odef, SCPad, PetriNet)*. [online]. Available from: <https://github.com/Thr44/Thr44-SC-quark> (Accessed 12 August 2018).
- Chan, W.-Y. et al. (2010) Visualizing the semantic structure in classical music works. *IEEE transactions on visualization and computer graphics*. [Online] 16 (1), 161–173. [online]. Available from: <http://dx.doi.org/10.1109/TVCG.2009.63>.
- Chanan, M. (1995) *Repeated Takes: a Short History of Recording and its Effects on Music*. London: Verso.
- Chen, M. (2003) *Petri Nets*. [online]. Available from: <http://www.techfak.uni-bielefeld.de/~mchen/BioPNML/Intro/pnfaq.html> (Accessed 29 March 2021).
- Church, L. & Green, T. (2008) 'Cognitive Dimensions - a short tutorial', in *Proceedings of the 20th annual workshop of the psychology of programming interest group, PPIG 2008, lancaster, UK, september 10-12, 2008*. 2008 Psychology of Programming Interest Group.
- Collins, D. (2005) A synthesis process model of creative thinking in music composition. *Psychology of Music*. 33 (2), 193–216.
- Collins, D. (2007) Real-time tracking of the creative music composition process. *Digital Creativity*. 18 (4), 239–256.
- Collins, K. (2008) *Game Sound: An Introduction to the History, Theory, and Practice of Video Game Music and Sound Design*. Cambridge, MA: MIT Press.
- Collins, R. (1994) Why the social sciences won't become high-consensus, rapid-discovery science. *Sociological Forum*. [Online] 9 (2), 155–177. [online]. Available from: <http://link.springer>.

- com/article/10.1007/BF01476360.
- Cone, E. T. (1968) *Musical Form and Musical Performance*. New York: W. W. Norton.
- Cooper, A. et al. (2014) *About Face: The Essentials of Interaction Design*. 4th edition. John Wiley & Sons.
- Cooper, A. (2004) *The Inmates are Running the Asylum: Why High-tech Products Drive Us Crazy and How to Restore the Sanity*. 2nd edition. Sams Publishing.
- Csikszentmihalyi, M. (2014) 'Society, culture, and person: A systems view of creativity', in *The systems model of creativity*. Springer. pp. 47–61.
- Cycling '74 (2019) *Max 8*. [online]. Available from: <https://cycling74.com/products/max/> (Accessed 20 November 2019).
- Cycling '74 (2020) *Max for live*. [online]. Available from: <https://cycling74.com/products/maxforlive> (Accessed 29 March 2021).
- Dannenberg, R. B. (1993) Music Representation Issues, Techniques, and Systems. *Computer Music Journal*. 17 (3), 20–30.
- Davis, P. & Letz, S. (2019) *Jack audio connection kit*. [online]. Available from: <http://jackaudio.org/> (Accessed 29 September 2020).
- DeMarco, T. (1979) 'Structure analysis and system specification', in *Pioneers and their contributions to software engineering*. Springer. pp. 255–288.
- Desain, P. & Honing, H. (1993) 'Tempo curves considered harmful', in J D Kramer (ed.) *Time in contemporary musical thought*. Contemporary Music Review. pp. 123–138.
- Diaz-Jerez, G. (2012) *FractMusic*. [online]. Available from: <http://www.gustavodiazjerez.com/?cat=14> (Accessed 15 September 2017).
- Dourish, P. (2003) The appropriation of interactive technologies: Some lessons from placeless documents. *Computer Supported Cooperative Work (CSCW)*. 12 (4), 465–490.
- Duignan, M. et al. (2005) 'A taxonomy of sequencer user-interfaces', in *International computer music conference (ICMC)*. 2005
- Duignan, M. et al. (2010) Abstraction and Activity in Computer-Mediated Music Production. *Computer Music Journal*. 34 (4), 22–33.
- Duignan, M. (2008) Computer mediated music production: A study of abstraction and activity. PhD thesis thesis. Victoria University of Wellington.
- Duignan, M. et al. (2004) 'Metaphors for electronic music production in Reason and Live', in *Asia-pacific conference on computer human interaction*. 2004 Springer. pp. 111–120.
- Eisenberg, E. (2005) *The Recording Angel: Music, Records and Culture from Aristotle to Zappa*. 2nd edition. New Haven: Yale University Press.
- Eno, B. (1996) *Generative music 1*. Bracknell, Berkshire.: SSEYO Ltd.
- Eppinger, S. D. (1991) Model-based approaches to managing concurrent engineering. *Journal of Engineering Design*. 2 (4), 283–290.
- Essl, K. (2010) *Lexikon-Sonate*. [online]. Available from: <http://www.essl.at/works/Lexikon-Sonate.html> (Accessed 12 August 2018).
- Exarchos, M. (2019) 'Producing music: Perspectives on music production', in New York: Routledge.
- Fagan, M. E. (2001) 'Advances in software inspections', in *Pioneers and their contributions to software engineering*. Springer. pp. 335–360.
- Farnell, A. (2008) *Pure Data workshop: Make Art 2008*. [online]. Available from: <http://makeart.goto10.org/2008/?page=workshop> (Accessed 10 February 2014).
- FAST IMPACt Project (2015) *FAST: Fusing audio and semantic technologies for intelligent music production and consumption* [online]. Available from: <http://www.semanticaudio.ac.uk/> (Accessed 15 August 2019).
- Fernández, J. D. & Vico, F. (2013) AI methods in algorithmic composition: A comprehensive survey. *Journal of Artificial Intelligence Research*. 513–582. [online]. Available from: <http://www.jair.org/papers/paper3908.html>.
- Fetterman, W. (2012) *John cage's theatre pieces*. Routledge.

- Fiebrink, R. et al. (2011) 'Human model evaluation in interactive supervised learning', in *Proceedings of the SIGCHI conference on human factors in computing systems*. CHI '11. May 2011 New York, NY, USA: Association for Computing Machinery. pp. 147–156.
- Fiebrink, R. (2015) *Wekinator*. [online]. Available from: <http://www.wekinator.org/> (Accessed 4 May 2021).
- Fiebrink, R. A. (2011) *Real-time human interaction with supervised learning algorithms for music composition and performance*. Citeseer.
- Fischer, G. et al. (2004) Meta-design: A manifesto for end-user development. *Communications of the ACM*. 47 (9), 33–37.
- Frayling, C. (1993) *Research in art and design*. 1. Vol. 1. London: Royal College of Art.
- Frith, S. (1996) *Performing Rites: on the Value of Popular Music*. Oxford: Oxford University Press.
- Gaver, W. (2012) 'What Should We Expect from Research Through Design?', in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '12. [Online]. 2012 New York, NY, USA: ACM. pp. 937–946. [online]. Available from: <http://doi.acm.org/10.1145/2207676.2208538>.
- Gessler, N. (1998) 'Skeuomorphs and cultural algorithms', in *Evolutionary programming VII*. 1998 Springer Berlin Heidelberg. pp. 229–238.
- Gracyk, T. (1996) *Rhythm and Noise: An Aesthetics of Rock*. Duke University Press.
- Green, P. & Wei-Haas, L. (1985) *The Wizard of Oz: a tool for rapid development of user interfaces*. [online]. Available from: <https://deepblue.lib.umich.edu/bitstream/handle/2027.42/174/71952.0001.001.pdf?sequence=2&isAllowed=y>.
- Green, T. R. G. (1989) Cognitive Dimensions of Notations A Sutcliffe & L Macaulay (eds.). *People and Computers V*. 443–460.
- Green, T. R. & Petre, M. (1996) Usability Analysis of Visual Programming Environments: a 'cognitive dimensions' framework. *Journal of Visual Languages and Computing*. 7131–174.
- Green, T. & Blackwell, A. (1998) 'Cognitive Dimensions of Information Artefacts: a tutorial', in *BCS HCI Conference*. October 1998 [online]. Available from: <http://iihm.imag.fr/blanch/ens/2010-2011/M1/EIHM/cours/1998-Green-CognitiveDimensions.pdf>.
- Gündüz, G. & Gündüz, U. (2005) The mathematical analysis of the structure of some songs. *Physica A: Statistical Mechanics and its Applications*. [Online] 357 (3–4), 565–592. [online]. Available from: <http://www.sciencedirect.com/science/article/pii/S0378437105003547>.
- Haus, G. & Sametti, A. (1991) Scoresynth: a system for the synthesis of music scores based on Petri nets and a music algebra. *Computer*. [Online] 24 (7), 56–60. [online]. Available from: <http://dx.doi.org/10.1109/2.84837>.
- Hedges, S. A. (1978) Dice Music in the Eighteenth Century. *Music & letters*. 59 (2), 180–187.
- Helo, P. T. (2006) Product configuration analysis with design structure matrix. *Industrial Management & Data Systems*. 106 (7), 997–1011.
- Holland, S. et al. (2002) Applying Direct Combination to afford spontaneity in pervasive computing. *International Journal of Ad Hoc and Ubiquitous Computing*. [online]. Available from: http://mcl.open.ac.uk/sh/uploads/Spontaneous_TR.pdf.
- Holland, S. & Oppenheim, D. (1999) 'Direct Combination', in *CHI '99*. 1999 Pittsburgh: ACM. pp. 262–269.
- Hollingsed, T. & Novick, D. G. (2007) 'Usability inspection methods after 15 years of research and practice', in *Proceedings of the 25th annual ACM international conference on design of communication*. SIGDOC '07. October 2007 New York, NY, USA: Association for Computing Machinery. pp. 249–255.
- Honing, H. (1993) Issues on the representation of time and structure in music. *Contemporary Music Review*. 9 (1–2), 221–238.
- Horn, D. (2000) 'Some Thoughts on the Work in Popular Music', in *The Musical Work: Reality or Invention?* Liverpool: Liverpool University Press.
- Hungerford, B. C. et al. (2004) Reviewing software diagrams: A cognitive study. *IEEE Transactions*

- on *Software Engineering*. 30 (2), 82–96.
- Hunt, S. et al. (2018) ‘A cognitive dimensions approach for the design of an interactive generative score editor’, in May 2018 Montréal, Canada: Fourth International Conference on Technologies for Music Notation; Representation. [online]. Available from: <http://eprints.uwe.ac.uk/36321>.
- Huron, D. (2002) Music Information Processing Using the Humdrum Toolkit: Concepts, Examples, and Lessons. *Computer Music Journal*. [Online] 26 (2), 11–26. [online]. Available from: <http://dx.doi.org/10.1162/014892602760137158>.
- Ignelzi, M. & Rosato, P. (1998) Signification in music: between structure and process a new role for paradigmatic analysis. *Contemporary Music Review*. [Online] 17 (1), 39–55. [online]. Available from: <http://dx.doi.org/10.1080/07494469800640021>.
- Intermorphic (2015a) *Mixtikl* 7. [online]. Available from: <https://intermorphic.com/archive/mixtikl/7/guide/> (Accessed 12 August 2018).
- Intermorphic (2015b) *Noatikl* 3. [online]. Available from: <https://intermorphic.com/archive/noatikl/3/> (Accessed 12 August 2018).
- Izhaki, R. (2012) *Mixing Audio: Concepts, Practices and Tools*. 2nd edition. Focal Press.
- Jacob, B. L. (1996) Algorithmic Composition as a model of creativity. *Organised Sound*. [Online] 1 (03), 157–165.
- Jan, S. (2017) *The memetics of music: A neo-darwinian view of musical structure and culture*. Routledge.
- Jeffries, R. et al. (1991) ‘User Interface Evaluation in the Real World: A Comparison of Four Techniques’, in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’91. [Online]. 1991 New York, NY, USA: ACM. pp. 119–124. [online]. Available from: <http://doi.acm.org/10.1145/108844.108862>.
- Jefferies, C. (2010) *Cylob Music System*. [online]. Available from: <http://durftal.com/cms/cylobmusicsystem.html> (Accessed 15 September 2017).
- Jensen, F. V. et al. (2006) Sequential influence diagrams: A unified asymmetry framework. *International journal of approximate reasoning: official publication of the North American Fuzzy Information Processing Society*. [Online] 42 (1–2), 101–118. [online]. Available from: <http://www.sciencedirect.com/science/article/pii/S0888613X05000678>.
- Johnson, S. (2015) *How we got to now: Six innovations that made the modern world*. Riverhead Books.
- Katz, M. (2004) *Capturing Sound: How Technology has Changed Music*. Berkeley: University of California Press.
- Katzan, H. (1976) *Systems design and documentation; an introduction to the HIPO method*.
- Keller, R. (2019) *Impro-Visor*. [online]. Available from: <https://www.cs.hmc.edu/~keller/jazz/improvisor/> (Accessed 29 March 2021).
- Kelley, J. F. (1983) ‘An empirical methodology for writing user-friendly natural language computer applications’, in *Proceedings of the SIGCHI conference on human factors in computing systems*. 1983 ACM. pp. 193–196.
- Kessell, A. M. & Tversky, B. (2008) ‘Cognitive Methods for Visualizing Space, Time, and Agents’, in G Stapleton et al. (eds.) *Diagrams 2008*. Berlin Springer-Verlag. pp. 382–384.
- Kimball, R. & Harslem, B. V. E. (1982) Designing the star user interface. *Byte*. 7 (1982), 242–282.
- Kindermann, L. (2006) *MusiNum*. [online]. Available from: <http://www.reglos.de/musinum/> (Accessed 15 September 2017).
- Knowles, J. D. & Hewitt, D. (2012) Performance recordivity: Studio music in a live context. *Journal on the Art of Record Production*.
- Ko, A. J. et al. (2004) ‘Six learning barriers in End-User programming systems’, in *2004 IEEE symposium on visual languages - human centric computing*. September 2004 ieeexplore.ieee.org. pp. 199–206.
- Kuhn, T. S. (1970) *The Structure of Scientific Revolutions*. 2nd edition. University of Chicago Press.
- Kutar, M. et al. (2000) ‘Cognitive Dimensions – An Experience Report’, in A.F.Blackwell & E.Bilotta

- (eds.) *PPIG*. 2000 Psychology of Programming Interest Group. pp. 81–98.
- Leman, M. (2008) *Embodied music cognition and mediation technology*. MIT Press.
- Lerdahl, F. & Jackendoff, R. (1983) *A Generative Theory of Tonal Music*. MIT Press.
- Lewis, C. et al. (1990) 'Testing a walkthrough methodology for theory-based design of walk-up-and-use interfaces', in *Proceedings of the SIGCHI conference on human factors in computing systems*. 1990 ACM. pp. 235–242.
- Li, J. & Dey, S. (2013) *Wizard of Oz in Human Computer Interaction*. [online]. Available from: http://pages.cpsc.ucalgary.ca/~jiannali/course/wizard_of_oz.html (Accessed 23 May 2018).
- Lieberman, H. et al. (2006) 'End-user development: An emerging paradigm', in *End user development*. Springer. pp. 1–8.
- London, J. (2012) Three Things Linguists Need to Know About Rhythm and Time in Music. *Empirical musicology review: EMR*. 7 (1-2), 5–11.
- Longo, P. (2000) *Introduction to Petri Nets*. [online]. Available from: <http://www.peterlongo.it/Italiano/Informatica/Petri/> (Accessed 14 September 2017).
- Maloney, J. et al. (2010) The scratch programming language and environment. *ACM Transactions on Computing Education*. 10 (4), 1–15.
- Marrington, M. (2016) Paradigms of music software interface design and musical creativity. *Innovation in Music II*. 52.
- Marsden, A. (2005) Generative Structural Representation of Tonal Music. *Journal of New Music Research*. 34 (4), 409–428. [online]. Available from: <http://eprints.lancs.ac.uk/4311/1/JNMR05forEprints.pdf>.
- Martin, J. & McClure, C. L. (1985) *Diagramming techniques for analysts and programmers*. Prentice-Hall Englewood Cliffs, NJ.
- Mazzola, G. & Andreatta, M. (2007) Diagrams, gestures and formulae in music. *Journal of Mathematics & Music. Mathematical and Computational Approaches to Music Theory, Analysis, Composition and Performance*. [Online] 1 (1), 23–46. [online]. Available from: <http://dx.doi.org/10.1080/17459730601137716>.
- McCartney, J. (2002) Rethinking the Computer Music Language: SuperCollider. *Computer Music Journal*. [Online] 26 (4), 61–68. [online]. Available from: <http://dx.doi.org/10.1162/014892602320991383>.
- McCartney, J. (2021) *SuperCollider 3* [online]. Available from: <https://supercollider.github.io/> (Accessed 4 May 2021).
- McLean, A. (2014) 'Making programming languages to dance to: Live coding with tidal', in *Proceedings of the 2nd ACM SIGPLAN international workshop on functional art, music, modeling & design*. 2014 ACM. pp. 63–70.
- McLean, A. & Wiggins, G. (2010) 'Tidal-pattern language for the live coding of music', in *Proceedings of the 7th sound and music computing conference*. 2010
- Media Art Net (2006) *Eno, Brian: Generative Music 1*. [online]. Available from: <http://www.medienkunstnetz.de/works/generative-music-1/> (Accessed 14 September 2017).
- Middleton, J. N. (2004) Musical Algorithms. Northwest Academic Computing Consortium (NWACC) [online]. Available from: <http://musicalgorithms.ewu.edu/index.html> (Accessed 12 June 2016).
- Middleton, R. (2000) 'Work-in-(g) Practice: Configuration of the Popular Music Intertext', in *The Musical Work: Reality or Invention?* Liverpool: Liverpool University Press.
- Miller, G. A. (1956) The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological review*. 63 (2), 81.
- Moggridge, B. (2006) *Designing Interactions*. MIT Press.
- Moher, T. G. et al. (1993) 'Comparing the comprehensibility of textual and graphical programs', in *Empirical studies of programmers: Fifth workshop*. 1993 Ablex, Norwood, NJ. pp. 137–161.
- Mooney, J. (2011) Frameworks and affordances: Understanding the tools of music-making. *Journal of Music, Technology & Education*. 3 (2-3), 141–154.

- Morzenti, A. et al. (1989) 'TRIO, a logic formalism for the specification of real-time systems', in *Real Time 1989*. 1989 pp. 26–30.
- Myers, B. et al. (2000) Past, present, and future of user interface software tools. *ACM Transactions on Computer-Human Interaction (TOCHI)*. 7 (1), 3–28.
- Nakatani, L. H. & Rohrlich, J. A. (1983) 'Soft machines: A philosophy of user-computer interface design', in *Proceedings of the SIGCHI conference on human factors in computing systems*. CHI '83. 1983 New York, NY, USA: ACM. pp. 19–23.
- Nash, C. (2014) 'Manhattan: End-user programming for music', in Caramiaux, B., Tahirolu, K., Fiebrink, R., Tana, A. (ed.) *Proceedings of the International Conference on New Interfaces for Musical Expression*. 2014 Goldsmiths, University of London. pp. 221–226. [online]. Available from: https://www.nime.org/proceedings/2014/nime2014_371.pdf.
- Nash, C. (2012) Supporting virtuosity and flow in computer music. PhD thesis thesis. University of Cambridge.
- Nash, C. (2015) *The cognitive dimensions of music notations*.
- Nash, C. & Blackwell, A. (2014) 'Flow of creative interaction with digital music notations', in K Collins et al. (eds.) *The oxford handbook of interactive audio*. Oxford University Press. pp. 387–404.
- Nash, C. & Blackwell, A. F. (2012) 'Liveness and flow in notation use', in *12th international conference on new interfaces for musical expression*. 2012 University of Michigan, Ann Arbor: NIME.
- Nash, C. & Blackwell, A. F. (2011) 'Tracking virtuosity and flow in computer music', in *ICMC*. 2011 researchgate.net.
- Nielsen, J. (2006) *Progressive disclosure*. [online]. Available from: <https://www.nngroup.com/articles/progressive-disclosure/> (Accessed 20 November 2019).
- Nielsen, J. (2003) *Usability 101: Introduction to usability*. [online]. Available from: http://didattica.uniroma2.it/assets/uploads/corsi/143228/Nielsen_5_articles.doc (Accessed 12 August 2018).
- Nielson, J. (1995) 10 Heuristics for User Interface Design. useit. com [online]. Available from: <http://www.nngroup.com/articles/ten-usability-heuristics/> (Accessed 8 August 2018).
- Nierhaus, G. (2009) *Algorithmic Composition: Paradigms of Automated Music Generation*. Wien; New York: Springer. [online]. Available from: <http://public.eblib.com/EBLPublic/PublicView.do?ptiID=511127>.
- Norman, D. A. (1986) Cognitive engineering. *User centered system design*. 3161.
- Norman, D. A. (1988) *The psychology of everyday things*. Vol. 5. New York: Basic Books.
- Nurani, L. M. (2008) Critical review of ethnographic approach. *Jurnal sosioteknologi*. 7 (14), 441–447.
- Oppenheim, D. V. (1994) 'Slappability: A New Metaphor for Human Computer Interaction', in Matt Smith et al. (eds.) *Music Education: An Artificial Intelligence Approach*. Workshops in computing. Springer London.
- Orlikowski, W. J. & Gash, D. C. (1994) Technological Frames: Making Sense of Information Technology in Organizations. *ACM Transactions on Information and System Security*. [Online] 12 (2), 174–207. [online]. Available from: <http://doi.acm.org/10.1145/196734.196745>.
- Paternò, F. (2013) End user development: Survey of an emerging field for empowering people. *ISRN Software Engineering*. 2013.
- Paternò, F. & Santoro, C. (2019) End-user development for personalizing applications, things, and robots. *International Journal of Human-Computer Studies*.
- Payne, S. J. (1993) Understanding calendar use. *International journal of human-computer interaction*. 8 (2), 83–100.
- Percussa (2013) *Audiocubes*. [online]. Available from: <http://percussa.us/> (Accessed 10 February 2014).
- Percussa (2012) *Improvisor*. [online]. Available from: <http://land.percussa.com/audiocubes-improvisor/> (Accessed 10 February 2014).
- Petre, M. (1995) Why Looking Isn't Always Seeing: Readership Skills and Graphical Programming.

Communications of the ACM. [Online]

- Petri, C. A. (1962) Kommunikation mit automaten. PhD thesis. Mathematisches Institut der Universität Bonn.
- Piekut, B. (2014) Indeterminacy, free improvisation, and the mixed Avant-Garde:: Experimental music in london, 1965–1975. *Journal of the American Musicological Society*. 67 (3), 769–824.
- Polson, P. et al. (1994) ‘Cognitive walkthrough method: A practitioners guide’, in *Usability inspection methods*. pp. 105–140.
- Polson, P. G. et al. (1992) Cognitive walkthroughs: A method for theory-based evaluation of user interfaces. *International Journal of man-machine studies*. 36 (5), 741–773.
- Polson, P. G. & Lewis, C. H. (1990) Theory-based design for easily learned interfaces. *Human-computer interaction*. 5 (2), 191–220.
- Potter, K. (2002) *Four Musical Minimalists: La Monte Young, Terry Riley, Steve Reich, Philip Glass*. Vol. 11. Cambridge University Press.
- Propellerhead Software (2019) *ReWire - technical information*. [online]. Available from: https://www.propellerheads.com/developer/index.cfm?fuseaction=get_article&article=rewiretechinfo (Accessed 20 November 2019).
- Puckette, M. (1997) ‘Pure Data’, in *International computer music conference*. 1997 San Francisco: International Computer Music Association. pp. 224–227. [online]. Available from: https://www.researchgate.net/profile/Miller_Puckette/publication/230554908_Pure_Data/links/577c1cca08aec3b743366f5c/Pure-Data.pdf.
- Ramakrishnan, C. (2002) *Java OSC*. [online]. Available from: <http://www.illposed.com/software/javaosc.html> (Accessed 26 February 2019).
- Raskin, J. (1994) Intuitive equals familiar. *Communications of the ACM*. 37 (9), 17–19.
- Reich, S. & Hartenberger, R. (1980) *Clapping music*. Universal Edition London.
- Repenning, A. et al. (2000) AgentSheets: End-user programmable simulations. *Journal of Artificial Societies and Social Simulation*. 3 (3), 351–358. [online]. Available from: <http://jasss.soc.surrey.ac.uk/3/3/forum/1.html>.
- Repenning, A. et al. (1998) Learn to Communicate and Communicate to Learn. *Journal of Interactive Media in Education*. [Online] 1998 (7), [online]. Available from: <http://jime.open.ac.uk/article/view/1998-7>.
- Repenning, A. & Ioannidou, A. (1997) ‘Behaviour processors: layers between end-users and Java virtual machines’, in *Proceedings of the 1997 IEEE Symposium on Visual Languages*. [Online]. September 1997 pp. 402–409. [online]. Available from: <http://dx.doi.org/10.1109/VL.1997.626611>.
- Repenning, A. & Sumner, T. (1995) Agentsheets: A medium for creating domain-oriented visual languages. *IEEE Computer Society Press*. 28 (3), 17–25. [online]. Available from: http://www-ui.is.s.u-tokyo.ac.jp/~takeo/course/2006/media/papers/agentsheet_ieee95.pdf.
- Resnick, M. et al. (2005) *Design principles for tools to support creative thinking*.
- Resnick, M. et al. (2009) Scratch: Programming for everyone. *Communications of the ACM*. 52 (11), 60–67.
- Reynolds, R. (1965) Indeterminacy: Some considerations. *Perspectives of New Music*. 4 (1), 136.
- Reznor, T. (2011) *Remix*. [online]. Available from: <http://remix.nin.com/> (Accessed 10 February 2014).
- Rink, B. (2017) *Waveform*. [online]. Available from: <https://www.tracktion.com/products/waveform> (Accessed 8 February 2018).
- Rittel, H. W. J. & Webber, M. M. (1973) Dilemmas in a general theory of planning. *Policy Sciences*. [Online] 4 (2), 155–169. [online]. Available from: <http://link.springer.com/article/10.1007/BF01405730>.
- Roads, C. (1988) Introduction to granular synthesis. *Computer Music Journal*. 12 (2), 11–13.
- Roads, C. (2004) *Microsound*. MIT Press, Cambridge.
- Roberts, C. & Wakefield, G. (2016) ‘Live coding the digital audio workstation’, in *Proceedings of the*

- 2nd international conference on live coding. 2016
- Rogers, Y. et al. (2019) *Interaction Design: Beyond Human-Computer Interaction*. 5th edition. John Wiley & Sons.
- Rosenberg, N. (1982) *Inside the black box: Technology and economics*. Cambridge University Press.
- Rumsey, F. & McCormick, T. (2009) *Sound and Recording*. 6th edition. Oxford: Focal Press.
- Russ, M. (2009) *Sound Synthesis and Sampling*. 3rd edition. Focal Press.
- Schenker, H. & Oster, E. (1979) *Free Composition: Volume III of New Musical Theories and Fantasies*. Pendragon Press.
- Scherzinger, M. (2005) Curious Intersections, Uncommon Magic: Steve Reich's It's Gonna Rain. *Current musicology*. 79207–244. [online]. Available from: <http://academiccommons.columbia.edu/catalog/ac:179139>.
- Schürger, T. (2016) *SoundHelix 0.9*. [online]. Available from: <http://www.soundhelix.com/> (Accessed 15 September 2017).
- Sens, O. (2013) *Usine*. [online]. Available from: <http://www.sensomusic.com/usine/> (Accessed 10 February 2014).
- Shapiro, P. & Lee, I. (2000) *Modulations: A History of Electronic Music: Throbbing Words on Sound*. New York: Caipirinha Productions.
- Smith, J. et al. (2009) Constructures: Supporting human ingenuity in software. *Digital Creativity*. 20 (1-2), 79–94.
- Solberg, R. T. (2014) Waiting for the bass to drop: Correlations between intense emotional experiences and production techniques in build-up and drop sections of electronic dance music. *Dancecult: Journal of Electronic Dance Music Culture*. 6 (1), 61–82.
- Sorensen, A. (2010) *Impromptu*. [online]. Available from: <http://impromptu.moso.com.au/> (Accessed 17 July 2020).
- Spicer, M. (2004) (Ac)cumulative form in pop-rock music. *Twentieth-Century Music*. 1 (1), 29–64.
- Spiliotopoulos, K. et al. (2018) A comparative study of skeuomorphic and flat design from a UX perspective. *Multimodal Technologies and Interaction*. 2 (2), 31.
- Stay, J. F. (1976) HIPO and integrated program design. *IBM Systems Journal*. 15 (2), 143–154.
- Steinberg Media Technologies GmbH (2021) *Cubase 11*. [online]. Available from: <https://www.steinberg.net/cubase/> (Accessed 7 November 2021).
- Stell, J. (2004) Music as Metaphysics: Structure and Meaning in Skryabin's Fifth Piano Sonata. *Journal of Musicological Research*. [Online] 23 (1), 1–37. [online]. Available from: <http://dx.doi.org/10.1080/01411890490277007>.
- Sternberg, R. J. (1999) *Handbook of creativity*. Cambridge University Press.
- Synleor (2013) *Harmony Improvisator*. [online]. Available from: <http://www.synleor.com/improvisator.html> (Accessed 14 July 2019).
- Szerlip, P. & Hoover, A. (2012) *Maestro Genesis*. [online]. Available from: <http://maestrogenesis.org/> (Accessed 10 February 2014).
- Tanimoto, S. L. (1990) VIVA: A visual language for image processing. *Journal of Visual Languages & Computing*. 1 (2), 127–139.
- Taruskin, R. (1995) *Text and Act: Essays on Music and Performance*. Oxford: Oxford University Press.
- Tattersall, S. & Knottenbelt, W. (2014) PIPE — The Great Re-Plumbing. PhD thesis thesis. Imperial College London. [online]. Available from: <http://www.doc.ic.ac.uk/teaching/distinguished-projects/2014/s.tattersall%20.pdf>.
- Tchounikine, P. (2017) 'Designing for appropriation: A theoretical account', in *Human-computer interaction*. 2017 Taylor & Francis. pp. 155–195.
- Tingen, P. (2004) Autechre: Recording Electronica. Sound On Sound [online]. Available from: <http://www.soundonsound.com/sos/apr04/articles/autechre.htm> (Accessed 4 September 2019).
- Vercoe, B. (1996) 'Extended csound', in *Proceedings of the international computer music conference*. 1996 International Computer Music Association. pp. 141–142.

- Walker, R. (2011) *Tune Smithy*. [online]. Available from: <http://www.robertinventor.com/software/tunesmithy/music.htm> (Accessed 20 February 2016).
- Wang, G. & Cook, P. R. (2018) *ChucK 1.4*. [online]. Available from: <http://chuck.cs.princeton.edu/> (Accessed 12 August 2018).
- Wattenberg, M. (2002) 'Arc diagrams: visualizing structure in strings', in *IEEE Symposium on Information Visualization, 2002. INFOVIS 2002*. [Online]. 2002 [online]. Available from: <http://dx.doi.org/10.1109/infvis.2002.1173155>.
- Wiggins, G. et al. (1993) A Framework for the Evaluation of Music Representation Systems. *Computer Music Journal*. [Online] 17 (3), 31–42. [online]. Available from: <http://www.jstor.org/stable/3680941?origin=crossref>.
- Wilson, S. et al. (2011) *The SuperCollider Book*. MIT Press.
- Wolfram Research Labs (2005) *WolframTones*. [online]. Available from: <http://tones.wolfram.com/> (Accessed 10 February 2014).
- Wright, M. (2005) Open Sound Control: an enabling technology for musical networking. *Organised Sound*. [Online] 10 (03), 193–200. [online]. Available from: http://journals.cambridge.org/abstract_S1355771805000932.
- Xenakis, I. (1971) *Formalized Music: Thought and Mathematics in Composition*. Indiana University Press.
- Xenakis, I. (1966) The origins of stochastic music 1. *Tempo*. (78), 9–12.
- Yourdon, E. (1989) *Structured walkthroughs*. Yourdon Press.
- Zagorski-Thomas, S. (2014) *The musicology of record production*. Cambridge University Press.
- Zak, A. (2001) *The Poetics of Rock: Cutting Tracks, Making Records*. Berkeley: University of California Press.
- Zappa, F. (1969) *Hot rats*.
- Zimmerman, J. et al. (2010) 'An Analysis and Critique of Research Through Design: Towards a Formalization of a Research Approach', in *Proceedings of the 8th ACM Conference on Designing Interactive Systems*. DIS '10. [Online]. 2010 New York, NY, USA: ACM. pp. 310–319. [online]. Available from: <http://doi.acm.org/10.1145/1858171.1858228>.

Appendix A

User guide for the first user study

This appendix outlines the design of Choosers v1 at the time of the first user study. It is presented as an introduction to Choosers as a new notation which allows non-programmers to create nondeterministic music. An overview of the design and evaluation of Choosers v1 is presented in Chapter 6.

A.1 The play area

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. To begin with we will consider how to add samples to the play area.

A.2 Adding material

A.2.1 Single-lane Choosers

There are two ways to add samples to the play area. The user can select [File > Add sample] to open a file picker, or samples can be dragged from the Finder directly into the play area. Once added, samples are shown as a rectangle containing the name of the sample—see Figure A.1. 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.

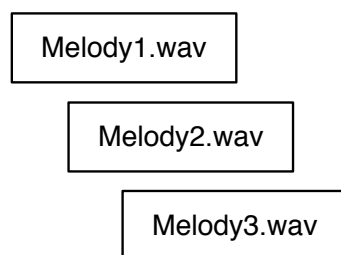


Figure A.1: Three separate samples shown in the main window: each is a single-lane Chooser.

Choosers will contain a sample name if they are added in this way. Optionally, as seen on the right of Figure A.2 they can contain the name of another Chooser.



Figure A.2: Two single-lane Choosers—one containing a sample on the left, and one containing the name of a Chooser on the right.

A.2.2 Multi-lane Choosers

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, as seen in Figure A.3.

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.

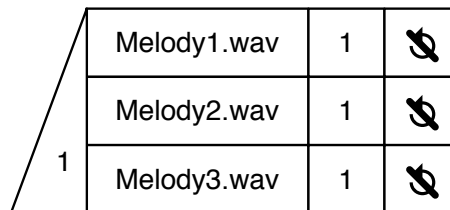


Figure A.3: This Chooser contains playable media; in this instance, samples. As such it is a *Soundable Chooser* and has a nose cone which slopes downwards.

The anatomy of a multi-lane Chooser is shown in Figure A.4. The user can opt to name a Chooser to allow it to be nested inside another Chooser (explained later in this guide). A Chooser which has been named has a name tab. This is optional: unnamed Choosers do not have a name tag.

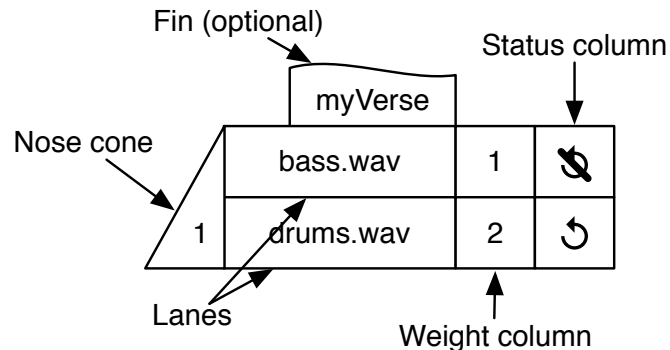


Figure A.4: Anatomy of a Chooser.

A.2.3 Exploring the parameters of the Chooser

Here is an example. Figure A.5 shows a three-lane Soundable Chooser. If you want the software to randomly select one lane you would change the number in the nose cone to 1.

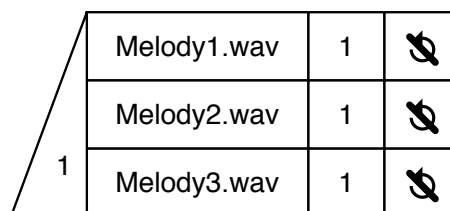


Figure A.5: One lane to be selected.

Note that the weight of each lane is a 1, meaning that they are equally likely to be picked. If, as shown in Figure A.6, the weight of the uppermost lane is changed to 2 it is now more likely to be picked—the likelihood has increased from 1 in 3 (33%) to 2 in 4 (50%).

1	Melody1.wav	2	⊘
	Melody2.wav	1	⊘
	Melody3.wav	1	⊘

Figure A.6: One lane to be selected, lane 1 has a weight of 2.

If the nose cone number is 2 then two lanes will be selected and played together—see Figure A.7.

2	Melody1.wav	2	⊘
	Melody2.wav	1	⊘
	Melody3.wav	1	⊘

Figure A.7: Two lanes to play concurrently.

If the nose cone is set to 3 then all three lanes will be played together. In this particular case the weighting of the lanes does not matter as all three will be played regardless of weight—see Figure A.8.

3	Melody1.wav	2	⊘
	Melody2.wav	1	⊘
	Melody3.wav	1	⊘

Figure A.8: All three lanes will be selected and the weight of the lanes will not be a factor.

A lane can have a weight of zero, meaning that it will never be selected for playback (see Figure A.9). This is a useful way of trying arrangement ideas without removing the lane altogether.

The nose cone can be set to zero, as seen in Figure A.10, which will result in no playback. A Chooser which does not play will be skipped. This can be used to create some interesting arrangements, and will be fully explored later in the guide.

A.2.4 Nesting Choosers

One Chooser can be nested inside another Chooser. Figure A.11 shows a nested child Chooser inside the third lane of a parent 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.

A.2.5 Constraining the duration of a Chooser

As described above, a Chooser will play back the chosen sample. We will now consider a method for adding, and in some circumstances manipulating, duration information to a Chooser.

1	Melody1.wav	0	⊘
	Melody2.wav	1	⊘
	Melody3.wav	1	⊘

Figure A.9: One lane with a weight of zero, meaning that the lane’s contents will not be selected.

0	Melody1.wav	1	⊘
	Melody2.wav	1	⊘
	Melody3.wav	1	⊘

Figure A.10: A Chooser with zero in the nose cone. This means the entire Chooser will not run.

1	Melody1.wav		2	⊘
	Melody2.wav		1	⊘
	1	Melody 3.wav	4	⊘
		Melody 4.wav	3	⊘

Figure A.11: A nested Chooser.

A time lane (see Figure A.12) is a lane which contains a musical duration. A time lane can be included in a **Time Chooser**, as shown in Figure A.13. Note that the Time Chooser has a nose cone which is visually distinct from the Soundable Chooser. Other than the orientation of the nose cone its function is identical.

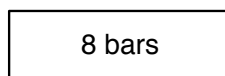


Figure A.12: A time lane.

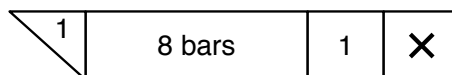


Figure A.13: A Time Chooser.

A.2.5.1 Adding the Time Chooser to a Soundable Chooser

The Time Chooser can be added to the bottom of a Soundable Chooser—see the example in Figure A.14. Note how the nose cones fit together. The duration of the Soundable Chooser will now be controlled by the Time Chooser. Together the two types of Chooser can be referred to as a Full Chooser, or a Chooser for short.

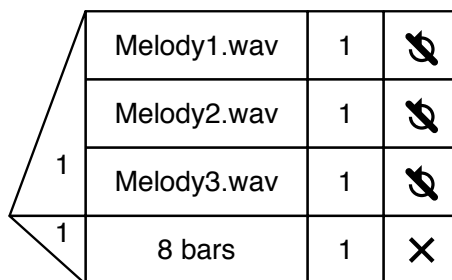


Figure A.14: A Soundable Chooser containing three samples and a Time Chooser containing a time lane with a duration of eight bars.

The Time Chooser contains a time lane set to an exact musical duration. The lanes in the Soundable Chooser will play until the time lane in the Time Chooser has reached the end of its duration. You will see later that time lanes contain more information, such as time signature and tempo, and so the duration imposed will be the duration as judged by the time lane.

Once the duration has been reached the lanes in the Soundable Chooser will be stopped using either a hard or soft stop, as set by the user in the status column of the Time Chooser. If loop is enabled in the status column of the Soundable Chooser lanes any sample shorter than the duration of the Time Chooser will be looped until the time has finished.

A.2.5.2 Status column

The status column allows you to control whether content will loop, and how Choosers will stop files.

The status column on a Soundable lane Lanes in Soundable Choosers use the status column to denote whether the contents of the lane will loop while the Chooser is playing. If the status column shows a crossed-out circular loop icon it means that the lane will play once and then stop. If the circular loop icon is shown in the status column the contents of the lane will continue to play until the Chooser finishes.

The status column on a time lane As has already been seen, the Time Chooser is used to stop a Soundable Chooser. When a Time Chooser stops it will stop any lanes that are currently playing. The status column on a time lane is used to make all other lanes stop immediately (a hard stop, shown in Figure A.16) or it will allow all other lanes to complete the currently playing file before stopping (a soft stop, as shown in Figure A.15).

1	Melody1.wav	1	🔇
	Melody2.wav	1	🔇
	Melody3.wav	1	🔇
1	8 bars	1	>

Figure A.15: In this example one of three samples will be selected for playback. The samples will not loop, meaning that the chosen sample will play once and then stop. The time lane is set to a soft stop (the > symbol in the status column). The Chooser will run for an 8 bar duration, and if it is still playing after 8 bars the sample will be allowed to finish before the Chooser is released.

1	Melody1.wav	1	🔄
	Melody2.wav	1	🔄
	Melody3.wav	1	🔄
1	8 bars	1	✖

Figure A.16: In this example the chosen sample will loop until the duration of the time lane has completed. The time lane is set to a hard stop (the X icon). Once the duration is complete the sample that is playing will be stopped immediately.

A.2.5.3 Nose cone combinations

The nose cones of Soundable and Time Choosers can be used in various combinations. Examples of this are shown in Figures A.17, A.18, A.19, A.20. It is possible for a Chooser to have more than one time, and for the Chooser to select one of them when run. This will be covered later in the guide.

1	Melody1.wav	1	🔇
	Melody2.wav	1	🔇
	Melody3.wav	1	🔇
1	8 bars	1	✖

Figure A.17: The Chooser will play for 8 bars, after which the playing sample (if it has not already finished) will be stopped using a hard stop.

1 0	Melody1.wav	1	⊘
	Melody2.wav	1	⊘
	Melody3.wav	1	⊘
	8 bars	1	X

Figure A.18: The time nose cone has been set to zero, meaning that the Soundable Chooser will run as though the Time Chooser is not there. One sample will play for it's length and the Chooser will be released.

0 1	Melody1.wav	1	⊘
	Melody2.wav	1	⊘
	Melody3.wav	1	⊘
	8 bars	1	X

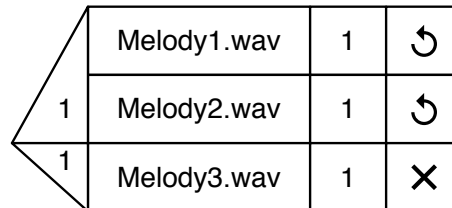
Figure A.19: The Soundable Chooser's nose cone is set to zero, meaning that none of the samples will be selected. The Time Chooser's nose cone is set to 1, meaning that the duration shown in the time lane will run. This Chooser therefore runs silently for 8 bars, creating an 8 bar rest.

0 0	Melody1.wav	1	⊘
	Melody2.wav	1	⊘
	Melody3.wav	1	⊘
	8 bars	1	X

Figure A.20: In this example both Soundable and Time Choosers have their nose cones set to zero. This means that the entire Chooser is skipped.

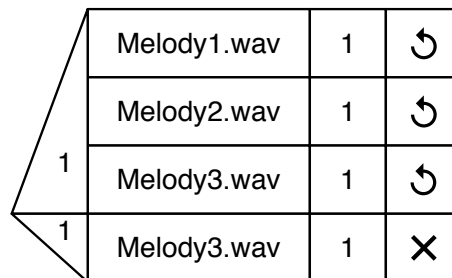
A.2.5.4 Using a sample to set the duration of a Chooser

If you want a sample to set the duration of the Chooser it can be added to the Time Chooser. It is important to note that any sample added to the Time Chooser will not be audible; instead its duration is used to control the time. An example of this is shown in Figure A.21. If you want a sample to be both audible and used to set the duration of the Chooser you will need to add it to both the Soundable and Time Choosers, as shown in Figure A.22.



1	Melody1.wav	1	↺
	Melody2.wav	1	↺
1	Melody3.wav	1	✕

Figure A.21: Making Melody 3 the time. Either Melody1.wav or Melody2.wav will be selected to play, and will loop until the duration of Melody3.wav has elapsed.

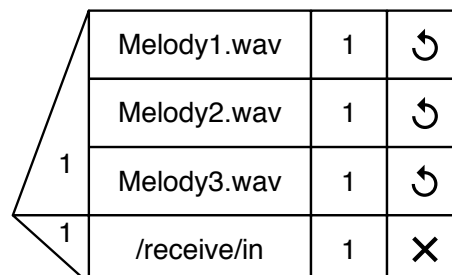


1	Melody1.wav	1	↺
	Melody2.wav	1	↺
	Melody3.wav	1	↺
1	Melody3.wav	1	✕

Figure A.22: This is the same as the previous example except that Melody3.wav can also be selected by the Soundable Chooser.

A.2.5.5 Using an external input to constrain Chooser duration

An external input can be added as a lane in a Time Chooser—see Figure A.23. The Chooser will be halted (using a hard or soft stop) when the external input is received. We will see later in this document that Choosers can be used in a sequence. If there is another Chooser sequenced to occur after the current Chooser the external input will both stop the current Chooser and start the next one.



1	Melody1.wav	1	↺
	Melody2.wav	1	↺
	Melody3.wav	1	↺
1	/receive/in	1	✕

Figure A.23: A lane containing an external input message in the Time Chooser. The status column in the Soundable Chooser shows that the lanes containing samples will loop until stopped. Once the external input is received the Chooser will stop. Note that the time lane is set to a hard stop, and so the currently playing sample will be stopped immediately.

A.2.5.6 Using an external input as a start button

Figure A.23 shows how an external input can be used to stop a Chooser. If a single-lane Chooser is created with an external input it becomes a start button; if a Chooser is sequenced to play next, as shown in Figure A.24, it will play once the external trigger is received. This is because the single-lane Chooser has an effective duration of zero: it starts when the external input is received and then immediately stops, triggering the next Chooser in the sequence. Sequencing will be covered later in the guide.

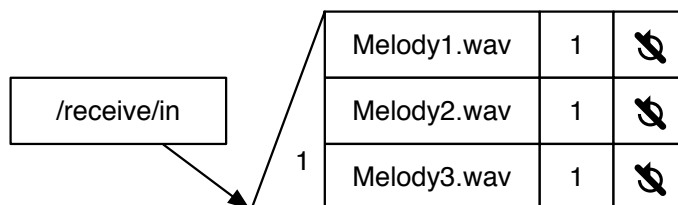


Figure A.24: A single-lane Chooser containing an external input will start the playback of a subsequent Chooser once the external input is received.

A.2.6 Time Chooser with multiple lanes

So far we have used a Time Chooser with one lane, but Time Choosers can contain multiple lanes. An example is shown in Figure A.25. One of the key differences between Soundable Choosers and Time Choosers is that the nose cone of a Time Chooser can only be set to either zero or one.

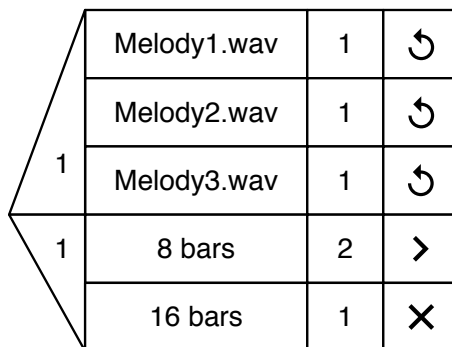


Figure A.25: In this example there are two lanes in the Time Chooser. If the uppermost lane is selected the Soundable Chooser will run for 8 bars and the sample will be allowed to complete before the Chooser is released (a soft stop). If the lower lane is selected the Soundable Chooser will run for 16 bars before being immediately stopped (a hard stop).

A.2.7 Alternative to nesting via named Choosers

An alternative to nesting a Chooser is to reference a Chooser's name inside a lane of the parent Chooser. The example in Figure A.26 behaves the same as the previous example. 'myLength' is the child and is nested inside the leftmost Chooser just as before. This notation can be preferable in some situations, and the user is free to use either notation in any combination.

A.2.8 Forcing lanes to always play

Sometimes it is desirable to ensure that a Chooser lane is always selected for playback. We have seen that the time lane is always picked. In addition, any lane with 'A' in the weight column will always be selected and played. The nose cone is constrained by the number of mandatory lanes (set

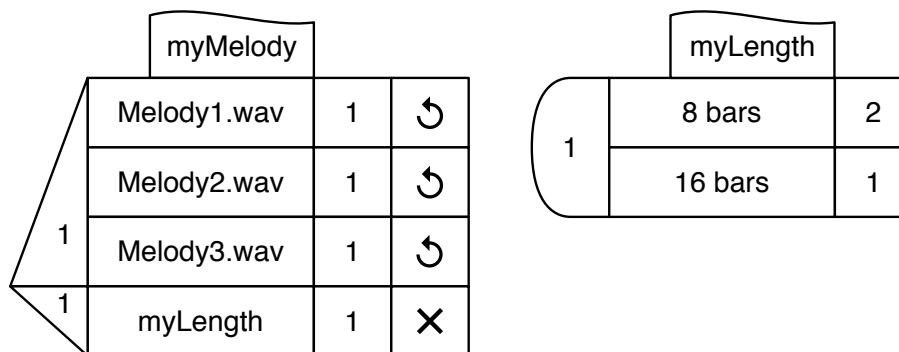


Figure A.26: A variable called myLength called inside the Chooser myMelody.

using 'A' for 'always play'). In the example in Figure A.27 the nose cone of the Soundable Chooser can only be set to 3 or 0.



Figure A.27: A musical section showing that all three playable lanes will be selected for playback. The section has an associated Time Chooser meaning that the section will be exactly 16 bars long.

Choosers can be used to play material together (concurrent playback) or to select one or more of a number of options (alternatives). They allow for Choosers to contain other Choosers (nesting). They also contain a 'repeat until' function which loops a sample until a given musical duration has elapsed, until another named sample has finished playing, or until an external input is received.

A.3 Creating a sequence

Sequencing in music is essential: we need a way of creating sections and order.

A.3.1 Sequence

A sequence is shown by an arrow. The direction of the arrow denotes the order of the sequence. In the example shown in Figure A.28 the first Chooser will play and, once it has finished playing, the second Chooser will begin.

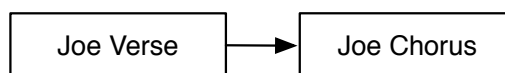


Figure A.28: Arrow showing sequence.

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. The system will show any reference without a Chooser of the same name as a dangling reference. This process is explained in more detail later in the guide.

A.3.2 Top-down or bottom-up

The software is designed to allow 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. In order to make this clear we will consider two different ways of working. The first example will show a top-down approach in which the user begins with the musical structure. The second example will show the user starting at the note level and assembling the piece from these smaller fragments.

A.3.3 Arrangement example - top down

The macro arrangement of a piece can be created using single-lane Choosers as shown in Figure A.29. Movement from one section to another is shown by a horizontal arrow. Repeats are shown by adding the appropriate notation underneath the relevant block ('x2' to repeat twice, for example).

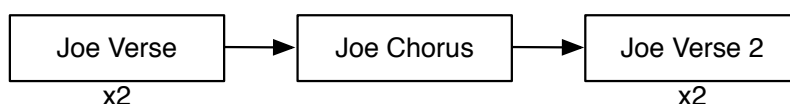


Figure A.29: Macro arrangement.

A.3.3.1 Adding durations to arrangement sections

If the user wishes to add duration information to the arrangement at this stage they can do so by making use of named and referenced Choosers and time lanes, as shown in Figure A.30.

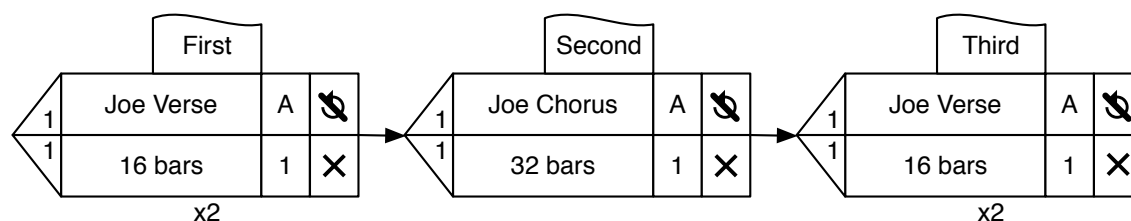


Figure A.30: The macro arrangement with duration information for each section.

A.3.3.2 Defining the musical sections

The user may now wish to define the musical sections they have created. To do this they can either nest Choosers inside the existing Choosers, or they can create new Choosers and reference them using their names. The two examples above referenced a Chooser called Joe Verse. This name was being used as a placeholder. The user can now create a Chooser with that name; see Figure A.31 for an example. Each named lane in the Joe Verse Chooser needs to be defined in turn. For example, Figure A.32 shows the 'myDrums' Chooser which will select a drum sample to be played. It is important to note that the duration of the 'Joe Verse' Chooser is set by the parent. The parent Chooser imposes its timing and loop status onto all child Choosers.

In the example above, the parent lane referencing the 'myDrums' Chooser is set to loop. The 'myDrums' Chooser has looping turned off on all lanes. If the drum samples are shorter than the length dictated by the parent time (in this case, 16 bars) then the following sequence of events will take place:

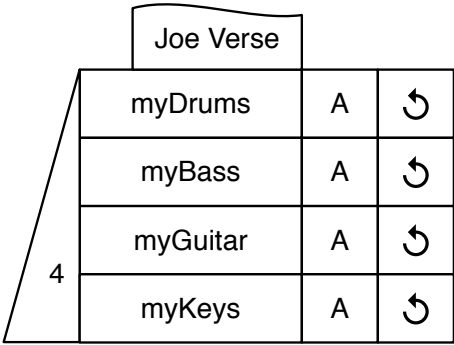


Figure A.31: A new Chooser, Joe Verse, which is used to populate the existing arrangement.

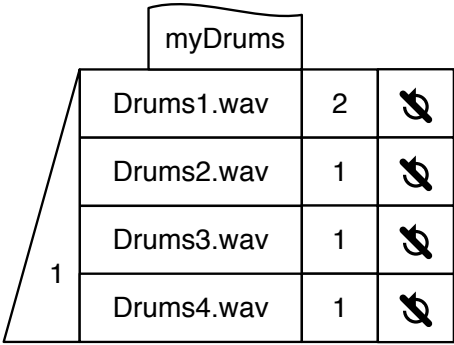


Figure A.32: The myDrums Chooser, which populates the myDrums lane of the Joe Verse Chooser.

1. The ‘Joe Verse’ Chooser will run and trigger four Choosers - ‘myDrums’, ‘myBass’, ‘myGuitar’, and ‘myKeys’. The example focusses on just the first of these.
2. ‘myDrums’ runs, and one of the four samples is selected. As looping is turned off for all lanes, the chosen sample will play once and then stop.
3. The parent Chooser, ‘Joe Verse’, has looping on for the ‘myDrums’ lane. This means that, if the ‘myDrums’ Chooser finishes before the ‘Joe Verse’ Chooser is stopped, the ‘myDrums’ Chooser will be run again with the possibility of a different lane being chosen.
4. ‘Joe Verse’ will be rude-stopped after 16 bars by its parent Chooser. This results in all child Choosers being stopped immediately.

A.3.4 Arrangement example - bottom up

An alternative approach is for the user to create a small musical section and to slowly build up to an entire piece of music. In the example shown in Figure A.33, the user has created a musical fragment that has a duration of 32 bars. Can you see which elements will always play, and which may be played?

The user can then create an arrangement using single-lane Choosers which reference Choosers by name, such as the arrangement shown in Figure A.34. These single-lane Choosers are the parents of the referenced Choosers. Note that the duration of ‘Ideal’ is defined inside the ‘Ideal’ Chooser shown above. If a time lane was added to the parent it would take precedence over the time lane of the child.

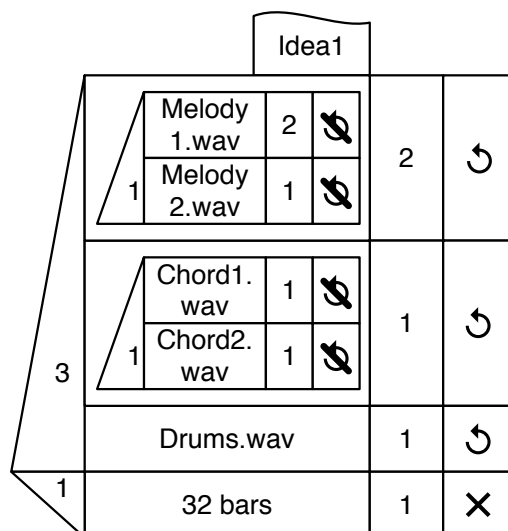


Figure A.33: A musical section, named 'Idea 1'.

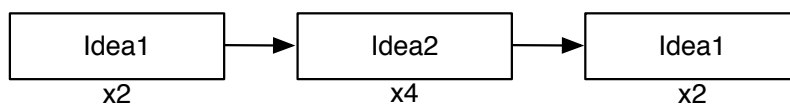


Figure A.34: An arrangement that creates a sequence by referencing Choosers.

A.4 Advanced techniques

A.4.1 Variables

We have already seen how the user can assign a name to a Chooser and then reference the named Chooser from inside another Chooser. This is an example of a variable - by issuing a name to a Chooser we create a variable which can be reused elsewhere. Variables allow the user to easily reuse material by defining it once and then creating multiple references to the single definition. Variables are more flexible than the examples have so far shown as they can contain several different types of information. An important distinction between variable Choosers and the Choosers shown so far is that nose cone of a variable Chooser can be set to 1 or 0; either one lane is picked or the Chooser is not used.

A.4.1.1 What can be in a variable Chooser?

Numbers A variable Chooser can contain numbers. An example is shown later in this guide.

Status A named Chooser can be used to select the status of a playable lane (i.e. loop or non-loop) or of a time lane (i.e. polite, rude or inherit).

Clocks A named Chooser can be used for time lane selection, as shown below.

A.4.1.2 New selection each time

By default, any variable which refers to a Chooser will result in the Chooser being run each time it is called. This may result in a different selection each time it is called - a fresh 'roll of the dice' each time the Chooser is run.

A.4.1.3 Reusing the same selection multiple times - variable history

There are some instances when the user may want to reuse the same result from a variable. For example, if one melody was chosen for a musical section the user may want the same melody to be

used later in the piece. This can be done by appending a number or name—the ID—to the end of the variable name when it is used. Every time the user refers to a variable using the appended name the same result will be returned, allowing for the same output each time.

For example, Figure A.35 shows a Chooser which is given the variable name *xyz*.

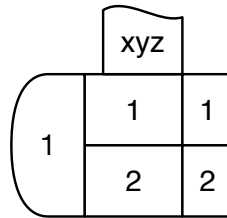


Figure A.35: A variable Chooser called *xyz*.

If the variable name is used without appending a number or name then the variable Chooser will re-run each time it is used; this results in a ‘fresh roll of the die’ and a potentially different result. An example is shown in Figure A.36.

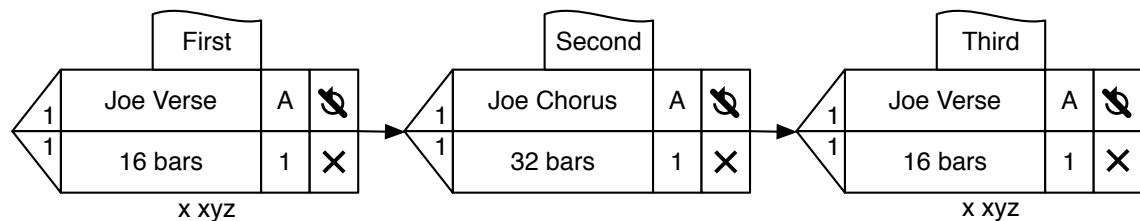


Figure A.36: The variable ‘*xyz*’ is used to control the number of repeats of two Choosers. The variable selection is independent.

If the user would like to make one selection and then use the same selection in multiple places they can optionally add a number or name to the variable name. The format for this is ‘Name@ID’, with the name being the name of the variable (in this case, ‘*xyz*’) and the ID being the number or name given by the user. The two examples shown in Figures A.37, A.38 are functionally identical.

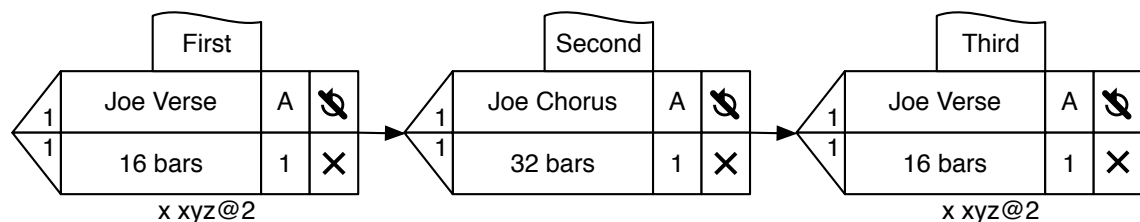


Figure A.37: Both references to the variable Chooser add ‘@2’ to the end of the name, showing that the same selection should be used for both.

A.4.2 The variable list

The variable list is an alphabetical list on the right of the user interface. When a variable is created by adding a name to a Chooser the new variable name is added to the variable list. When a reference is made to a variable that does not yet exist the name is added to the variable list in italics. This allows users to add names as placeholders and to populate them without having to first create the variable. The italicised variable names show the dangling variable references.

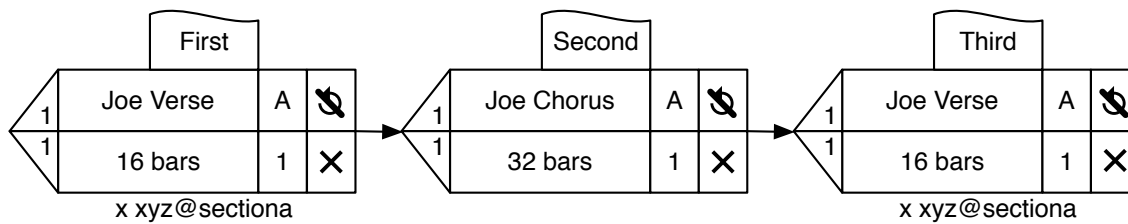


Figure A.38: This example is functionally identical to the previous one. This time the user has chosen to give a more meaningful name to the variable Chooser.

A.4.3 Example: variable selection

An advanced technique is to create a variable Chooser which will select a number. This number can then be used in the nose cone of another Chooser. In the example shown in Figure A.39, the user creates two Choosers, 'myMelody' and 'myLength'.

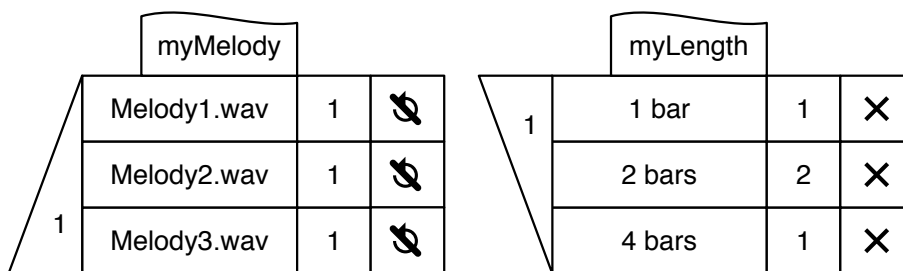


Figure A.39: Two Choosers are created. They are not yet connected.

Figure A.40 shows the result when a new variable Chooser called 'var' is created to control the nose cone value of 'myMelody'.

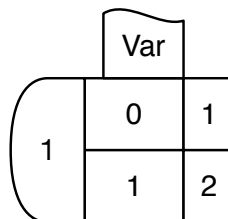


Figure A.40: A Chooser named 'Var' which will select one of two integers.

The user now creates a new Chooser called 'Top', shown in Figure A.41. The Chooser has two lanes, which include references to myMelody and myLength. The Soundable Chooser in the 'Top' Chooser does not have a fixed value in the nose cone. Instead, it refers to the 'var' Chooser. Note that the nose cone will accept this Chooser as valid as the 'var' Chooser will output an integer. Any other type of output would not be valid and would therefore be prevented by the system.

If the 'var' Chooser selects a 1 then both 'myMelody' and 'myLength' will be selected and played. One of three melodies will be selected, and it will play for one of three durations before stopping. If the selected sample is shorter than 'myLength' then once it has finished the loop on the 'myMelody' lane will result in 'myMelody' being re-run. This will continue until the 'Top' Chooser is stopped by the time. As the time is set to a polite stop, the currently running sample will run to completion before the 'Top' Chooser is stopped. If the 'var' Chooser selects a 0 then only the Time Chooser will be selected. There will therefore be a rest of 1, 2 or 4 bars duration. To create a harmonic accompaniment, the user creates the Choosers shown in Figure A.42.

One of 'Harm1.wav' and 'Harm2.wav' will be selected by the 'myChords' Chooser, and then the 'HarmProg' Chooser will play the file for one, two or four bars before stopping. If the duration

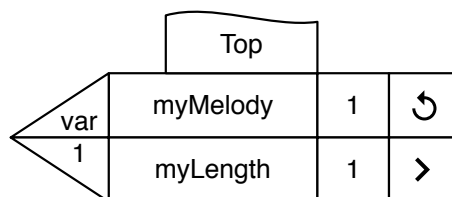


Figure A.41: A Chooser named 'Top' in which the 'Var', 'myMelody' and 'myLength' Choosers have been nested.

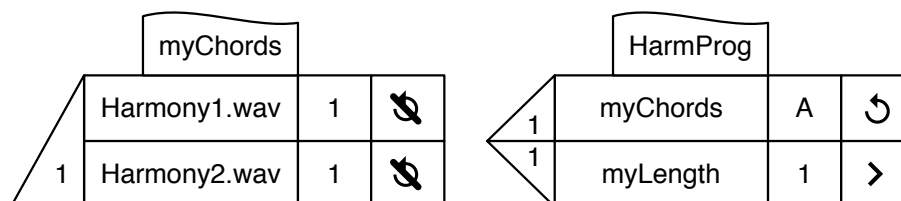


Figure A.42: The creation of Choosers named 'myChords' and 'HarmProg'.

chosen by 'myLength' is longer than the selected sample the 'myChords' Chooser will be run again due to the loop in the 'HarmProg' Chooser. Note that the 'HarmProg' Chooser mandates that both lanes should play (a 1 in each nose cone): 'HarmProg' will not contain any rest periods. Once it is stopped the 'HarmProg' Chooser will allow the currently playing sample to complete due to the time's polite stop setting.

Finally, both 'myMelody' and 'HarmProg' are played together in the final Chooser, named 'Drift', shown in Figure A.43. The nose cones shows that all three lanes of the parent Chooser will play concurrently. The Time Chooser contains a nested Chooser which will make the entire section play for either 64 or 92 bars. 'Melody' has been designed to add rests and so there will not be a constant melodic part.

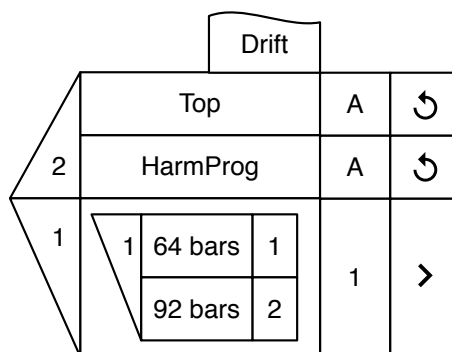


Figure A.43: 'myMelody' and 'HarmProg' are nested inside 'Drift'.

A.4.4 Multi-column variable Choosers

At some point you may want to coordinate variable choices; for example, if the Chooser selects a duration of 16 bars it should also set the status to hard stop and set the previous Chooser to repeat twice. This can be achieved by using a multi-column variable Chooser.

So far the examples have used a variable Chooser with a single column which contains the values to be selected, and a weight column; see Figure A.44 for an example. Additionally, the examples have shown that calling the variable elsewhere (in this instance, using the name 'xyz') will result in a new selection each time. Calling the variable by appending a number or name after an @ (i.e. 'xyz@2' or 'xyz@verse') will result in the same result being used multiple times.

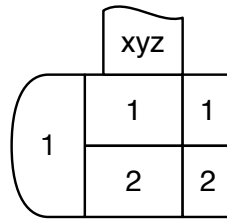


Figure A.44: The variable Chooser xyz.

A multi-lane column adds names to each column. The user is free to give the columns any name they wish; two examples are shown in Figures A.45, A.46.

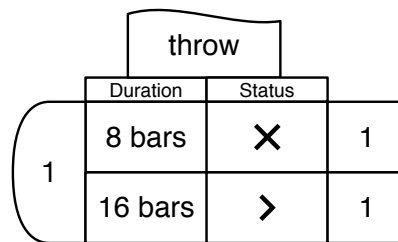


Figure A.45: A multi-lane variable Chooser with descriptive column names.

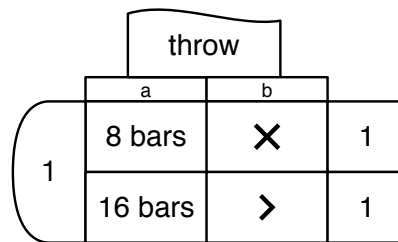


Figure A.46: A multi-lane variable Chooser with non-descriptive column names.

Figures A.47, A.48 show the multi-column variable Chooser in action, using both non-descriptive and descriptive column names. To use a multi-column variable Chooser you need to use the format 'Name: Column name'.

Coordinated multi-column variable Choosers As with single-column variables, it is possible to append a number or name (the ID) to the variable when it is used in order to access the variable history and reuse a selection multiple times. The format for this is 'Name@ID:column'. Importantly, this allows for coordination between the columns of the variable Chooser. Figures A.49, A.50 contain two examples showing the use of an ID to reuse a variable selection.

A.4.5 Zero in the nose cone

The software is designed to allow the user to make quick changes in order to test ideas. In some cases it may be desirable to skip a Chooser. To do this the number in the nose cone can be set to zero.

throw			
1	a	b	
	8 bars	✕	1
2	16 bars	>	1
	Melody1.wav	1	↺
	Melody2.wav	1	↺
1	Melody3.wav	1	↺
	throw@2:a	1	throw @2:b

Figure A.47: In this example the variable is called ‘throw’ and the columns are named ‘a’ and ‘b’. Therefore, the possible options are ‘throw:a’ and ‘throw:b’. Note that each reference to the ‘throw’ variable is independent, and represents a new ‘throw of the die’.

throw			
1	Duration	Status	
	8 bars	✕	1
2	16 bars	>	1
	Melody1.wav	1	↺
	Melody2.wav	1	↺
1	Melody3.wav	1	↺
	throw:Duration	1	throw: Status

Figure A.48: This is a functionally identical example which uses the more descriptive column names. The format remains the same.

throw			
1	a	b	
	8 bars	×	1
	16 bars	>	1
2	Melody1.wav	1	↻
	Melody2.wav	1	↻
	Melody3.wav	1	↻
1	throw@2:a	1	throw @2:b

Figure A.49: Example using non-descriptive column names and an ID. Note that the result of the ‘throw’ variable will result in a duration of 8 bars *and* a hard stop *or* a duration of 16 bars *and* a soft stop.

throw			
1	Duration	Status	
	8 bars	×	1
	16 bars	>	1
2	Melody1.wav	1	↻
	Melody2.wav	1	↻
	Melody3.wav	1	↻
1	throw@verse:Duration	1	throw@ verse:St atus

Figure A.50: Example using descriptive column names and an ID.

Appendix B

Choosers v1 design table

User need	General solution - point to the literature	Chooser feature - implementation
Be able to play multiple musical events simultaneously	Stacked lanes for concurrency – working with DAW expectations if the user has this experience	Stacked lanes; nose cone to set the number of concurrent lanes for playback; weights for weighted nondeterministic choice
Moderating the duration of musical objects is an essential musical move.	Allow the length of the object to be an option (e.g. DAWs). User to be able to impose a duration. Literature - Honing? DAW imposition of time?	Time Chooser; nose cone to turn on/off; operates independently from the Soundable Chooser while imposing duration.
Be able to have nondeterministic duration	One of the key motivations for the project, so important to the work. Progressive disclosure keeps the complexity out (no weights) until more than one lane has been added.	Multiple lanes in the Time Chooser. Nose cone set to either zero or one to avoid multiple duration selection. Weights progressively disclosed when >1 time lane present. Design consistent with Soundable Chooser rules; nose cone, weights etc.
Allow for music content and duration to be separately controllable	Empty	Soundable and Time Choosers work combinatorially but are separately controllable. This allows for each to be disabled: 1/1 Full Chooser; 1/0 infinite playback; 0/1 musical rest; 0/0 no playback, skipped if in a sequence.
Mark some lanes as high priority	Musically useful to say ‘I want this to always play’. Music often has a front line, top line, or genre-specific hierarchy (vocal line in pop, drums and bass in dance music, etc.).	‘Always play’ as a legal lane weight. The number of A-weighted lanes bounds the lower value of the Soundable nose cone.

User need	General solution - point to the literature	Chooser feature - implementation
Create and subsequently edit a sequence	Essential for musical structure.	No more than a single arrow in and out of a box or Chooser. From diagrammatic representations; flow chart etc.
Loop musical material	From tape loops, then sequencers, drum machines etc. Primary building block of electronic dance music, samplers, granular synthesis etc.	Boolean loop on/off per soundable lane.
Reuse of material with either new choices each time it runs, or a repeat of the same choices as a previously run event.	Both valid and desirable musical moves. Example of 1: electronic music or repetitive pop music. Example of 2: the selection of radically different files, or different takes of the same melody line – could be subtle, depending on content. The default is for new choices on each run as that is best fit with the ethos of the system.	Via nesting or referencing. 'Same choices as' via @ notation.
Control over the way in which a file behaves when stopped; stop immediately, or stop when finished.	Empty	Soft stop allows for files to finish their current iteration.
Allow a music event such as an audio sample to control the duration of a Chooser.	Similar to a command issued to musicians - 'watch me for the changes'	Soundable content in a time lane.
Allow interaction with other software and hardware.	Empty	OSC input in a box as a start command. OSC input as a time lane to initiate either hard or soft stop. OSC output in a lane or box to send message out to external software or hardware.

Appendix C

Example user test transcript

This appendix presents an example user test transcript. This particular example is taken from the second round of user tests outlined in Chapter 9. Each user test was videoed and fully transcribed, with each transcript analysed using the programming walkthrough methodology (see Section 3.4). The transcriptions offer an exhaustive list of participant comments on multiple aspects of the design. As outlined in the walkthrough protocol shown in Section 3.5, each transcript was annotated to show the following four types of response. Each response type is marked on the transcript using a single word in square brackets.

- Questions (e.g. why does the loop do that?)—[QUESTION];
- Problems (e.g. I don't understand what these lanes are for)—[PROBLEM];
- Suggestions (e.g. maybe the cone should be a different shape)—[SUGGESTION];
- Other observations (e.g. I like the fins)—[OBSERVATION].

In order to further marshal the information obtained by the programming walkthrough, participant responses were also organised by the themes that emerged 'bottom-up' from participant comments. These themes were identified via the analysis of all transcripts. The four themes that emerged from the second round of user tests, and the annotation used to represent each theme, were:

- The potential use of Choosers as an introduction to nondeterministic music—[NONDET-INTRO]
- Identifying and implementing suitable metaphors for non-programmers—[METAPHOR]
- Issues relating to end-user expectations—[EXPECTATION]
- Possible improvements for design and implementation—[IMPROVEMENT]

These themes are discussed in detail as part of Section 9.3 in Chapter 9. The same chapter outlines the scenarios presented here (Section 9.2). In the following transcript, each scenario is outlined before showing the participant's comments and conversations.

C.1 Scenario 1: simple sequence

The participants were shown a video (Bellingham, 2020b) which introduced the handling of audio samples and the design of lanes and sequences. They were then asked to answer the following two questions and to complete a short task.

- Questions
 - What will happen when the sequence shown in Figure C.1 runs?
 - How could you change the order of the sequence?

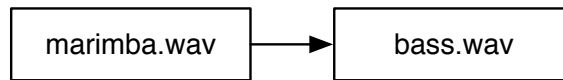


Figure C.1: The simple sequence example shown to participants in scenario 1 as part of the second user study.

- Task
 - Using the whiteboard, update the sequence to play the marimba, then the bass hook, then the marimba again, then stop.

Scenario 1 transcript

Facilitator (F): We'll start by watching a video.

Video voiceover: *The largest central area of the user interface is known as the play area. It is here that the user is able to assemble, audition, and package musical structures. We can drag samples into the play area. Samples are shown in lanes. Anything in a lane can be auditioned by clicking on it. A sequence is shown by arrows connecting lanes or Choosers; the direction of the arrow indicates the order of the sequence. Only a single arrow can enter or exit each element in a sequence. Here we have a simple example showing a sequence of samples; once the first sample has finished playing, the second sample will begin.*

F: Okay, so if I can show you this image (Figure C.1)—what do you think is going to happen?

Participant 1 (P1): Let's have a look.

Participant 2 (P2): It looks like like a percussion sound goes to some type of bass sound.

P1: So we'll play the marimba and then we'll play the bass.

P2: That's what the video says; in sequence.

F: How could you make it play the bass first and then the marimba?

P2: Swap it around and put the arrow the other way.

F: Exactly; you could put the bass here in the marimba there [changing the order of the boxes] and have the arrow going left to right, or you could just turn the arrow around.

P1: It's the direction of the arrow that's important.

P2: You said you can only have one inlet and outlet.

F: That's right.

P2: I have another question; if you wanted to play it together, do they stack them? [QUESTION, EXPECTATION]

F: That's a really good question.

P2: Let's say, for example, marimba and bass, how would you get them to play on top of each other? [QUESTION, EXPECTATION]

F: You can, and we will come back to this; it's a great point and I want to make a note of it [writes note]. Before we do, can you make a quick sequence which plays marimba, then bass, then marimba, then stops. Don't worry about writing [the whole word] marimba, you can just write 'm' and 'b' as a shorthand.

P1: One or both of us?

F: One of you could do the drawing but you should both have some input. The reason we're doing the test in pairs is so that you can talk through each scenario.

The participants successfully completed the task; the whiteboard is shown in Figure C.2.

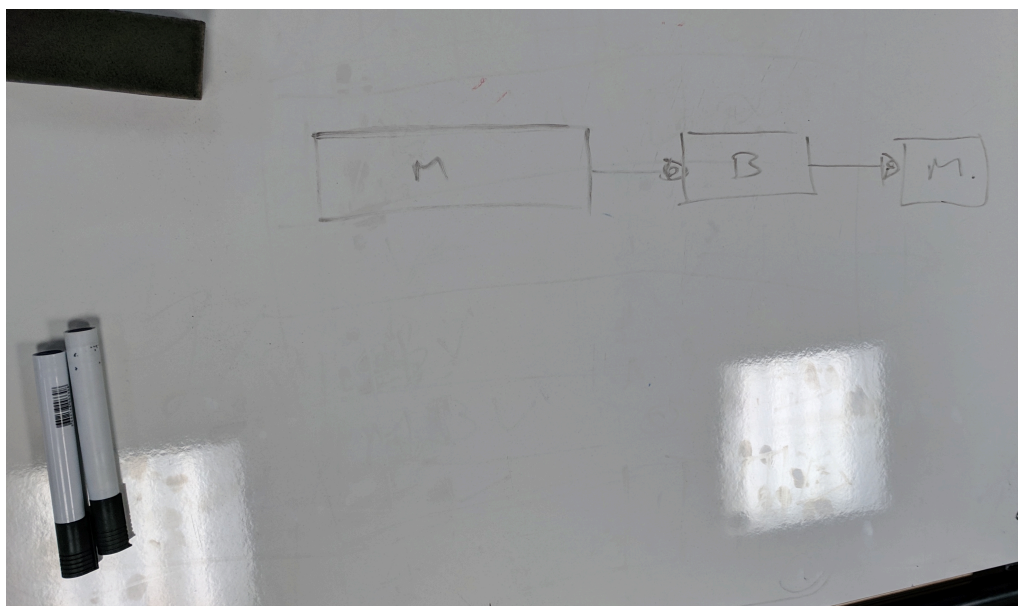


Figure C.2: The result of scenario 1, successfully completed on a whiteboard.

F: Are you both happy with that? So, the arrows give you the sequence order—this, then this, then this—this is obviously the start as there is no arrow coming in, and that one is obviously the end as there is no arrow coming out. I can play you the sequence you have just written.

[Playback of scenario 1]

P2: OK; sequence is arrows.

C.2 Scenario 2: Soundable Choosers

The participants were shown a video (Bellingham, 2020b) which introduced Soundable Choosers including the nose cone, loop/non-loop, and the weight column. After a brief aural introduction to the scenario's focus on choosing between two different samples they were then asked to answer a series of questions to test their understanding, followed by a practical task.

- Questions
 - If the Chooser in Figure C.3 is played by itself, how many samples will play? How do you know?

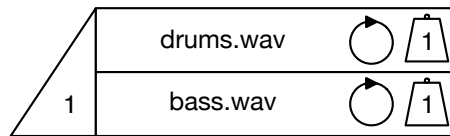


Figure C.3: An image of a Soundable Chooser, used as part of scenario 2 in the second user study.

- 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?
- Task
 - Using the whiteboard, draw a Soundable Chooser for the following scenario:
 - You want a Chooser which will select and play two out of three samples; `drums.wav`, `bass.wav`, and `guitar.wav`. You want the samples to loop, and you want the drums to be twice as likely to be selected as any other sample.

Scenario 2 transcript

F: Here's the next video.

Video voiceover: *If two or more lanes containing samples are snapped together vertically they become a Soundable Chooser. If there are two or more lanes in the Chooser the lane will display a nose cone on the left and additional controls on the right of each lane. The nose cone allows the user to specify how many of the lanes will be played simultaneously. For example, if there are three lanes and the nose cone contains the number two, two lanes will be chosen to play simultaneously. If the nose cone contains the number one, only a single lane will be chosen. Note that every time the Chooser is played the choice of lanes is made afresh non-deterministically; that is, at random. The first control on the right controls loop behaviour; for now we will use this control to either loop the lane or to play once without looping. The weight control shown on the far right determines the relative likelihood of a lane to be selected for playback. Here we have three lanes, each with a weight of one, meaning that they are equally likely to be selected. If one lane's weight was changed to two it would now be more likely to be selected for playback. If a lane is given a weight of zero it will never be selected for playback; this can be a useful way of trying arrangement ideas without removing the lane altogether.*

P2: It's like priority; when you choose to give priority and certain things to play first.
[OBSERVATION, METAPHOR]

F: I don't suppose you can remember where you've come across this same idea in other software?

P2: It might come to me. After 15 years of working with music software ... it just came to me, 'I know that from somewhere'.

F: If you do remember today or in the future please drop me a message as it would be very useful to know where you have encountered the idea of priority before. So, this is a Soundable Chooser; 'soundable' because it's able to play samples and it is going to make a sound when you run it. So, looking at this, what do you think will happen when you run that?

[Facilitator points to the scenario 2 example, shown in Figure C.3]

P2: Only one sound is going to play.

F: How do you know that?

P2: Because of the one in the nose cone.

F: Great; so it's going to pick either this [pointing to one lane] or that [pointing to the other lane].

P2: Because the priority is one each, but there's no loop in either. Does that mean it's going to be selected for looping? [QUESTION, EXPECTATION]

P1: That means it *is* looping, doesn't it?

F: Yes; if it *wasn't* looping that [pointing to the loop icon] would be crossed out, so that means it is looping.

P2: So, whichever it picks at random, it's going to play in a loop.

F: Yes. In terms of choice, is there one that is more likely than the other to be selected at the moment?

P1: I don't know how to determine who plays first. [PROBLEM, NONDET-INTRO]

P2 It's going to pick it at random, as you said in the video, isn't it? There's no high priority, they've got equal priorities.

P1: I don't know how, like, the people who create programs, how they can choose it at random. [QUESTION]

F: That's a great question because you're thinking how random choices are actually made by a deterministic system, which is a system designed to do the same thing every time. They [the lanes] have equal weight, which is the term we use in this context. This means that it's like a coin toss; we don't know which one is going to get selected. So, how could you make the drums more likely than the bass to be selected?

P2: You'd turn the weight up to two.

P1: Yes.

F: So, that could go to a number two, and then this would be twice as likely to be selected than this. Does that seem reasonable?

P2: Is that giving it, like, more probability? Is that what it's doing? [QUESTION, NONDET-INTRO]

F: Yes, exactly that. I'm now going to play you the Chooser exactly as you have written it there.

[The Chooser is run multiple times, with either drums or bass selected each time]

F: It's 50/50 each time. How can we make it so that it plays both simultaneously?

P2: Put number two in the nose cone, yeah, and leave the priority—the weight—at number one.

F: So, what are we saying when we change that [the nose cone] to two?

P2: We're saying two channels can play simultaneously.

F: Pick two and, as there are only two to pick from, it has to pick them both. Right, so that would sound like this ...

[The Chooser is run, this time with the nose cone set to 2]

F: I'm going to ask you to draw a new Chooser and I will play what you draw. I will give you a scenario and I'll keep this up on the screen for reference. Can you draw a Chooser with three samples in it rather than two; those three samples will be drums, bass, and guitar. The drums need to be twice as likely to be selected as the bass and the guitar. The Chooser will play two out of the three samples.

P2: So, bass and guitar are random optionals, drums priority.

F: The drums are more likely to play but they might not be selected.

P2: Okay.

[Participants draw the Chooser on the whiteboard]

F: As a shorthand, feel free to put the initials in the lanes—drums, bass, guitar could be replaced with 'd', 'b', and 'g'. You can just draw a circle for the loop symbol.

P1: You said that drums has to play twice?

F: Drums are twice as likely to be selected.

[The participants successfully completed the task, which is shown in Figure C.4]

F: Great; that is absolutely right. Drums, bass, and guitar are all selectable but we're only going to hear two at a time. We don't know which two as it's going to be a randomised process.

P2: Does that automatically change for every loop? [QUESTION, NONDET-INTRO, EXPECTATION]

F: Not every loop. Once it's started and the selections have been made they will be the same until the Chooser is run again. So, if I play the Chooser multiple times we get different outputs.

[The Chooser is run multiple times]

F: So, the system makes nondeterministic choices which we get to control. Do you have any questions or thoughts at this point?

P1: You do things like this in video games. [OBSERVATION]

F: Absolutely; game sound and game music are often non-linear as you don't know what the player is going to do, so games do have to use some type of dynamic control.

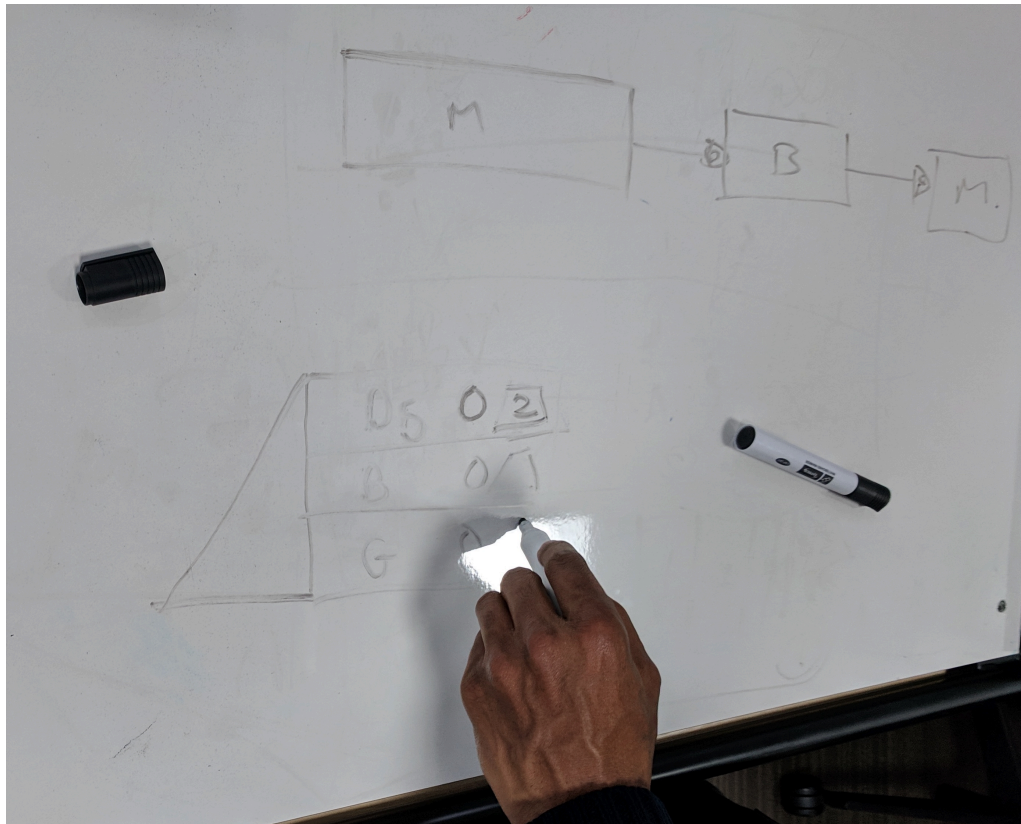


Figure C.4: Scenario 2 being completed. In this instance, P2 is drawing and P1 is making suggestions.







2	drums.wav		
	bass.wav		
	guitar.wav		

Figure C.5: The example used in scenario 3 of the second user study, showing the use of infinite weight.

C.3 Scenario 3: infinite weight

Participants watched a video (Bellingham, 2020b) which introduced infinite weight and the priority seating and lottery analogy. The participants were then presented with the following question and tasks:

- Question
 - What will happen when the Soundable Chooser in Figure C.5 runs?
- Tasks
 - Using the whiteboard, draw a Soundable Chooser for the following scenario:
 - * You want a Chooser which will always play the `drums.wav` and `bass.wav` samples. The Chooser needs to also play either `guitar.wav` or `marimba.wav`; `marimba.wav` needs to be twice as likely to be picked as `guitar.wav`.

Scenario 3 transcript

F: Let's look at the next video.

Video voiceover: *Let's say you have a lane which needs to always be selected; for example, you may need a vocal line to always play. To do this you can give the lane a weight of infinity, which means that the lane will be selected before any non-infinitely weighted lanes. In order to understand infinity weight it can be useful to think of an airplane with a limited number of seats. Lanes with infinity weight are the priority boarding passengers, and they are given the first seats on the plane. If there are still seats left after the priority boarders have taken their seats there is a lottery held among the remaining passengers. Here's an example; imagine we have a plane with four seats to fill and six passengers waiting to board. There are obviously more passengers than seats, so we need a system. Each passenger has a weight; passenger one has a priority boarding pass—infinity weight—and so takes one of the seats. Passenger four also has infinity weight, and takes another seat. This leaves two seats and four passengers. All four passengers had the opportunity to purchase lottery tickets for this eventuality. One passenger, passenger five, does not have a ticket and therefore cannot travel. A lottery is held between the three valid passengers and passenger six is the winner; she takes a seat on the plane. A second lottery is held and passenger two is the winner; this is despite them having only one lottery ticket compared with passenger three's two tickets. The final seat is thereby filled.*

F: Can you describe what's going to happen when this Chooser (Figure C.5) runs?

P1: Drums can go as many times as possible.

P2: Drums will always play; that's what it says to me.

F: Infinity in this situation is the weight of it rather than the length of it; it is not the duration of it but how likely is it to be picked.

P1: So it will always get picked.

P2: Of the two samples are going to play, drums will always play and then bass has more weight over the guitar.

F: That's right. It's going to pick two; it will always pick that one [drums] and then the other one will be either bass or guitar, and bass is more likely to be selected. Let me play that for you and then I'll ask you to make a Chooser. Here is the example we just looked at:

[The Chooser shown in Figure C.5 is played multiple times]

F: Can you make a Chooser with four lanes rather than three, and populate those lanes with drums, bass, guitar, and marimba. All lanes should be set to loop. The drums and the bass should always play, and either the guitar or the marimba should play, with the marimba twice as likely as the guitar.

P1: So a three in the nose cone?

P2: What you're saying there is: pick three three instruments at a time.

P1: Drums and bass have got to both play because they are they're priority boarders, yeah, and then there's one [seat] left once you've seated these two passengers ... so it's going to be one of these two [guitar and marimba].

F: So, if I play that:

[Plays the Chooser written by the participants, shown in Figure C.6]

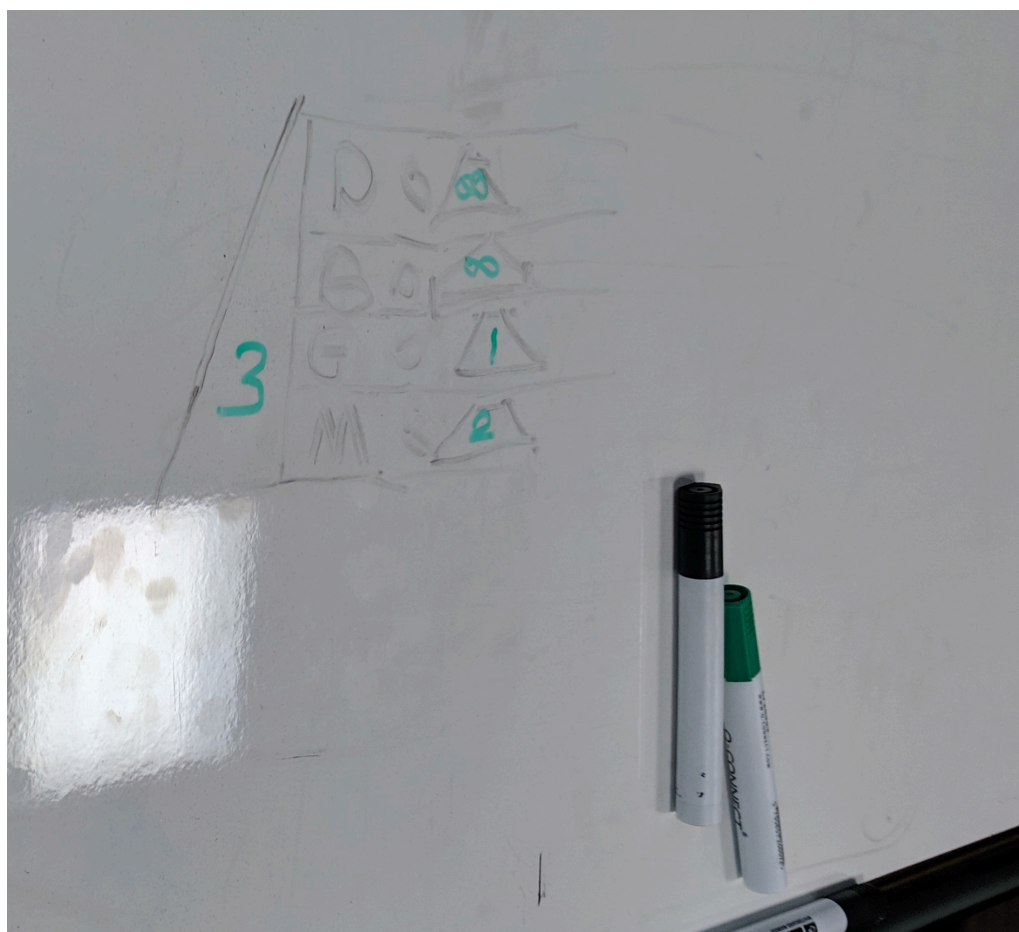


Figure C.6: The result of scenario 3, collaboratively written by both participants.

P1: How come the the number doesn't exclude the bass or the drums? [QUESTION]

F: Well, we could exclude lanes; if you wanted to say 'never play me this', can you see how you would do that?

P1: Put zero in the weight.

P2: That was a couple of videos back.

F: Yes, that's right; if a lane has zero weight it can't be picked.

P2: What's the highest number that can be picked? [QUESTION, NONDET-INTRO]

F: You can choose any number; there is no upper limit. Infinity has a special role, as we have talked about. Very good. Any thoughts or questions or observations at this stage?

P2: Isn't *Ableton* similar? Doesn't it do something like this? [OBSERVATION, EXPECTATION]

F: In what way do you think do you think *Ableton* is similar?

P2: I have only used *Ableton* like once and hated it. From what I remember it's similar it's like that isn't it? It's tiles, you work on tiles. It was just muting and making your own little tile sequence in *Ableton*.

P1: *Ableton* lets you turn things on and off while it's playing.

F: Yes, it's designed for real-time interactivity, as the name suggests.

P2: The difference is that there is a human turning things on and off in *Ableton*, but in this [Choosers] the software is making the selections once the human has told it what rules to follow. [OBSERVATION, EXPECTATION]

F: That's a really good observation.

P2: I did hate it [*Ableton*] though as I don't want to control things while they are playing. [OBSERVATION, NONDET-INTRO]

C.4 Scenario 4: Full Chooser, including hard and soft stops

The participants were shown a video (Bellingham, 2020b) which introduced a Time Chooser to a Soundable Chooser, making a Full Chooser; the nose cone limitations for a Time Chooser; infinity repeats; and hard and soft stops. The participants were then asked to answer the following questions and to complete the following task:

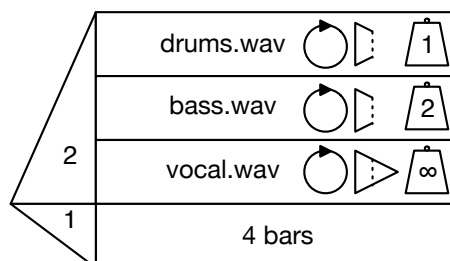


Figure C.7: The Full Chooser used in scenario 4 of the second user study, introducing Time Choosers and both hard and soft stops.

- Questions

- Describe what will happen when the Full Chooser in Figure C.7 is run.
 - What will happen when the Time Chooser's duration has elapsed?
 - The Time Chooser's nose cone is currently set to 1. What else could it be set to? What would happen if it is
- Task
 - Using the Chooser shown in Figure C.7 as a starting point, you need to create the next musical section. The Chooser for the new section needs to do the following;
 - Select and play four out of six samples; the samples are drums.wav, bass.wav, guitar.wav, marimba.wav, bvs.wav, and vox.wav;
 - The vox.wav and marimba.wav samples are essential to the track, so they always have to be selected;
 - Of the other samples, the drums should be more likely to be selected;
 - The Chooser should run for a minimum of 6 bars, with all samples looping;
 - The vocal tracks (vox.wav, bvs.wav) should be allowed to finish playing. The other tracks should stop as soon as 8 bars has elapsed.

Scenario 4 transcript

Let me show you the next video.

Video voiceover: *A time lane is a lane whose purpose it is to specify a musical duration, usually in beats, or bars, or seconds. A time lane can be included in what's known as a Time Chooser, as seen here. Note that the Time Chooser has a nose cone which contrasts visually with the nose cone of a Soundable Chooser. Time Choosers and Soundable Choosers have similarities and differences. a Soundable Chooser can be given a limited musical duration by adding a Time Chooser to its bottom, as seen here. Note how the nose cones fit together. The duration of a Soundable Chooser will now be controlled by the Time Chooser. Together, the two types of chooser are referred to as a Full Chooser, or just a Chooser for short. The purpose of a Time Chooser within a Full Chooser is to moderate in a non-deterministic manner how long the Soundable Chooser and its individual lanes play. When you add a Time Chooser to a Soundable Chooser you can control how each soundable lane will stop when the time chooser's duration has elapsed. Most music software allows us to stop playback immediately; in Choosers this is known as a hard stop. Choosers contain another type of stop, called a soft stop. When a lane is set to a soft stop it allows the currently playing sample to finish before stopping. For example, you might want a melody to complete before stopping. Here we have a Full Chooser consisting of a Soundable Chooser containing three samples, and a Time Chooser with a duration of eight bars. The status column of the Time Chooser shows either a soft stop or a hard stop. In this example, one of three samples will be selected for playback. All samples are set to loop, as shown by the circular arrow. When the duration shown in the Time Chooser—eight bars—has elapsed, the currently playing sample will receive a stop command. If the currently playing lane is set to a hard stop, the sample selected by the Soundable Chooser will be stopped as soon as time's up. If, by contrast, it was set to a soft stop, the sample would not be stopped at this point and instead the loop controlling the sample would be turned off and the sample would play to its end. Consequently, if the sample was shorter than eight bars duration it would loop one or more times before playing to its end after the loop was switched off. If the sample were longer than eight bars the entire sample should play just once.*

P2: [Laughing] Now it's getting complicated!

[The participants are shown the example in Figure C.7]

Here's an example with a Time Chooser at the bottom for the duration, and then a Soundable Chooser at the top which is going to play the sounds.

P2: So, the number one in the four bars nose cone means one cycle of four bars? [QUESTION]

P1: I'm still a bit lost. [PROBLEM]

F: It means 'pick one of one lane'; essentially it can only pick one because there's only one lane, but you'll see in a bit that we could have more lanes down here. At the moment we're saying 'pick me one of those [Time Chooser lanes], and pick me two of those [Soundable Chooser lanes]'.

P2: So it makes sense—so the vocals are infinitely weighted so we know that the vocals will definitely be one of those two and the bass has more weight over the drums with a hard stop.

P1: Ah, I see.

F: The Soundable Chooser will select the only lane that is available. The nose cone will make more sense when there are more lanes, but at the moment it's essentially on or off; it's either going to pick it, or not pick it.

P2: Yes, so you're choosing the four bar lane. So, does everything play to a four-bar duration? Is that what it's saying? [QUESTION]

F: What it's saying is, when four bars has elapsed, then this [the Time Chooser] will tell these [the lanes in the Soundable Chooser] to stop playing. Depending on their stop status they will stop; these two will just stop immediately regardless of what they're doing, and this one will be allowed to finish before it stops.

P1: That bottom has to be infinity, isn't it? [QUESTION]

F: Oh, do you mean in terms of its weight?

P1: Yeah yeah, because there is only one lane. It's got to pick that one because there's no other choice.

[The Chooser in Figure C.7 is run multiple times, always playing vocals, and either drums or bass]

F: The vocal sample is eight bars long.

P2: The bass and drums are probably just one bar loops? [QUESTION]

F: The drums is eight bars and the bass is four bars.

P2: It didn't compute before in my brain. Now it makes sense because now the vocal's eight bars and it will carry on because you're telling it to carry on until the end of the sample. The drums will stop because of the hard stop.

F: So, if I can give you a scenario to draw a new Chooser. We want a Soundable Chooser with six lanes. The samples are drums, bass, guitar, marimba, backing vocals, and vocals. All samples should be set to loop. Set the vocal and marimba lanes so they are always selected. Of the remaining lanes, set the drums so they are more likely to be selected than the other lanes. We want four out of the six lanes to be selected. Give the Chooser a duration of six bars. Finally, the vocals and backing vocals should be allowed to finish once they have started, but everything else should be stopped at the end of the six bars.

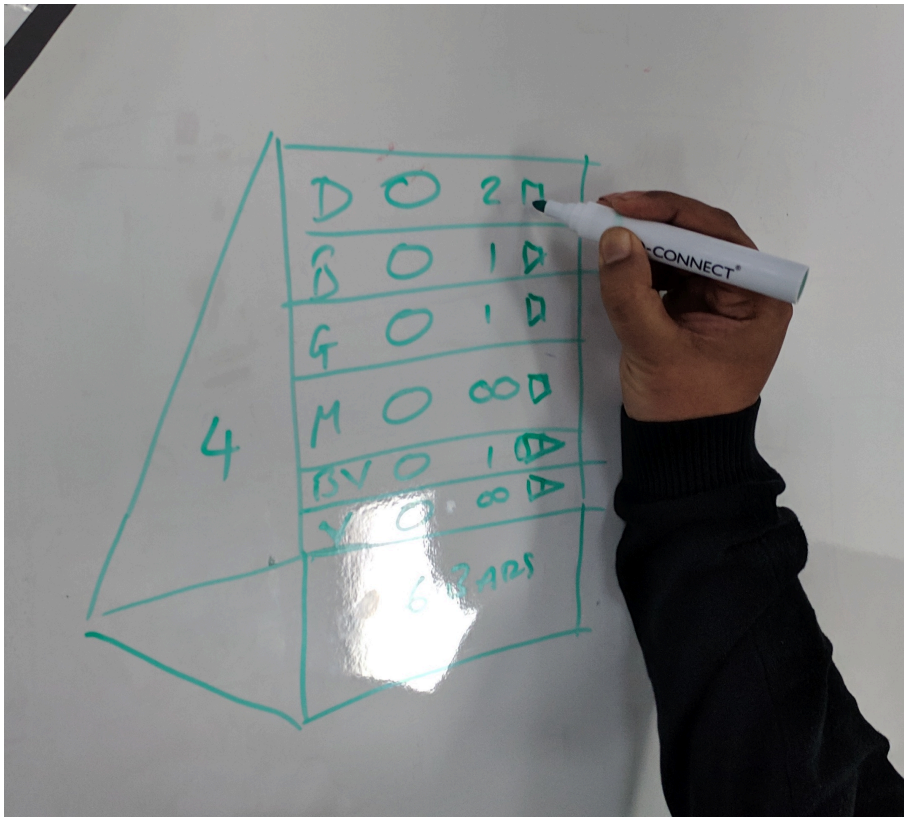


Figure C.8: Scenario 4, with changes being made by P2.

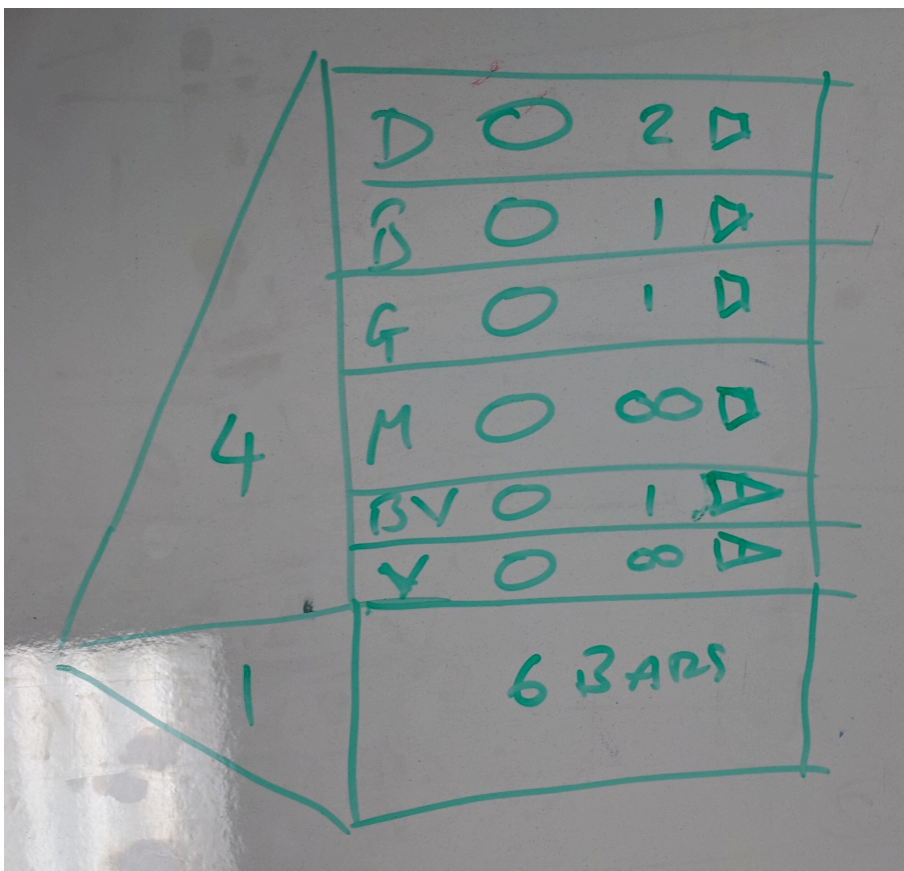


Figure C.9: The finished Chooser for scenario 4.

[Participants successfully complete the task without any errors; they are shown working in Figure C.8, with the finished Chooser shown in Figure C.9]

F: Let's listen to it.

[Playback of the participant-created Chooser shown in Figure C.9]

P1: It played for six bars and then everything but the vocal was stopped.

P2: The other thing is your time signatures; you've got to know your time signatures. [OBSERVATION, EXPECTATION]

F: You're absolutely right to pick that up. This test only looks at some of the implementation. There is a global setting for things like tempo and time signature, because six bars is only meaningful in the context of those other factors. The current design also allows you to open each one of the lanes to give you more information, as well as a rudimentary mixer.

P2: OK, sounds useful.

C.5 Scenario 5: Multi-lane Time Choosers

Participants watched a video (Bellingham, 2020b) which introduced infinity for the Soundable nose cone, and infinity as a duration in a Time Chooser; multiple lanes in a Time Chooser, including weighted choice and nose cone limitations; and soundable content in a Time Chooser, including mute/non-mute.

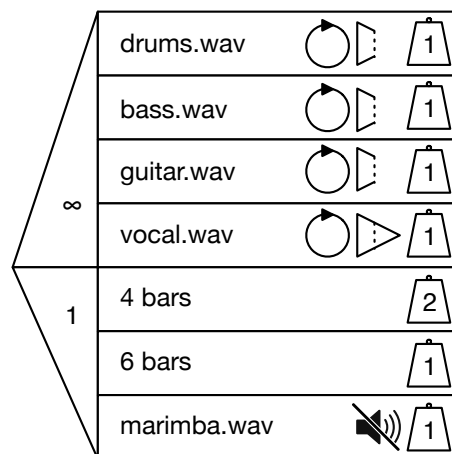


Figure C.10: The example used in scenario 5 of the second user study, showing infinity in the nose cone of the Soundable Chooser, the user of a multi-lane Time Chooser, and soundable content in a Time lane.

- Questions
 - Explain what will happen in the Time Chooser in Figure C.10.
 - What effect will the Time Chooser's selected duration have on the Soundable Chooser lanes?
 - How could we make the marimba.wav sample audible if it is selected?
 - In the Soundable Chooser, which lanes will be selected?
 - What effect will the Soundable nose cone setting have? How could you quickly change the Chooser to play two of the four samples?

- Tasks
 - Sketch out the previous example on the whiteboard, and make the following musical changes:
 - * Make it so 2 lanes are chosen; the vocal and one other instrument;
 - * Make it so the Full Chooser has a duration of either 4 or 8 bars, with hard stops on all lanes.
 - Next, create a second Chooser and sequence it so it plays before this Chooser. The new Chooser should have the following characteristics:
 - * Drums and bass should always play, and should loop until hard-stopped;
 - * Either the vocal or backing vocals should play, looping until soft-stopped. The choice between them should be equal;
 - * The Chooser should have a duration of 4 or 8 bars, with a 4 bar duration three times more likely to be chosen than an 8 bar duration.

Scenario 5 transcript

F: Right, onto the final video.

Video voiceover: *Infinity could be used in the nose cone of a Soundable Chooser as a quick way of playing all available lanes. If any lane has a weight of zero it is not available and will therefore not be played. As in Soundable Choosers, Time Choosers can contain multiple lanes, each with a weight to control the likelihood of selection. One key difference is that the nose cone in a Time Chooser will only allow one selection or zero; this is to prevent us from selecting multiple durations simultaneously. The most common setting for the Time Chooser nose cone is one, meaning that it will select one duration. If the Time Chooser's nose cone is set to zero, any connected Soundable Chooser will run as though there is no Time Chooser. This allows for quick arrangement changes 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 samples will play and the Chooser will be released when they have finished playing, regardless of length. You may want to use the duration of a sample to control the duration of the Chooser; to do this, put the sample into a lane of a Time Chooser. You will see that there are two parameters to the right of the lane; the weight, to control the likelihood of selection, and a mute/play control. The mute/play allows you to control whether you want to hear the sample if the lane is selected, or if you want to use it for duration only without hearing it.*

P2: So, did the duration of the backing vocal sample supersede the duration? Is that what it's saying? [QUESTION]

F: In this particular example, in the Time Chooser, there are three possible durations and it's going to pick one of them; four bars, eight bars, or however long that it [pointing to the backing vocal sample].

P2: Yes, that's what i'm saying. So, if that one's selected, and if that's 12 bars, it's going to play for 12 bars.

F: Exactly. So, the final example image here [opens Figure C.10]. What's going to happen when you press play?

P1: All four are going to play.

P2: They're equally weighted as well.

P1: Yeah, equally weighted, although the weight wouldn't matter here because they're all playing.

P2: If one of them was weighted zero then it won't play.

F: Absolutely right; if it's got zero weight it's a bit like having no ticket to get on the plane; it just can't get on. What will happen with the Time Chooser?

P2: It's going to choose one [lane]. Four bars has double probability of playing compared to the six bars and the length of marimba. It's not chosen to play—as in sound—it's muted.

P1: Yes, that's correct.

F: What about the stop behaviour in the Soundable Chooser?

P1: Just half it, I would say, like the guitar would be two bars, same as the bass and drums.

P2: [To P1] That [hard stop icon] doesn't mean half; the drums, bass, and guitar will stop, and the vocal will play to the end of the sample.

P1: [To P2] Ah, so they have different settings, is that what you're saying?

P2: [To P1] Most of the samples will stop at the end of the phrase whichever—four or six bars or whatever—and if the vocal is eight bars long then it will continue to play until the end of the eighth bar.

P1: So, if it's four bars, the whole song is just four bars. This whole Chooser would just stop; four bars and then stop.

P2: But if the vocal is eight bars it will play until the end.

P1: So what's the guitar then, and the bass? How many bars is that for the samples? [QUESTION]

P2: [To P1] He said before, the bass sample is four bars, the guitar sample is eight bars long, but they'll never get to eight bars [because they] will be stopped before then.

F: How do you feel about these icons? It is interesting that you read the hard stop icon as halving.

P2: The icons—especially that one [soft stop], play until the end, and stop dead [hard stop], that one's not really interactive. The rest are pretty cool. [PROBLEM, EXPECTATION]

P1: It's because usually nowadays samples usually just have the seconds, how many seconds they last, you know? [SUGGESTION, EXPECTATION]

P2: Yeah. If that's your user interface, knowing the sample length—let's say seven bars—it will really make life easy ... I've been using applications for 15 years, and to me that's hard work at the moment. I have to compute it in my head. [SUGGESTION, EXPECTATION, IMPROVEMENT]

F: The design here is that this is at least partially for people who are not particularly experienced in using music software.

P2: They will find that [Choosers] hard. [OBSERVATION]

P1: I'll probably find it easier, way easier than *Pro Tools*, because it just looks just bigger and easier to read. [OBSERVATION, METAPHOR]

F: Do you think it would matter if the user already knew DAW software like *Logic* or *Pro Tools*?

P2: I use *Logic* and I can see that this isn't anything like *Logic*. It's confusing that it doesn't use the same layout. [PROBLEM, EXPECTATION]

P1: That is probably why I like it. I don't feel like I need to know as much. Depends on the age of the person too. [OBSERVATION, EXPECTATION]

F: Would you like to have an 'expert switch' to show more information?

P2: That could work if it showed the length of the samples here [in the lane]. Time signature information, tempo information, would be helpful. But that's [for] advanced users, yeah? If you're talking [about] somebody who's picked up on a basic level, it makes sense [other than] the icons. [SUGGESTION, EXPECTATION, IMPROVEMENT]

P1: This seems like something you would see as kind of an app, rather than like a music DAW; like a phone app. [OBSERVATION, EXPECTATION]

F: Regarding the icons, we started off with hard stop being like a traffic stop sign [drawing it]. Can you think of another piece of music software that issues a command that says 'stop when you've finished'? What icon could be used? What could visually represent 'stop when you've finished'?

P2: If you talk about old samplers, like the EXS24, you've got this interactive on and off point where you can click and stuff, like a one shot. I don't know whether you should call it one shot. Maybe one shot's a bit too advanced for the basic user, but something similar. Maybe a toggle switch? Play to end. [SUGGESTION, METAPHOR, EXPECTATION, IMPROVEMENT]

F: Do you think that would need text?

P2: Maybe you could do like what you did there [points to soft stop] with the picture. [SUGGESTION, IMPROVEMENT]

F: Do you think once you've used it for 10 minutes you just learn what it means? Do you think you just pick it up, and is that good enough?

P2: I picked it up. [OBSERVATION]

P1: I think you just have to usually, just find out yourself. It looks easy; you've gone step by step and it looks easy, to be honest. If I was to look at this when I first came in I wouldn't understand it. [OBSERVATION, EXPECTATION]

F: I will ask to you do one more of these [Choosers]. What I'd like you to do is to make another Chooser next to the existing one and then we'll sequence the two together. So, if you can make a Soundable Chooser with four lanes and a Time Chooser with two lanes. In the Soundable Chooser add drums, bass, backing vocals and vocals. All four of them will loop. We want drums and bass to always play. We want either of the vocal samples to play with equal probability.

P1: Ah, so the nose cone should be three.

P2: The weights [on the two vocal lanes] could be two each, or five each, or ten each—as long as they are equal. [OBSERVATION]

F: In the Time Chooser, make it select either four bars or eight bars duration, with four bars three times more likely than eight bars.

P1: Three times more likely...

P2: Three and one.

P1: Of course.

F: Drums and bass should both hard stop, and the vocals should both soft stop.

[The participants successfully created the Chooser shown in Figure C.11. Note the use of simplified icons: a square for a hard stop, and a triangle for a soft stop]

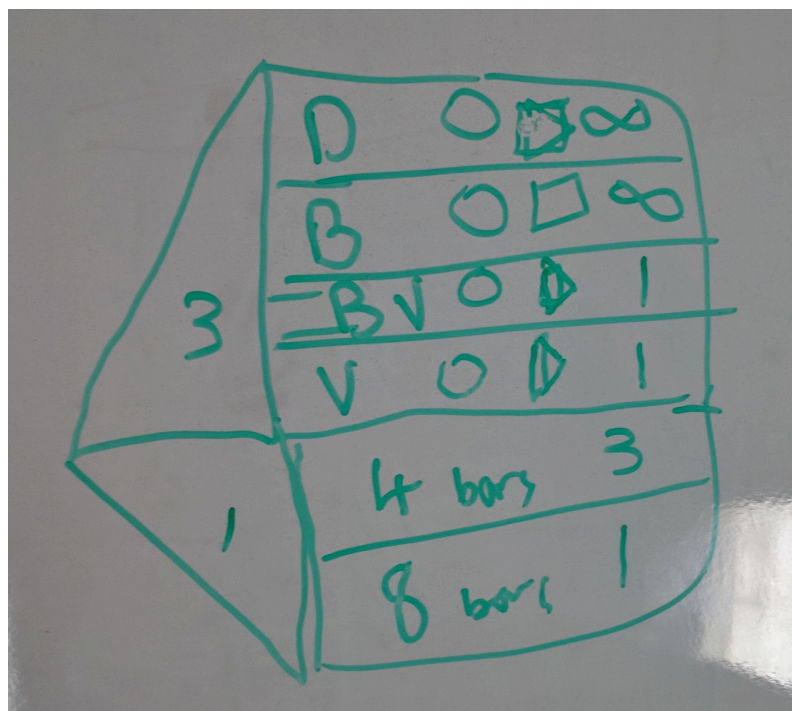


Figure C.11: The result of scenario 5.

C.6 Scenario 6: Playground

Participants were given the following task:

- Using the whiteboard and the available samples, 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.

Scenario 6 transcript

F: Finally we want to create a sequence of three Choosers; you get a free hand here, and you can use any of the samples you've used so far. Just imagine this Chooser [the example shown in Figure C.10, which was still on the computer screen] was on the same screen as that one [Figure C.11, still on the whiteboard]; how would you make that one play first, and then this one play second?

P1: Stick an arrow out of the end, to the right here, and then it would go into that [Chooser].

[The participants created a sequence of three Choosers]

F: OK. Let's play your sequence. [playback]

P2: [Correctly identifying an error] That played the marimba, didn't it? [PROBLEM, OBSERVATION]

F: Yes, you're right, I typed in the wrong duration. Let me change that to six bars.

[Facilitator corrects the code and plays the sequence three times]

F: That was my fault; that wasn't an issue with anything that you did. You can see that every time you run the code there can be the possibility for different outputs, and it's up to you as the composer or as the producer how much nondeterminacy you want to include in your work.

P2: I think it's really good for random arrangements. When you sitting down, you're muting parts, and you're just kind of working out an arrangement, moving things left and right. I think this could be good for arrangement, but you've got to be good at spotting what's going on. [OBSERVATION, NONDET-INTRO]

P2: You still haven't got—because it's playing at random, either one of these or either one of these—it doesn't play the same thing every time does it? [OBSERVATION, NONDET-INTRO, EXPECTATION]

F: You could make it do that if you wanted it to.

P2: But what I'm trying to say is ... if the computer picks an arrangement for you [and] you might like it—wow, that was good—

P1: Like an inspiration machine.

P2: Yeah, like that sequence of the drums and bass playing for eight bars with just the backing vocal, and it's stopping, and the hard stops and whatnot. It could be 'oh yeah, that's that's a pretty good sequence', but as a user you have to be good at spotting it otherwise it would go so fast it would go past you. [PROBLEM, OBSERVATION, NONDET-INTRO, IMPROVEMENT]

P1: Could you select the one that you like, take the information out, and put it into another tool? [QUESTION, NONDET-INTRO, IMPROVEMENT]

P2: Yeah, 'I like sequence f', you know? But, I think the whole concept of recording is to get the best out there in the first place. I don't want the recording to change once I've made a decision. [OBSERVATION, NONDET-INTRO]

C.7 Final questions

Participants were asked the following questions at the end of the second user study:

- Can you see anything this would be useful for?
- Can you see any ways in which this is similar to other tools you have used?
- Is there anything that is made easier by this system? Anything which was not possible made possible/hard and made easier? You've already mentioned Ableton; is there any other software that is similar to this that you've used?

Final questions transcript

P2: *GarageBand* is pretty close, I think. I mean, I don't use it but I see my son using it, he's just picking things at random. He's controlling it, not the software. [OBSERVATION, METAPHOR]

P1: The algorithm is the computer picking it for you. [OBSERVATION, NONDET-INTRO]

P2: With this [Choosers] you're giving it the scenario—'what if', and trying to set the parameters so there isn't one that sounds terrible. You're still recording in the traditional way, so you make your audio segments, whatever length they are, and you just turn to the computer to say 'these are the parameters, play me whatever you can'. [OBSERVATION, NONDET-INTRO]

P2: The decisions would go back to the recording studio when they are recording the music. Maybe the vocalist doesn't want to do multiple takes; 'why would I want to do lots when I can do one good take and say I'm satisfied, I'm done'. [OBSERVATION, NONDET-INTRO, EXPECTATION]

P1: Or, do you do four takes [of] the same [melody] or do you actually change the melody so that ...

P2: Oh yeah!

P1: ... someone listening to it would actually hear a big change. [OBSERVATION, NONDET-INTRO, EXPECTATION]

P2: That's a good point, that's a better argument. Different takes are different melodies, or a different spin on the idea. If you've got a different drum roll, or a different guitar [part] here, that's a user can pick up on [and think] 'well, I like that one'. [OBSERVATION, NONDET-INTRO, EXPECTATION]

F: Thank you for being so generous with your time today.

Appendix D

Design exercises

D.1 Design exercises used to inform Choosers v1

D.1.1 Lessons from an informal design exercise using Direct Combination

The following design exercise, referenced in Section 5.2.1 in Chapter 5, uses Direct Combination in the creation of a series of lo-fi prototypes, first on paper (Figure D.1), then sketched in diagramming software (Figures D.2, D.3, D.4), and later in *SuperCollider* (Figure D.5). DC was a good match for several of the aims of the project and, via testing, evaluation, and discussion with pilot users, it was found that the following specific musical and interface design characteristics were shown to be particularly desirable.

Constraint Reducing the search space and showing only contextually-relevant content is a core principle of DC. On the basis of a series of informal scenario tests I found that constraining the search space may reduce the burden on the user as they do not need to learn a large number of operators. The early paper sketch in Figure D.1 shows a branching musical arrangement on the left, audio files on the upper right, and a list of possible actions on the lower right. The Intro section is currently selected, resulting in the constraint of the Action menu to only those actions relevant to this selection. With the Intro section selected, as shown, the audio files assigned to that selection are highlighted. One file will be selected per colour, meaning that three files will be selected when played—one each from the blue (one drums file), green (three bass files), and red (two guitar files) highlighted files. The duration of each file is shown in bars (L:16 means ‘16 bars’) and the relative weighting of each choice shown as a percentage.

Parsimony The design exercises explored the effects of constraining vs. increasing the number of interface elements. Early tests focussed on the combinatorial use of a small number of widgets, with later designs introducing a number of user-controllable widgets. Figure D.4 shows a user-controlled layout in which the user introduces as many widgets as required. In this way the user can create multiple widgets of the same type, such as multiple number widgets, to further constrain selection. The widgets shown in Figure D.4 were referred to as depots; in this context, unlike a menu or a button with a fixed function, a depot was considered to be a widget which could store one or more user-controllable values or objects. Note the use of multiple number depots in Figure D.4 (and later in the partial implementation in *SuperCollider* shown in Figure D.5); this allowed us to test the combinatorial options available. For example, if the user were to select two number depots, one set to 2 and the other to 4, the suggested actions would constrain to reflect these values—for example, ‘repeat between 2 and 4 times’. While a greater number of widgets increased expressivity, by running informal scenarios through lo-fi paper prototypes I found that the increased complexity reduced usability.

Semantic consistency The columnar DC design shown in Figure D.1 made use of a consistent interaction model; users select an object to show both the possible options and current con-

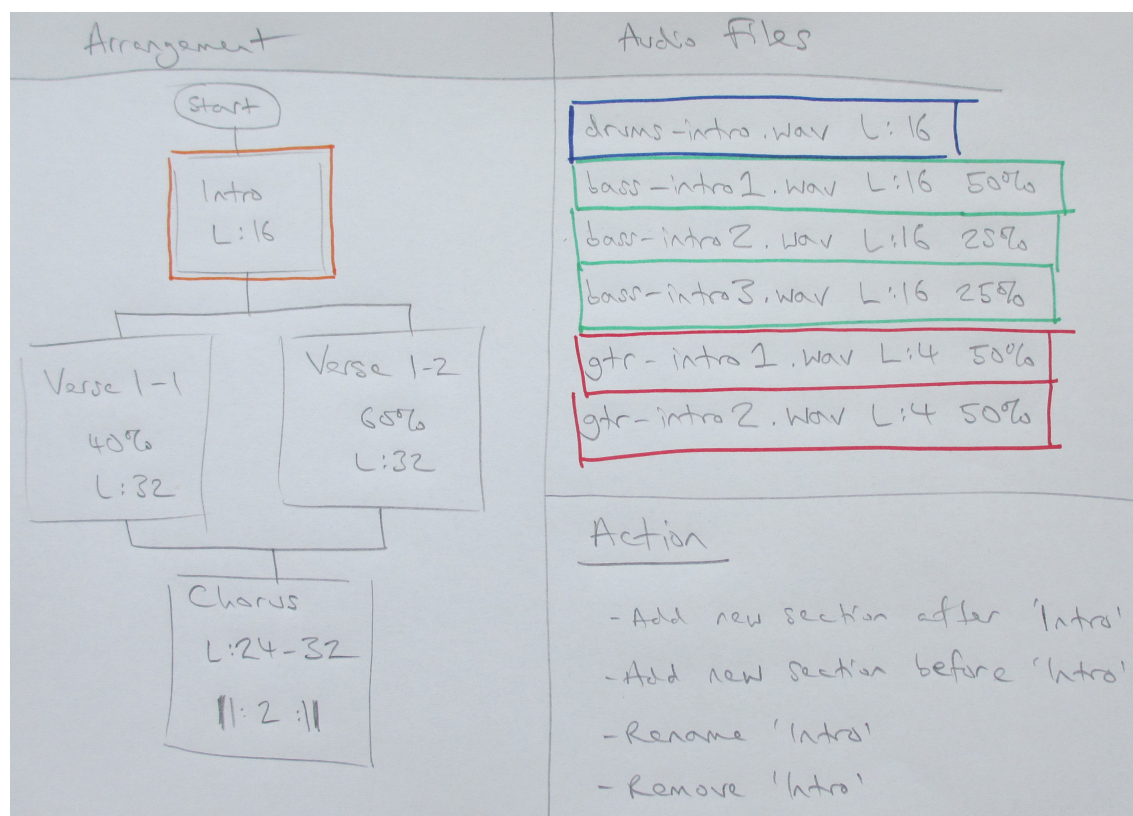


Figure D.1: A paper sketch showing a branching musical arrangement on the left, audio files on the upper right, and a list of possible actions on the lower right. The Intro section is currently selected, and the audio files that are assigned to that selection are highlighted. One file will be selected per colour, with duration shown in bars (L:16 means '16 bars') and weighting of each choice shown as a percentage. The Action section in the lower right shows the current available actions given the selection of the Intro section.

Arrangement	Audio Files	Number
Start	Drums1.wav L:16	8
Drifting L:64 to L:256	Drums2.wav L:32	256
	Drums3.wav L:8	4 8
	Bass1.wav L:8	16 32
	Bass2.wav L:8	1-10
	Bass3.wav L:32	11-20
	Pad1.wav L:16	21-30
	Pad2.wav L:32	31-40
	Pad3.wav L:8	41-50
		50+
Action Add 8 new sequential sections after 'Drifting' Add 8 new branching sections after 'Drifting' Add 8 new sequential sections before 'Drifting' Add 8 new branching sections before 'Drifting' Make 'Drifting' 8 bars long Make 'Drifting' a multiple of 8 bars Set tempo for 'Drifting' to 8 BPM Set volume for 'Drifting' to 8% Repeat 'Drifting' 8 times before continuing		

Figure D.2: In this sketch, the section named 'Drifting' has a duration of between 64 and 256 bars. It has been selected alongside the number 8 (upper right), meaning that the Action menu now shows all the actions which combine the section and the number.

Arrangement	Audio Files	Number
Start	Drums1.wav L:16	8
Drifting L:64 to L:256 8	Drums2.wav L:32	256
Moving L:64 to L:256 8	Drums3.wav L:8	4 8
	Bass1.wav L:8	16 32
	Bass2.wav L:8	1-10
	Bass3.wav L:32	11-20
	Pad1.wav L:16	21-30
	Pad2.wav L:32	31-40
	Pad3.wav L:8	41-50
	Silence L:1-8	50+
		∞
Action Add new section after 'Drifting' Add new section before 'Drifting' Rename 'Drifting' Remove 'Drifting'		

Figure D.3: This example shows the audio files assigned to the musical section named 'Drifting'. As with Figure D.1, colours are used to group files together; in this instance, one of Drums1 .wav, Drums2 .wav, or Silence will be selected.

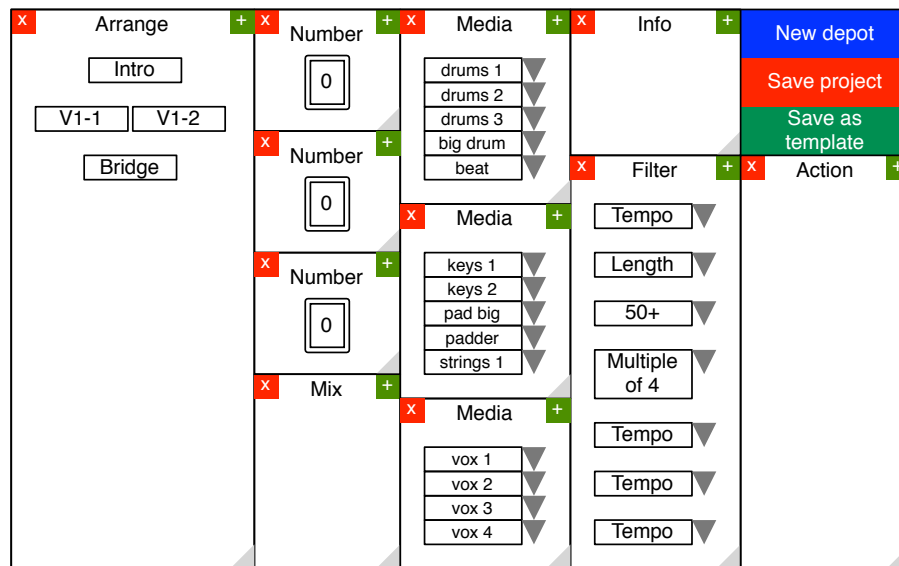


Figure D.4: This example tested a user-controlled layout; the user introduces as many widgets (called depots) as required. In this way the user can create multiple widgets of the same type, such as multiple number widgets, to further constrain selection. Note the down arrows to the right of menus, which are a filtering system.

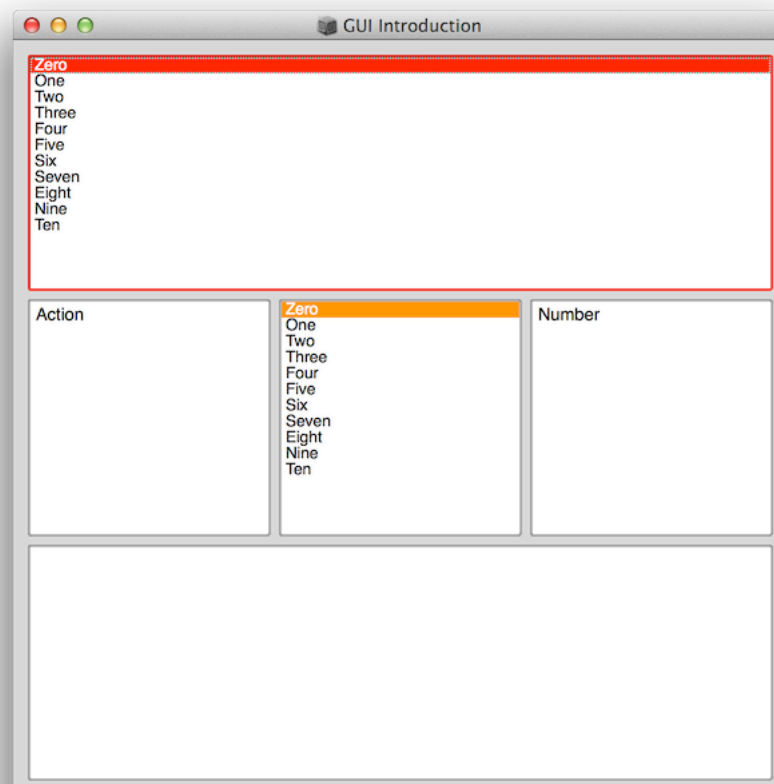


Figure D.5: This is an implementation of the DC design in *SuperCollider* (McCartney, 2002).

nections between elements. For example, Figure D.3 shows the selection of a musical section, ‘Drifting’. As a result of this selection, the interface highlights the audio files that will play as part of the section; in this case, five audio files and a dedicated ‘Silence’ option. As in Figure D.1, colours are used to group files together; for example, there are three files coloured red (Drums1.wav, Drums2.wav, and Silence), and one of these red files will be selected for playback. Note that there is a choice between two bass files, and as Pad2.wav is the only audio file coloured blue it will always be selected for playback. The surfacing of interdependencies was shown to be important for music making, leading to various designs to further consider how they could be more clearly visualised.

Combinatorial power The DC designs helped to reduce the number of core interface elements and to allow for more complex functionality via combinatorial usage.

Compositional flexibility Running through simple scenarios with lo-fi paper prototypes demonstrated that DC allowed users to input commands in a number of ways; it did not impose a strict order to either the input or editing of material. The sketch in Figure D.3 shows a design with the musical structure in the left column; one musical section is currently selected and the highlighted audio files show the contents of the section. This design allows the user to work in one of two ways: the user can create the macro structure before populating the sections (top-down), and it allows the user to create musical fragments/sections before assembling them into a structure (bottom-up).

Issues with constraining possible choices As paper (and implemented) prototype DC systems were developed and evaluated by inspection, it was found that certain actions were complicated by a proliferation of options. For example, Figure D.2 shows an example in which a musical section and a number have been selected. This selection constrains the ‘Actions’ menu to show all options which involve both the musical section and the number—for example, adding that number of sequential or branching sections before or after the currently selected section, changing the tempo, changing the volume, repeating the section that many times, and so on. This is an example of the number of options in certain circumstances becoming overlong and difficult to manage (Miller, 1956). A similar issue was encountered when the system was populated by a large number of audio samples. Both issues required further constraint of options, leading us to consider filtering or tagging in order to reduce the number of choices shown to the user. An example of one of the tests, allowing for greater search space constraint via user-controllable depots, as shown in Figure D.4. In this example the user is able to add an undefined number of depots of various types. The example shown in Figure D.4 allows the user to add depots via the menu in the upper right, with each freely positionable in the play area. Note that this example contains multiple instances of the same depot type, allowing for new behaviour. For example, multiple number depots allows the user to select multiple numbers which can then be used to constrain the possible actions—e.g. the selection of 2 and 4 could allow for ‘repeat either 2 or 4 times’, ‘repeat between 2 and 4 times’, and so on. However, I concluded that the design and management of such a system would place an unnecessary burden on the user.

Issues with a lack of visualisation DC is a useful design principle with a number of desirable characteristics which intersect with the goals of the project. However, the lack of structural visualisation proved problematic when considering musical arrangements.

D.2 Design exercises used to inform Choosers v2

D.2.1 Icon design

The following design exercises were undertaken as part of the development of Choosers v2; they are referenced in Section 8.1.3 in Chapter 8.

D.2.1.1 Weight icon

The weight control is a metaphor. Taking inspiration from a literal steel weight (Figure D.6), various test icons were created, such as those shown in Figure D.7 and Figure D.8. The final version of the weight icon, used from Figure D.16 onwards, adds a hoop at the top of the trapezium shape to enhance the visual similarity with a steel weight.

Various versions of all icons were considered. The most abstracted draft icons are shown in Figure D.8, where the weight is a simple square, the loop is a circle, and the stop is a triangle (the loop and stop icons will be explored in the upcoming two sections). While an experienced user would be able to link these shapes with their more expressive inspiration they are arguably too abstract to communicate their function to newer users. The circular loop icon (Section D.2.1.2) was therefore initially selected, together with a new stop icon (Section D.2.1.3).

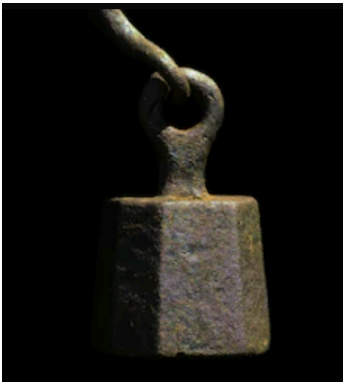


Figure D.6: A steel weight to be used as a metaphor for lane weight.

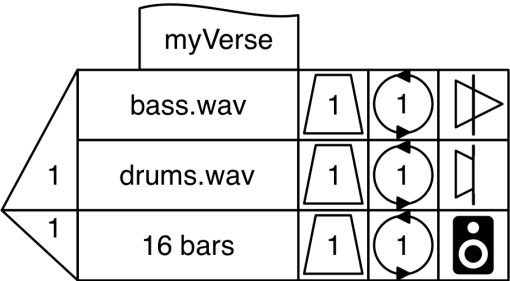


Figure D.7: An early icon test for Choosers v2, including a trapezium to represent the steel weight shown in Figure D.6. This is not necessarily a helpful image as it suggests physical weight rather than likelihood. A circular loop icon is used for repeats (see Section D.2.1.2). Hard and soft stops are shown using a play triangle either running through a line (soft stop will continue to play) or being bisected by the line (hard stop will stop immediately)—see Section D.2.1.3.

D.2.1.2 Loop icon

In order to understand the expectations of users who may not be familiar with typical DAW design, a review of icons used for looping in simple music playback applications was undertaken (see Figure D.10). This, in turn, led to various informal pilot tests of loop and repeat icons able to specify given finite numbers of loops or repeats. The ability to specify a number of loops per-lane was identified as desirable in the first user study (see Section 7.5.2); Choosers v1 allows only for on/off control over looping. The introduction of an integer in the loop icon could be used to specify the number of loops, and the introduction of infinity in Choosers v2 (see Section 8.1.1) now allows for ∞ inside the loop icon as an alternative notation for ‘loop forever’. The icon review also considered

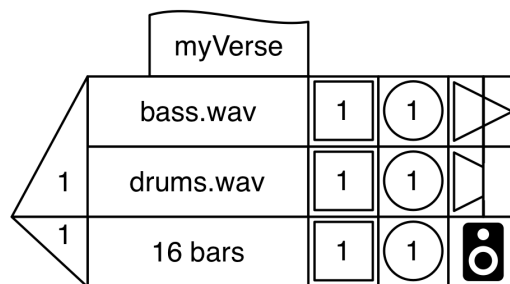


Figure D.8: A variant of the previous design (see Figure D.7) with more abstracted icons. This uses basic geometric shapes, resulting in relative simplicity while retaining the power of the metaphors. The icons are large for reasons of legibility; tests with smaller abstract icons are shown in Figures D.22, D.23.

the trade-offs of a *Spotify*-style circular loop icon (Figure D.9) compared with the type of multiplication notation that would be found on a lead sheet (Figure D.11). It should be noted that ‘loop twice’ and ‘play and repeat once’ are two ways of saying the same thing. The Boolean loop on/off design from Choosers v1 was adopted in Choosers v2 as the default behaviour; the above integer mechanism was considered as an advanced option, and was not implemented for the second round of user studies. In this advanced mode, the equivalent of ‘loop off’ can be achieved via a ‘1’ in the loop icon (‘play once’), and the equivalent of ‘loop on’ is achieved via ∞ in the loop icon.

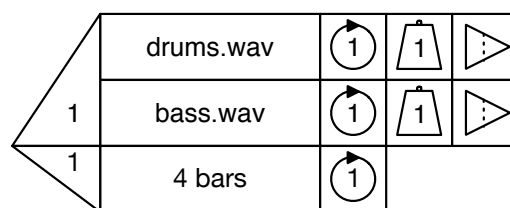


Figure D.9: An example showing a circular ‘repeat’ icon in the Soundable Chooser. Note that this example tested the use of a ‘repeat’ icon in the time lane, providing consistency while allowing for the duration to be easily changed. This design was rejected as it did not offer additional clarity or functionality when compared with simply changing the duration in the Time Chooser lane.

A related candidate refinement to the notation called ‘atomic repeats’ was considered—see Figure D.12. This candidate refinement involves introducing a distinction between repeats associated with the contents of a box and repeats associated with a lane. Whereas a soft stop only allows a single currently running lane repetition to finish, by contrast atomic repeats are considered to be indivisibly bundled into each play of the contents of the box, and are consequently all allowed to finish by a soft stop. To put it another way, an atomic repeat results in different soft stop behaviour when compared with the use of the loop or repeat function. Consider a lane which is set to play twice via a ‘x2’ command in the loop column; if this lane is soft-stopped during the first iteration it will complete playback but will not perform the second iteration. By marking a multiplication symbol next to the sample a new atomic unit with a longer duration is created, changing the soft stop behaviour; both iterations will play until complete.

Designs were considered for nondeterministic repeats, such as the Chooser shown in Figure D.13. This example allows the user to specify the minimum and a maximum number of repeats in a split column. This design was not implemented as similar functionality can be achieved via variables without adding complexity to the core design.

D.2.1.3 Stop icons

The first user study showed that several participants found the concept of a soft stop difficult to understand (see Section 7.5.1). Additionally, some users felt that the icon for a soft stop (a $>$, com-

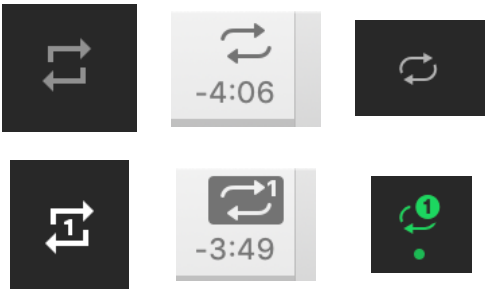


Figure D.10: Icons for looping behaviour in popular music playback applications. The upper row shows a disengaged loop (YouTube Music, iTunes, Spotify). The lower row shows the icon in the same applications denoting a single song loop. Note the use of a number, as in the prototype design for repeats. The tension here is between *repeats* as a musician would understand them (x1 means ‘play once’) and a *loop* (‘loop once’ means ‘play twice’). The number ‘1’ in the icons on the lower row refers not to the number of repeats or loops, but instead shows that a single song will loop infinitely. Framing, and the expectations it brings, would suggest that a different icon should be used, as the functionality is different.

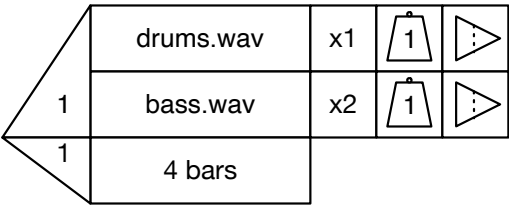


Figure D.11: A possible notation change to multiplication rather than the circular icon; this sketch shows a Chooser in which the drums.wav audio file will play once and the bass.wav file will play twice. The intention is to avoid the danger of a ‘1’ inside a loop symbol being misinterpreted as meaning ‘play once, and then loop once’. Instead, it was hoped that ‘x1’ might be more clearly understood unambiguously to mean ‘play one time’ and less likely to be misunderstood as meaning ‘play one time, then loop 1 time’. The ‘repeat’ column is not used in the Time Chooser as the duration can be changed by using multiplication notation directly in the time lane, as shown in Figure D.12.

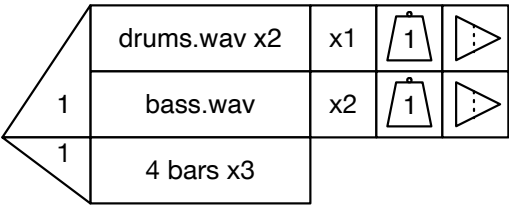


Figure D.12: This figure shows two examples of ‘atomic repeats’, which uses multiplication notation within sample or duration boxes. The drums.wav sample is being repeated twice. This ‘glues’ two copies of the sample together to create a new atomic unit. This atomic unit is then played once (the $\times 1$ in the repeat column). The Time Chooser lane also includes an ‘atomic repeat’ which results in a new duration of 12 bars (4 bars \times 3). This is equivalent to changing the duration to 12 bars and is offered as an alternative notation. Note the difference between the two soundable lanes—the drums.wav sample will play twice regardless of when it is stopped by the time lane, whereas the bass.wav sample may only play once if it is longer than 12 bars, or play twice if it is shorter than 12 bars. If the duration was to be changed using multiplication notation it would be done directly in the time lane, as this would also create a new atomic unit.

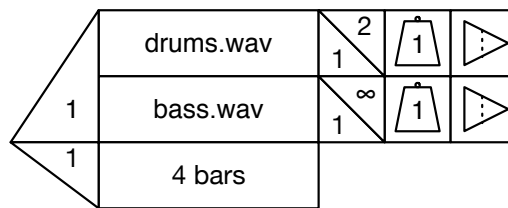


Figure D.13: Following on from Figure D.12, this example explores a possible minimum/maximum repeat column. Here the `drums.wav` sample plays for a minimum of 1 and a maximum of 2 times. The `bass.wav` sample plays at least once, and has an infinite maximum number of repeats. This design provides bounds but does not change the atomic unit. The suggested design in Figure D.12 appears to offer more musically meaningful options.

pared to the \times used for a hard stop) did not adequately communicate functionality (Section 7.5.4). As a result, the icons for both hard and soft stops were redesigned to be both consistent and descriptive. The draft icons, such as those shown in Figures D.7, D.8, use a rightward-facing triangle to represent playback (as on a tape machine's transport controls). A vertical line represents the 'stop' command, triggered by the Time Chooser. The hard stop icon represents an immediate stop by cutting off the triangle at the metaphorical stop line, and the soft stop icon shows that playback continues after the stop line.

The refreshed icons, to be adopted in the design of Choosers v2 and used in the second user study, are shown in Figure D.14.

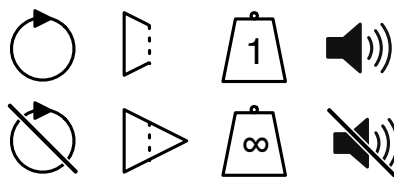


Figure D.14: The refreshed icons used in the second user study: loop and non-loop; hard and soft stop; weight of 1 and infinite weight; soundable file in a Time Chooser play and mute.

D.2.2 Testing designs via programming walkthroughs

Presented here are a number of walkthroughs, undertaken alone and without groups of analysts, to understand the impact of the proposed design changes. At this point in development Choosers still used internal columns, and some sketches experimented with different lane lengths in order to facilitate progressive disclosure and to reflect the disparity in the number of icons required by Soundable vs. Time Choosers, as discussed in Section 8.1.3.

A very simple scenario for a programming walkthrough focuses on using two samples to create a rudimentary nondeterministic musical fragment. 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. The user starts by dragging a single audio file into the play area; Figure D.15 shows that the audio file is now shown in a box. Note that this example includes a loop icon, included to indicate both that the sample will play once and to indicate to the user how to change the number of repeats. If the user were to snap a second lane onto the bottom of the first, the system would create the Soundable Chooser shown in Figure D.16. Without explicit user intervention, the system has added a nose cone on the left, and a weight column on the far right.

Various designs for each element were considered. As an example, Figure D.17 shows a Time Chooser without a nose cone and Figure D.18 shows the same Time Chooser with a nose cone. These versions have different strengths and weaknesses. The version without a nose cone is consistent with the single lane example shown Figure D.15, while the version with a nose cone allows for

disabling the Time Chooser by setting the nose cone to zero. The nose cone also has the virtue that the sloped shape clearly differentiates Soundable and Time Choosers.

If the user were to snap Figure D.18 to the bottom of Figure D.16 the system would update to show the Full Chooser seen in Figure D.19; note the introduction of stop icons on the far right of the Soundable Chooser lanes. The sketches at this stage in development explored the importance of visual consistency, leading to the ‘ragged right’ edge produced by the variable lane widths. As there is only one Time Chooser lane a lane weight is not shown until a second lane is added, as shown in Figure D.20. Finally, a mute control is added automatically by the simulated system if soundable content is added to a Time Chooser lane, as shown in Figure D.21. Note that the loop and weight icons are vertically aligned for legibility; this sketch also experiments with vertically aligning stop icons (only available for Soundable Chooser lanes) and the mute icon (only available for Time Chooser lanes with soundable content).

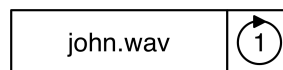


Figure D.15: Following a simple scenario as part of an inspection by programming walkthrough, the user drags a single sample into the play area—this creates a box and adds a repeat control.

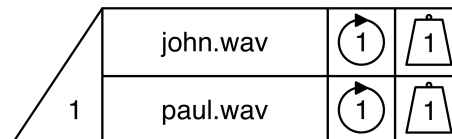


Figure D.16: The user snaps on a second lane to the bottom of the first sample box. This creates a Soundable Chooser, with a nose cone on the left, and a weight column on the far right.

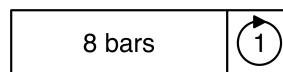


Figure D.17: Time Chooser test to explore the trade-offs in sharing a consistent design between Soundable and Time Choosers; this is a Time Chooser version of the Soundable Chooser in Figure D.15. Note the lack of a nose cone.

D.2.3 Design sketches for soundable content in Time Choosers

The sketches shown in Figure D.22 and Figure D.23 explore various icons for the management of soundable content in Time Chooser lanes; they are referenced in Section 8.1.4 in Chapter 8.

D.2.4 Control over lane contents

As part of the design process, controls for a variety of other functions such as simple mixer functionality (e.g. volume and pan controls) and file management were considered. The draft design of Choosers v2 was used to create various short pieces of music, including an implementation of Terry Riley’s *In C*. *In C* allows for the combination of a number of short musical fragments of differing lengths (Potter, 2002), and I was particularly struck by the level of musical complexity created by such a simple change. In order to make this type of music easily accessible I considered a control panel which could, for example, be opened by right-clicking on a Soundable Chooser lane. A sketch of one such control panel is shown in Figure D.24. In this design the user is able to view the current file name at the top of the control panel, and can open a file picker by clicking on the current file name (in this case, *bass.wav*). Simple horizontal faders are used for volume and pan control.



Figure D.18: The same design as in Figure D.18 but with a nose cone which clearly differentiates this Time Chooser from a Soundable Chooser. It also allows for the Chooser to be quickly skipped by setting the nose cone to zero.

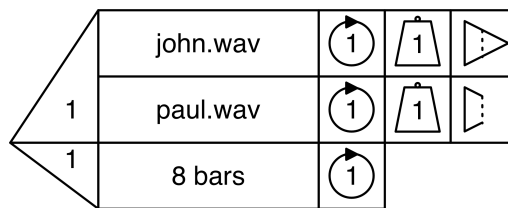


Figure D.19: A Full Chooser. Note the new icons, added stop controls, and the lack of weight and mute for the time chooser.

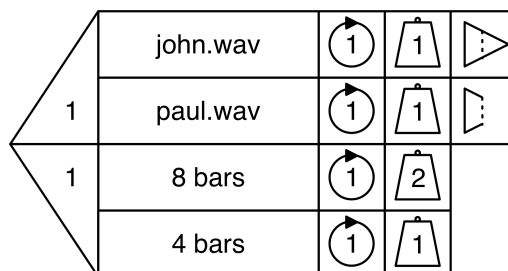


Figure D.20: A Full Chooser with nondeterministic duration, necessitating weights in the Time Chooser.

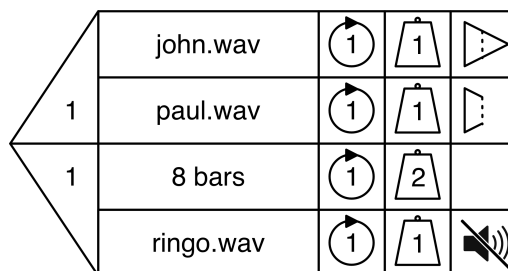


Figure D.21: A Full Chooser with a sample in one of the Time Chooser lanes, requiring the mute control. Note the lack of symmetry caused by the lack of a column in the uppermost time lane.

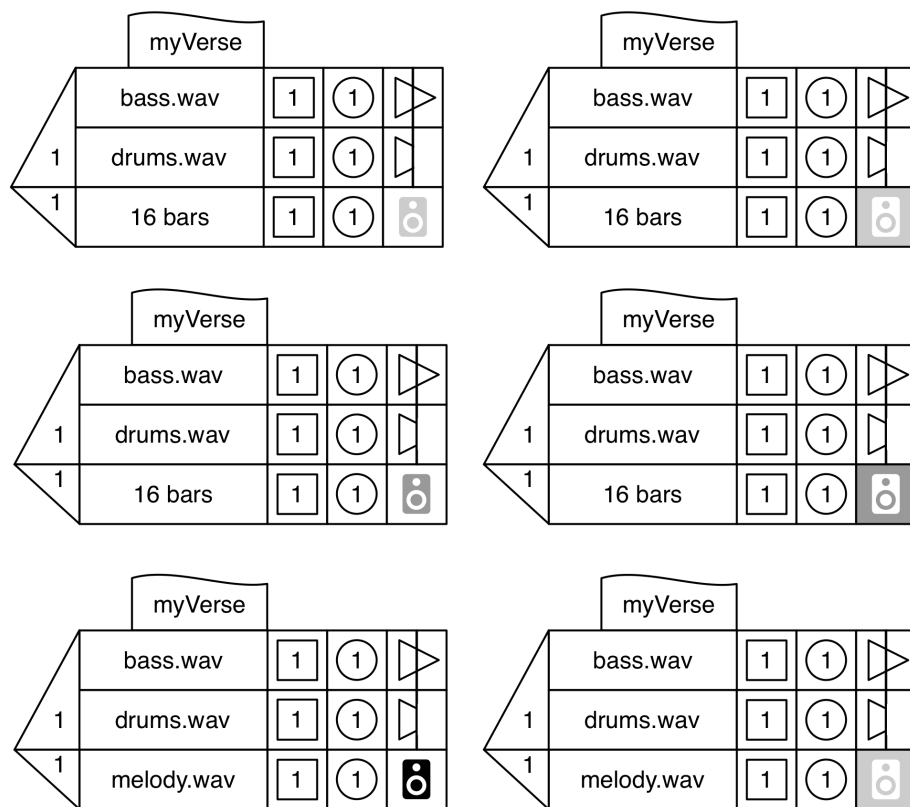


Figure D.22: A series of candidate icons for the management of soundable content in Time Chooser lanes. This series of images shows examples of greyed-out speakers to denote no audible playback of the contents of the time lane. Durations do not contain any audible material. The `melody.wav` example in the bottom row can be made audible or not audible.

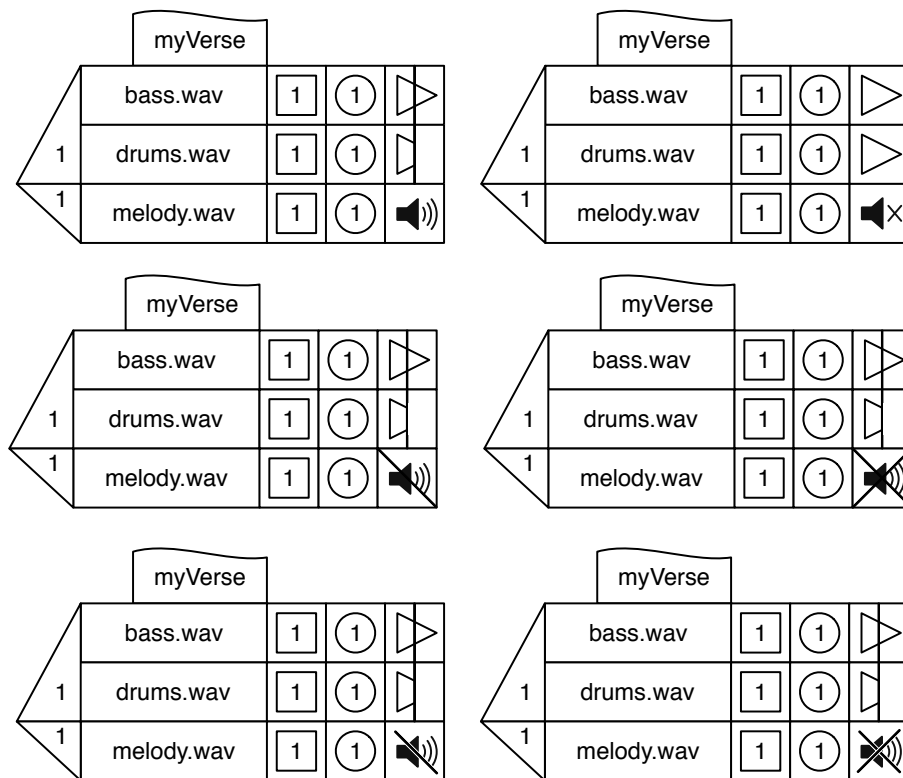


Figure D.23: A continuation of the candidate icons for the management of soundable content in Time Chooser lanes shown in Figure D.22. These tests use muted/unmuted speaker icons of the type used in operating systems such as iOS, MacOS, and Windows.

The ‘trim’ control, borrowed from low-threshold audio and video editing tools such as QuickTime Player, consists of two downwards-facing arrows which denote the points at which playback of the loaded file will start and end. If the user sets these arrows at the extremes the file will play in its entirety; this is the default setting, and is shown in Figure D.24. The user could use the ‘trim’ arrows to constrain the length of the sample at either the start or end of the sample. The number on the right of the ‘trim’ control optionally applies a grid, shown here as ticks along the horizontal trim line, to which the trim heads snap. Disabling the grid allows for freehand control. The arrowhead underneath the horizontal trim line is the anchor point; this represents the starting point of the sample, and allows for the sample start point to be moved independently of the overall length of the sample.

These, and similar refinements, were further developed for Choosers v3 and are presented in Chapter 10.

D.3 Design exercises used to inform Choosers v3

The following design exercises formed part of the development of Choosers v3; they are referenced in Chapter 10.

D.3.1 Shaded/filled weights vs. numbers.

Various icon designs were considered for Choosers v3; this included candidates for the stop icon, outlined in Section 10.2.2 and shown in Figure D.25. Multiple candidate representation of weights making use of shading were considered; an example is shown in Figure D.26. Using this system, a zero weighted lane would be represented by an unshaded weight icon, and an infinity-weighted lane would be a fully-filled icon. Fractional weights would be represented by partially-shaded icons

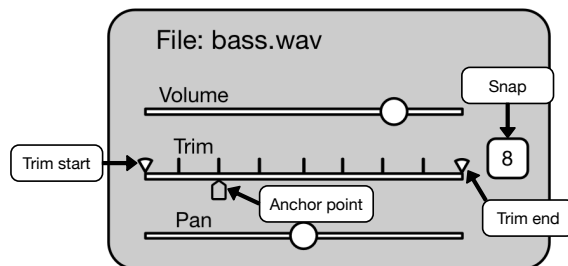


Figure D.24: Example popup controls associated with each lane in a Soundable Chooser. Volume and pan are shown: additional options include mixing desk-style controls such as filters and reverb level. The trim control allows the user to set the start and end point of the lane contents. The snap control is shown in a box to the right of the trim control—this sets the number of ‘slices’ that the faders will snap to. This can be clicked to disable the snap for freehand control.

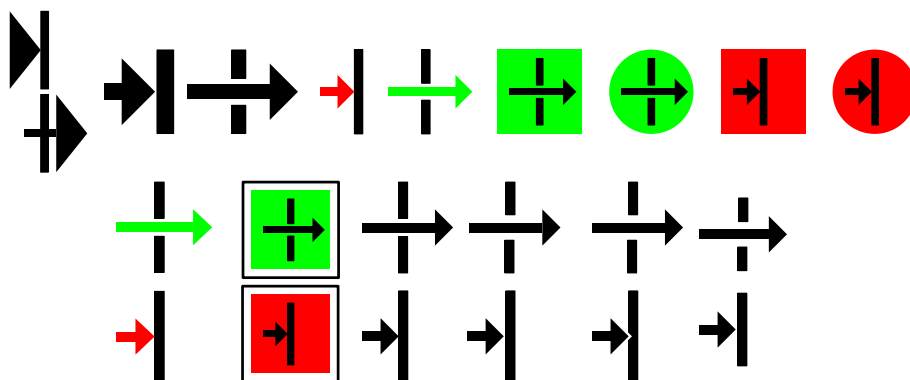


Figure D.25: Candidate stop icons from the process outlined in Section 10.2.2. The final design adopted in Choosers v3 is shown on the lower right.

showing various states, as though filled with water. The primary motivation for the shaded weight icons was to reduce the use of numbers in the notation. However, the use of shading for weights between zero and infinity introduces viscosity issues. A user may create a Chooser with three lanes, and set the weights to their preference. If a fourth lane is subsequently added, and if the user wishes the new lane to have a higher weighing than the existing lanes, they may need to reduce the weights in the existing lanes. This is unnecessarily complex when compared with the use of numbered weights, as a new lane's weight can simply be given a higher number. As a result, the shaded/filled weight icons were rejected.



Figure D.26: An alternative lane weight icon which uses progressive shading to denote relative weights, with ‘zero weight’ shown on the far left and ‘infinite weight’ on the far right.

D.3.2 Left-to-right sequencing via arrows

While the use of arrows to denote sequence is functional, a number of sketches were created as part of a process of reflection on the use of arrows in the type of diagrammatic representations shown in Section 2.10 of Chapter 2. In many of these systems arrows are used to denote nondeterminism, whereas Choosers contain nondeterminism in the primitive element. As a result, users may assume that a fan of multiple input or output arrows would be possible. An alternative to the use of arrows to sequence objects would be to use a ‘big arrow’ representing a left-to-right time sequence, as shown in Figure D.27. Such a system clearly shows the passage of time while indicating that fanning outputs is not possible. However, the ‘big arrow’ design was rejected as it does not allow for the significant flexibility and powerful secondary notation possibilities offered by the existing mechanism.

While the approaches taken in diagrammatic representations shown in Section 2.10 are of interest, it is important to note that Choosers are distinct from them in two important ways. First, nondeterminism is incorporated directly in the primitive element. Second, both precise and flexible control of musical time is included in the primary abstraction. As a result, Choosers do not need to represent nondeterminism in the same way. The ‘big arrow’ design was therefore not adopted for Choosers v3.

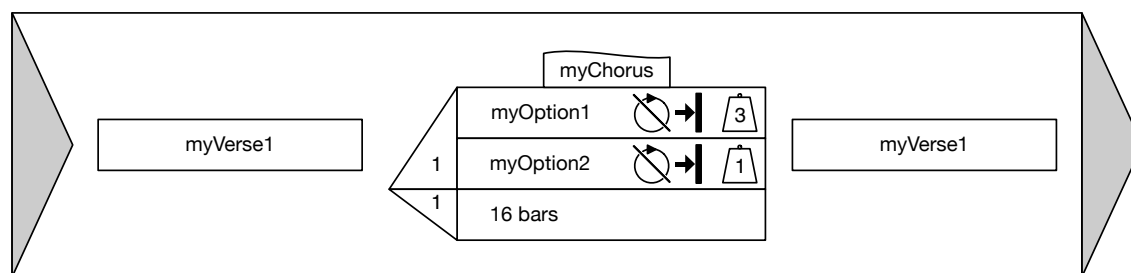


Figure D.27: A sketch showing an alternative to sequencing via arrows; a ‘big arrow’ in which Choosers can be placed to be played sequentially from left to right.

D.3.3 Design arc—programmatic control of Soundable Chooser lane parameters

The following section presents a series of speculative design sketches used to explore the programmatic control of Soundable Chooser lane parameters. Specifically, the sketches will be used to explore how the expressivity of Choosers could be improved by allowing finer-grained control of the types of parameters surfaced in the control panel overlay introduced in Section D.2.4. The annotated sketches explore whether detailed programmatic control of such parameters with a fine temporal granularity can be achieved while maintaining progressive disclosure of more complex features. The design sketches below will consider potential solutions and musical examples of programmatic control for the purpose of critiquing each stage and showing the motivations for successive iterations.

D.3.3.1 Speculative new Time Chooser behaviour: the Pulse Chooser

In order to facilitate temporal programmatic parameter control, early sketches considered speculative new semantics for Time Choosers when used in combination with Variable Choosers. In this alternative design, Time Choosers no longer control the duration of a Full Chooser but are instead set to loop, producing a regular metronomic pulse which is then used to trigger the attached Variable Chooser; this new behaviour requires additional threads. The combination of a looping Time Chooser and an attached Variable Chooser is referred to as a Pulse Chooser.

An early sketch of Pulse Choosers v1, showing three alternative notations, is shown in Figure D.28; note the rounded nose cone of the Variable Chooser and the Time Chooser's upwards-sloping triangular nose cone. The Time Chooser lanes make use of a loop icon, as used in Soundable Chooser lanes, to denote their new behaviour of providing regularly-repeating triggers. The Pulse Choosers in Figure D.28 show the three alternative and equivalent pulse duration notations; the user can write '16th notes' or 'semiquavers' in text, or a graphical semiquaver can be added to the Time Chooser lane. According to the currently set global tempo and time signature, the Time Chooser will now trigger the attached Variable Chooser at semiquaver/16th note intervals. The Pulse Choosers show the use of both a verbose ('random number from 0 to 15') and shorthand ('random 16') command; either will result in an integer between 0 and 15 being selected every time the Variable Chooser is triggered by the Time Chooser. The random number can then be accessed via the name given to the Chooser—in this case, 'Drum anchor'.

The speculative design of Pulse Choosers is potentially problematic as it risks confusing the fundamental function of a Time Chooser. Importantly, the sketch shown in Figure D.28 does not show how the resulting integer could be used to control any Soundable Chooser lane parameter. The following section will outline a candidate solution to this problem.

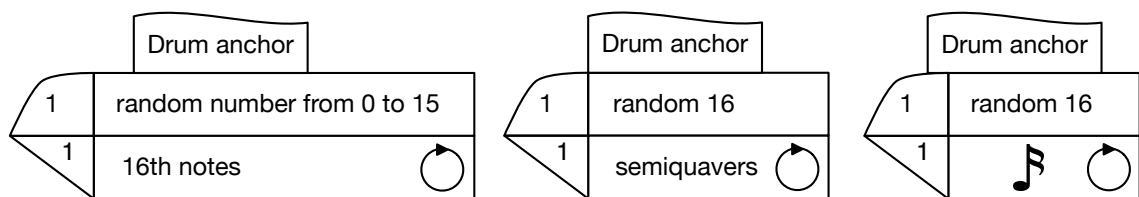


Figure D.28: Pulse Choosers v1: three equivalent sketches of a Variable Chooser (upper lane) being repeatedly triggered by a looping Time Chooser (bottom lane). The three Choosers demonstrate alternative equivalent notations.

D.3.3.2 New behaviour—Time Choosers and Soundable Choosers

The Pulse Chooser concept outlined above—using a looping Time Chooser attached to a Variable Chooser in a new way to repeatedly trigger it, thus modifying events in one or other Choosers that refer to that variable—led to the idea of exploring possible new roles for Time Choosers more

generally. For example, whereas up until now the sole role for a Time Chooser in a Full Chooser is to constrain its duration, a variety of other candidate roles are now worth considering. In part, these new possibilities arise from the fact that the availability of variables for entries in lanes mean that lane contents and parameters need no longer be static; they can be variable, or indeed computed (e.g. by using expressions built using arithmetic or other operators). Putting this idea together with the idea of possible expansions of the role of Time Choosers, it might be interesting to consider ‘operations’ that a Time Chooser might send repeatedly to the Soundable lanes it is attached to at a frequency determined by its static or calculated duration. For example, such a Time Chooser might increment the value of a variable (see Figure D.32 for an example).

The sketch in Figure D.29 shows a Full Chooser being ‘pulled apart’ to show the ‘operation’ that is sent to the Soundable Chooser when the duration in the Time Chooser lane has elapsed. In this instance, once the Time Chooser’s duration has elapsed it will send a ‘stop’ message to the Soundable Chooser; this is the normal behaviour of a Full Chooser, and is the only behaviour available until and unless the user ‘pulls apart’ the Chooser to override the operation.

The sketch in Figure D.30 was used to explore how the ‘pulled apart’ design may be fruitfully generalised so that a Time Chooser might send other messages. In this generalisation, the Time Chooser acts as the ‘trigger’, either when the chosen duration elapses when the Time Chooser is set to not loop (the default behaviour), or on each pulse if the Time Chooser is set to loop (see Section D.3.3.1 above). This trigger then results in an ‘operation’ which acts on the ‘receiver’.

The ‘pulled apart’ Chooser design is problematic due to a lack of granularity and visibility regarding the receiver of the operation. The ‘operation’ layer controls the entire attached Soundable or Variable Chooser, but a user could reasonably assume that the ‘operation’ only applies to the single lane to which it is attached; that is to say, the bottom lane of a Soundable or Variable Chooser with more than one lane. In addition, there is no mechanism to control a specific parameter in an arbitrary Soundable Chooser lane. These issues are to be addressed in the next design iteration.

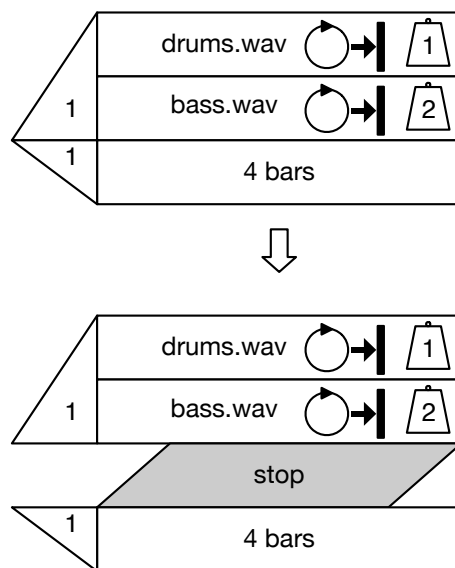


Figure D.29: A sketch showing a Full Chooser being ‘pulled apart’ to show the type of message sent from the Time Chooser to the Soundable Chooser.

D.3.3.3 Controlling the anchor point via Pulse Choosers v2

The speculative design outlined in the previous section (Section D.3.3.2) showed the outline of a possible method to exert programmatic control over a Chooser. The present section will expand this design into Pulse Choosers v2, a development designed to directly control one specific parameter. In this expansion of Pulse Choosers v1, an ‘operation’ will run each time the looping Time Chooser

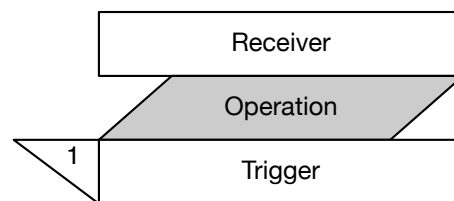


Figure D.30: A generalisation of the ‘pulled apart’ Chooser which will form the basis of Pulse Choosers v2.

fires. The operation will specify a receiving parameter, which may be a parameter of any current Chooser. A horizontal slider in the ‘receiver’ lane will then display the current parameter setting.

Figure D.31 shows an example Pulse Chooser v2 sketch in which a looping Time Chooser is outputting a metronomic pulse of 8th notes/quavers. Each pulse will result in the selection of a random number between 0 and 7 (via the random 8 command), and this random number is used to set the anchor point of a Chooser lane named drums1. The sketch shows the ‘receiver’ lane at the top as containing a single horizontal fader which shows the current setting as well as the names of both the lane and the parameter being controlled.

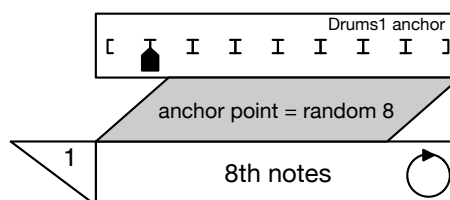


Figure D.31: A Pulse Chooser v2 sketch. Each time the looping Time Chooser fires the operation layer will produce a random number (in this case, a random number between 0 and 7) which is then used to set the anchor point. Note the infinite loop setting in the Time Chooser lane, meaning that the Time Chooser will repeatedly trigger the selection of a new random number every 1/8th of a bar.

A second example of the Pulse Chooser v2 design is shown in Figure D.32. This example is based on Steve Reich’s *Clapping Music* (1980), a phase music composition. This implementation uses two samples, clap1.wav and clap2.wav, which are two performances of the same rhythm. The uppermost lane, clap1.wav, plays without any change. The lower lane, clap2.wav, has its anchor point controlled by the Pulse Chooser at the bottom of the image. The Time Chooser will select a duration between 4 and 32 bars and, once this elapses, it will increment the anchor point by one. This process repeats 12 times until the two samples are back in phase.

Pulse Choosers v2 offers a solution to the problem of programmatically controlling a single parameter, but there are some issues which remain unclear. Firstly, the design is limited as it allows for only one controllable parameter. Secondly, the method by which the receiving parameter could be selected is unclear.

D.3.3.4 Simultaneous programmatic control of multiple parameters

At this stage in the speculative design arc, the next development was to explore how programmatic control of multiple anchor points could be used to create complex rhythmic and harmonic patterns from existing audio files. The first sketch to achieve this made use of multiple Pulse Choosers (all using v2 of the design), each controlling the anchor point of an audio sample in a specified Soundable Chooser lane. The sketch shown in Figure D.33, using samples from *Discipline* by Nine Inch Nails (Reznor, 2011), was implemented via a simple back-end written in *SuperCollider* to help critique the design. The output from the back-end can be heard in an audio example (Bellingham, 2020a), in which the user adds or removes lanes from playback by changing the Soundable Chooser

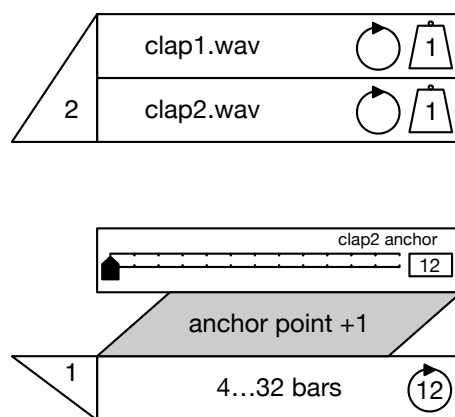


Figure D.32: A modified version of Steve Reich's *Clapping Music* (1980) using the Pulse Chooser v2 design. The anchor point of `clap2.wav` is incremented by 1 each time the Time Chooser's duration elapses. A new duration is selected each time. Reich's original piece can be played by replacing the random duration with a fixed value (e.g. 8 bars).

lane weights; a weight of zero removes a lane from selection.

The Soundable Chooser on the left of Figure D.33 will play five lanes simultaneously; note that two lanes are populated by one sample, which will result in that sample playing twice. The user can easily remove one or more of the lanes from selection by setting the lane weight to zero. During playback, the anchor points of four of the five lanes will be programmatically controlled by the Pulse Choosers on the right of the figure. Each of the Pulse Choosers on the right selects a random number when triggered by the attached looping Time Chooser lane. In this sketch, the random number is set to ensure the anchor point can be selected anywhere over the entire length of the sample; this is done by multiplying the note division by the length of the sample in bars. For example, the `drums.wav 2` Chooser on the top right of Figure D.33 selects a random number calculated by multiplying the note division (in this instance, 16) by the duration of the sample in bars (8). This is equivalent to issuing the command `random 128`, and will result in the selection of a random integer between 0 and 127 every time the Time Chooser fires. In this way, the Time Chooser controls both the frequency of selection and the length of sample playback before the next selection.

The user may initially create the relationships between Pulse Choosers and parameters via a Direct Combination (Holland and Oppenheim, 1999) interaction; see Section 2.9 in Chapter 2. For example, if a Time Chooser were to be dragged and dropped onto a Soundable Chooser lane, the system could then show a list of the lane parameters which could be controlled by the resulting Pulse Chooser.

Simultaneous programmatic control of multiple parameters seems to offer interesting musical flexibility. The multiple Pulse Chooser design shown in Figure D.33 leads to a large number of UI widgets; this may have the beneficial result of improving the software's visibility as all programmatically-controlled parameters are made visible. However, the design suffers from hidden dependancies as each Pulse Chooser-controlled parameter can be placed anywhere in the UI without a clear link to the lane being controlled. In simple systems with a limited number of Choosers the user may be able to create a workable layout, but the secondary notation of the system is likely to be harmed in more complex designs which contain a large number of Choosers. In addition, the horizontal fader used to visualise each anchor point becomes difficult to read when the number of steps (note duration \times bar length) is too high to be distinguishable; an example of which can be seen in the `drums.wav 2` Pulse Chooser on the upper right of Figure D.33.

So far in this design arc, random values have been used to set parameters; the next sketch considers a design to allow for selection from a list. The sketch in Figure D.34 shows the selection of a number from a list populated in a Variable Chooser. While this design is functional and meets the design goals of both consistency and parsimony, the addition of a textual language could offer

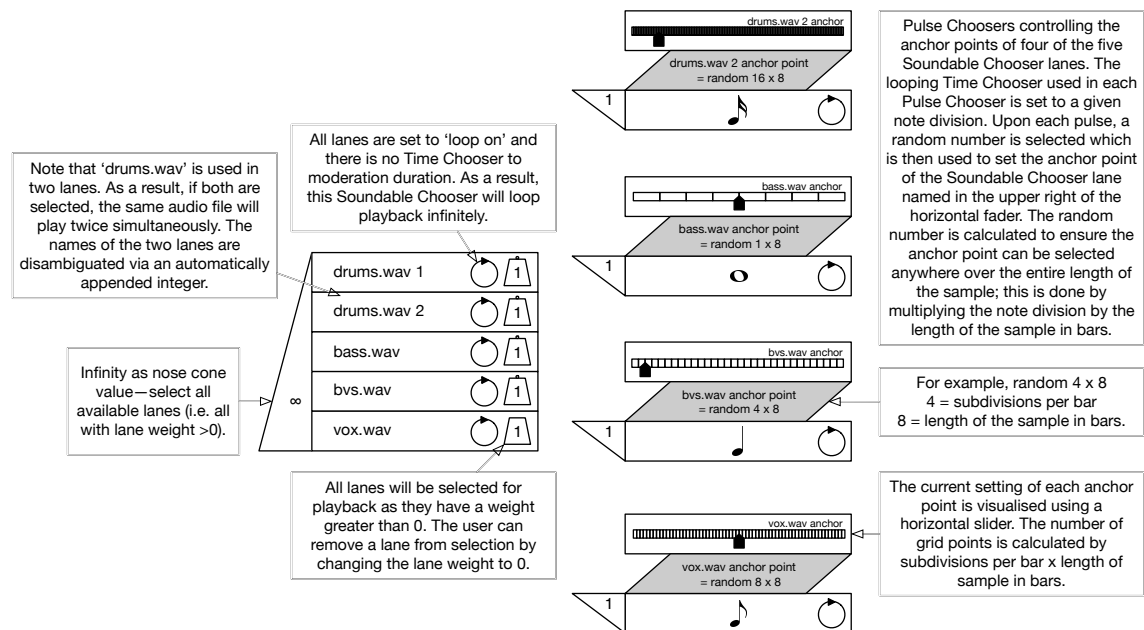


Figure D.33: Rock example using speculative design. As shown, the Soundable Chooser (left) will play all five lanes due to the use of ∞ in the nose cone; all lanes are selectable as their lane weights are greater than 0. The user can optionally use the Soundable Chooser's lane weights to remove ($=0$) and add (>0) lanes from selection; audio of this figure, demonstrating an example in which the user controls lane weights to add and remove samples to create an improvised musical arrangement, is available online (Bellingham, 2020a) – Rock example using speculative design.wav. The anchor points of four of the five Soundable Chooser lanes are programmatically controlled by the Pulse Choosers on the right of the figure. Each Pulse Chooser is rhythmically triggered by a looping Time Chooser; upon firing, a random number will be selected and used to set the anchor point of one of the Soundable Chooser lanes. While capable of surprisingly rich output, the use of multiple UI widgets leads to hidden dependencies and various legibility issues (e.g. the high number of grid points in the drums .wav 2 Pulse Chooser on the top right of the figure).

advanced users a more succinct alternative with sufficient expressivity. The learnability and usability of the basic functionality of Choosers is maintained by allowing more powerful functionality via progressive disclosure.

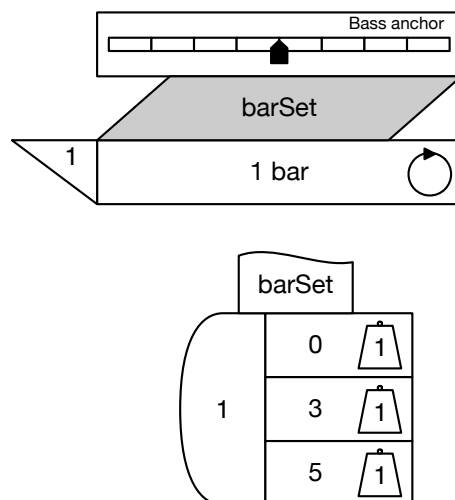


Figure D.34: Using a Variable Chooser to populate the operation layer of a Pulse Chooser.

D.3.3.5 Textual commands

The design of Choosers has so far focussed on making musically meaningful actions possible with optimal clarity via the use of the minimum possible number of elements. A lightweight language could be added to Choosers to allow for a limited range of meaningful control of an array of values—specifically, random selection with or without repetition, and incrementing and decrementing through the array with differing looping/cycling behaviour. The speculative lightweight language, presented below, draws on the features and terminology of music software and hardware (e.g. arpeggiators). The addition of more complex textual expressions refers back to the usability/expressivity trade-offs (Repenning and Ioannidou, 1997) discussed in Section 2.7, and would present a different set of trade-offs. The textual commands augment the existing tools, and offer more advanced users a more concise control and allow for new functionality. Such users do not require the same degree of visual support as novice users. The language allows for a limited range of meaningful control of an array of values—specifically, random selection with or without repetition, and incrementing and decrementing through the array with differing looping/cycling behaviour.

Random and shuffle Random values are to be selected from an array. The array can be specified by the user, or can be created using the `..` shorthand to fill in values between specified extremes. For example, `{1, 2, 3, 4, 5}` and `{1..5}` are equivalent. Random selection can result in repeated selection of one value. Random selection without repetition is provided by `shuffle`, with the word chosen to reflect its use in music playback software such as *iTunes* and *Spotify*.

2 Number 2

`random{1..10}` or `r{1..10}` Pick a new number from 1 to 10 each time

`shuffle{1..10}` or `s{1..10}` Pick a new number from 1 to 10 each time with no repeats

`random{1,2,4,6}` or `r{1,2,4,6}` Randomly pick between the listed numbers afresh each time

`shuffle{1,2,4,6}` or `s{1,2,4,6}` Randomly pick between the listed numbers each time with no repeats

Arpeggiator-style commands The arpeggiator, a feature of many synthesisers, is used as the model for this implementation of incrementing and decrementing through the values in an array.

`++{1..10}` or `++{1,2,4,6}` Increment through the enumerated options. When at the highest value of the array, cycle back to the lowest value and repeat.

`--{1..10}` or `--{1,2,4,6}` Decrement through the enumerated options. When at the lowest value of the array, cycle back to the highest value and repeat.

Mathematical operations Simple mathematical operations can be applied to all of the textual commands. For example, `s{10} + 20` is equivalent to `shuffle{20..30}`; both will randomly select values between 20 and 30 without repetition.

D.3.3.6 Addressing individual parameters via the ‘drawer’

The sketches above have shown that addressing individual parameters per lane can lead to musically interesting results, but also that multiple parameter control can lead to various issues (e.g. problems caused by a proliferation of widgets; see Section D.3.3.4). The sketch in Figure D.35 shows one possible solution to these problems—a drawer, optionally ‘pulled out’ from the right-hand edge of the control panel overlay (see Figure D.24) to reveal textual control of the lane’s parameters. The drawer adds the option of textual control via progressive disclosure, and the drawer handle shows the user where to access the additional controls. Pleasingly, the visual design hints at the drawbars of a Hammond organ.

The example sketch in Figure D.35 shows the control panel overlay for a Soundable Chooser lane at the top, and a Variable Chooser named `vv23` at the bottom. The Variable Chooser will select a new number on every beat; the new number will be either 2 or 4, with equal likelihood of selection. The Soundable Chooser lane’s anchor control is set to a grid of 8, and the output of the Variable Chooser named `vv23` is being used to control the anchor point. As a result, the anchor point will move to grid position 2 of 8, or 4 of 8, on every beat.

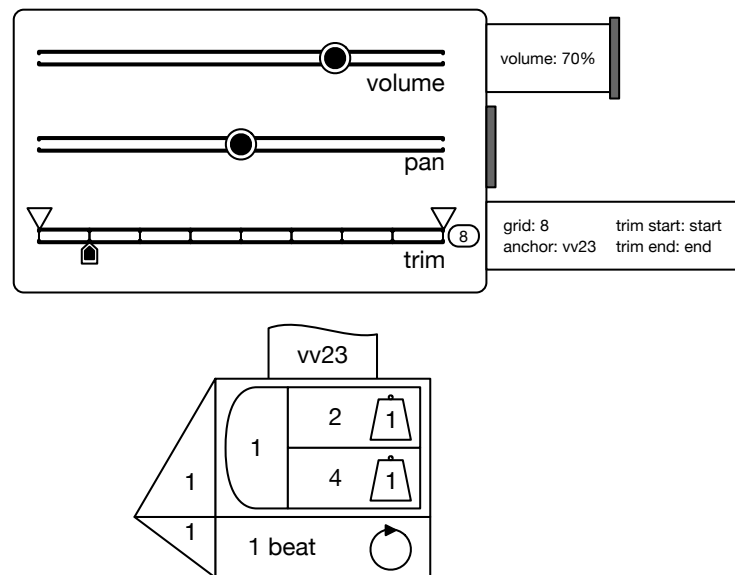


Figure D.35: An example showing the use of drawers to access parameters for textual control, and an example of a referenced Variable Chooser—`vv23`. The Variable Chooser will be repeatedly triggered on every beat and will select one of two numbers (2 or 4). This number is then used to set the anchor point.

Figure D.36 shows an implementation of Reich’s *Clapping Music* (1980) in which the anchor point of the `c1ap2.wav` lane can be manually moved to change the phase of the musical phrase. The next sketch, shown in Figure D.38, introduces a Variable Chooser to programmatically trigger the anchor point increment. A Variable Chooser is used to increment the anchor point by 1 each time the `myV` Chooser fires—the Time Chooser will select a random duration between 4 and 32 bars,

chosen afresh each time. An early draft of the textual language presented in Section D.3.3.5 was also sketched, and is shown in Figure D.38. This example defines a variable, *myAnchor*, and which will increment by 1 on each repeat. The variable is then used to control the anchor point.

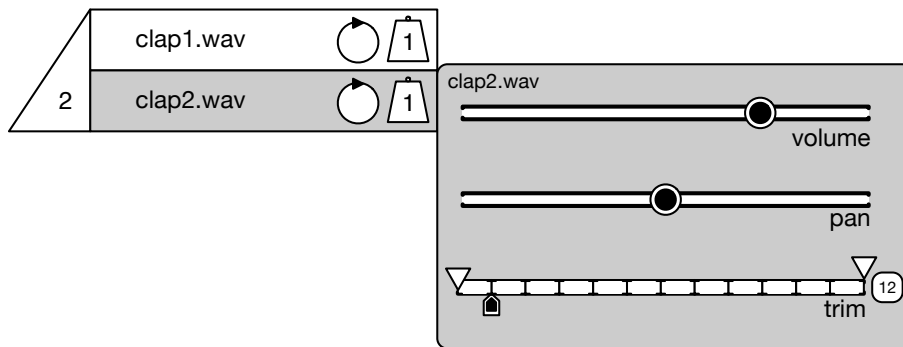


Figure D.36: Steve Reich's *Clapping Music* (1980) implemented using manual control. The user can move the anchor point of *clap2.wav* to a grid point, set using the control to the right of the trim slider. In this instance the user can move the controls to one of 12 grid points.

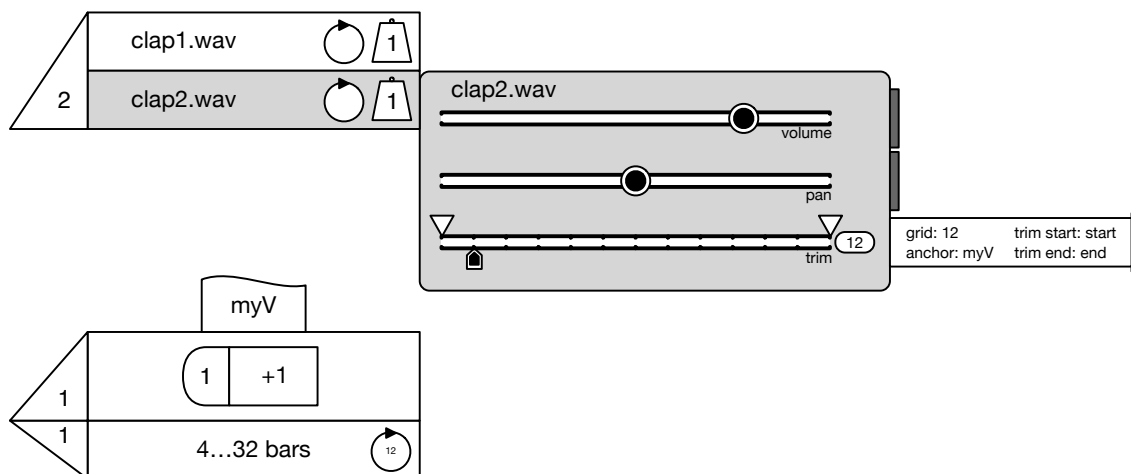


Figure D.37: A development of Figure D.36 with a Variable Chooser (*myV*) used to control the anchor point of *clap2.wav*. *myV* will be triggered 12 times, with an interval between 4 and 32 bars chosen afresh. Each triggering will increment the variable value by 1.

D.3.3.7 Reflection on design arc

The above design arc introduced, annotated, and critiqued a number of sketches used to explore potential designs for programmatic parameter control using Choosers. The arc makes significant use of a new Chooser, the Pulse Chooser (see Section D.3.3.1), designed to repeatedly trigger an attached Variable Chooser. The sketches make use of a loop icon in the Time Chooser to denote this new behaviour and to differentiate it from standard Time Choosers; the use of the loop icon in a Time Chooser lane was motivated by design consistency. However, allowing Time Choosers to have two distinct behaviours is potentially problematic as it risks confusing the user.

The final design in the arc, the 'drawer', meets the arc's initial design goals; the drawer improves the expressivity of Choosers by allowing programmatic control with temporal granularity, while maintaining progressive disclosure. While early sketches in the arc considered controlling parameters directly using Time Choosers and an 'operation' layer (see Section D.3.3.2), the 'drawer' design was developed to solve a number of the issues present in the earlier sketches. The 'drawer' leverages a clear metaphor for revealing and hiding advanced controls. The 'drawer' lowers the

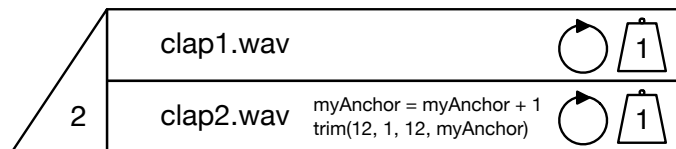


Figure D.38: A sketch exploring the use of the draft textual language to increment the anchor point of `clap2.wav`. The textual statement in the lower lane of the Chooser defines `myAnchor` as incrementing by 1 on each repeat. The variable `myAnchor` is then used in the trim control as the fourth setting (grid, start point, end point, anchor).

number of hidden dependencies when compared with the earlier sketches as the location for each parameter is clear; this potentially reduces error-proneness. Earlier sketches (e.g. Figure D.33) required the user to freely place Choosers in the UI but, while free placement can be beneficial to secondary notation, it places a greater burden on the user, and a complex piece of music may have a high number of Choosers in the UI. In comparison, the ‘drawer’ design provides a consistent location for lane parameters, meaning that the secondary notation is not contingent on the expertise of the user. Finally, the high visibility of parameter settings provided in the early sketches (e.g. Figure D.33) is necessarily reduced when the same parameters are moved to a ‘drawer’, which is a potential negative consequence of the drawer design. However, the user can still access multiple parameters simultaneously by opening several drawers at once, and the trade-off between reduced parameter visibility and the positive attributes of the ‘drawer’ was deemed acceptable.