Towards the Realisation of a fully integrated Interactive Computer Music System (ICMS), adopting Transformative Expressive Dimensions

George Meikle

Doctor of Philosophy in Music

University of Salford

School of Arts & Media

2017

Contents

List of Figures

Acknowledgements

I would like to thank the University of Salford and Arts and Humanities Research Council for providing me with the opportunity, resources and funding to carry out my research. Special thanks go to Prof. Stephen Davismoon for the impeccable level of support and guidance with which he has provided me throughout the duration of the research project, as well as the multitude of opportunities beyond the scope of my research that he has opened up and made available to me. I would also like to thank Prof. Alan Williams and Mr. Brendan Williams for their vital input in helping to inform and shape my research at various stages in its development.

Declaration

Large parts of the verbatim used in sections 2.3 Early and Experimental Interactive Works and Systems, 2.4 Web and Touchscreen-based Interactive Music Applications and Games, and 2.5 HCI in Popular Electronic Music Records/Releases and Music-Creation Applications/Virtual Instruments of the Literature Review have been published in *Contemporary Music Review, 35* (2) in the article "Examining the Effects of Experimental/Academic Electroacoustic and Popular Electronic Musics on the Evolution and Development of Human–Computer Interaction in Music" (Meikle, 2016). Other areas also include paraphrasing of the published text. The inclusion of this text is done so in accordance with the Taylor & Francis Publishing Agreement, which states that the author retains the right to include the article in a thesis or dissertation that is not to be published commercially. In-text citations have been used to indicate the appropriate sections of the text.

List of Abbreviations

HCI	Human-computer interaction
ICMS	Interactive computer music system
UI	User interface
GUI	Graphical user interface
CAAC	Computer-aided algorithmic composition
MIDI	Musical Instrument Digital Interface
CC	Control change
OSC	Open Sound Control
IP	Internet Protocol
DAW	Digital audio workstation
VST	Virtual Studio Technology
BCI	Brain-computer interfacing
CD	Compact disc
WAV	Waveform Audio File Format
EEG	Electroencephalography
MMU	Manchester Metropolitan University

Abstract

The principal aim of conducting this research project is to advance the field of human-computer interaction (HCI) in music through the inception of new and exciting ideas relating to the conceptual and aesthetic values and characteristics associated with interactive computer music system (ICMS) design, development and implementation. This exploratory investigation has been carried out through the continued development of a unique screen-based ICMS, *ScreenPlay*, which brings together aspects of many prominent, pre-existing system design models along with other novel inclusions, such as the capability of operating as both a multi-user-and-computer collaborative, improvisatory interactive performance system and a single-user-and-computer studio compositional tool for Ableton Live, and the implementation of Markovian generative and topic-theory-inspired transformative algorithms to provide new ways of breaking routine in collaborative improvisation and generating new musical ideas in composition, as well as providing new dimensions of expressivity. The intention being that the culmination of these efforts should be the establishment of a system design model that offers the user(s)/performer(s) a significantly more engaging, intuitive and complete interactive musical experience than that afforded by any currently available system.

This is not an objective that should be perceived as being born of arrogance or ignorance; many established and commercially successful ICMSs provide users with amazing and unique experiences, and the value of their contribution to the field of HCI in music is not to be underestimated or taken for granted. However, for all the relative strengths and possibilities of interacting with these systems, there is potential for improvement and evolution in equal measure. This is largely due to the tendency of ICMS developers to focus (sometimes almost exclusively) on providing the best possible experience when engaging with a system only with regard to a single parameter/characteristic of the musical output at the expense of providing depth-in-control at any level over some or any of the many other parameters/characteristics available. It is necessary for all forms of technology and art to continually improve and evolve beyond what has already been achieved in order to avoid extinction, and nowhere is this more apparent than within an innovative and niche field of research such as HCI in music.

1. Introduction

> join[ing] the mechanical power of the machine to the nuances and "subjectivizing" control of the human performer … is itself an aesthetic value for our new age. This ultimately will be the way music will need to move until the next big issues arise. (Garnett, 2001, p. 32)

HCI in music is a field of research that has been extensively documented over the course of the last few decades; dating back at least as far as the 1980s, as evidenced by Michel Waisvisz's sensory-based ICMS, *The Hands* (1984; 2006), and the work of "Computer Network Music" ensemble *The Hub* (Bischoff, Perkins, Brown, Gresham-Lancaster, Trayle & Stone, 1987-present; Early Computer Network Ensembles, n.d.; Brown, n.d.). It is, however, only in the relatively recent history of HCI in music that its relevance and impact within widespread popular culture has been recognisable; largely due to the massive surge in popularity of electronic dance music as well as the advent of powerful, portable, affordable and readily available touchscreen devices throughout the world. Through remixes, DJ sets and, more recently, live remixing popular electronic/dance music has brought about a new attitude and acceptance on the part of the audience/listener with regard to the open interpretation of musical works and improvisation in live performance. This ideology opposes that of composers during the structuralist era which, supported by the likes of Stravinsky (1947), proclaimed that deviation in recital from the score was unacceptable throughout a period that ranged from the 1890s up until the 1950s. First challenged in popular music by rock bands that discarded the score in favour of the recording as the primary musical artefact, the necessity for a primary musical artefact at all was then brought into question by the developments in electronic dance music described above. It is this prevailing attitude within popular electronic music/club culture, along with the predominantly loop-based structure and relatively rigid form of most dance music, which means it is perfectly suited both ideologically and stylistically to the design of ICMSs intended for use by non-expert users and novice musicians. In recent years this has led many artists to release songs, EPs and albums not as traditional recordings either via CD, vinyl or through digital download stores but instead as interactive applications; examples of which include *Biophilia* (Björk, 2011), *Reactable Gui Boratto* (Boratto, Jordà, Kaltenbrunner, Geiger & Alonso, 2012) and *Reactable Oliver Huntemann* (Huntemann, Jordà, Kaltenbrunner, Geiger & Alonso, 2012). Releasing music in this format has the advantage of going some way to replacing the tangibility of CDs and vinyls in a music business becoming ever more reliant on digital downloads and subscription streaming services by offering the consumer a unique and exciting experience that also has the potential to provide a more attractive alternative to illegal peer-to-peer file-sharing and torrenting of music. Interestingly, it also draws parallels with Roland Barthes' theory of two musics (1977, p. 149) and the music publishers of Tin Pan Alley who, in the late 19th and early 20th century, sold songs not as recordings but as lead sheets to be played back on a piano in the homes of the consumers as a collaborative/social pastime for families and friends.

My own personal interest in HCI in music stemmed, initially, from studying composition as a computer musician at undergraduate level with a focus on popular electronic music production. At postgraduate level my compositional studies branched out to include a research project into the open interpretation of electronic dance music as outlined above, as well as more experimental electroacoustic works during a project for which I, along with three other students, was commissioned to compose the accompanying music to an audiovisual installation inside Queen Street Mill in Burnley between 2011 and 2012, *Ways of Seeing: Interwoven Lives* (Kershaw, Kelledy, Meikle, O'neill & Reiser). The project explored the impact the mill had on the surrounding community through the memories of individuals who had either worked there themselves or were friends/relations of those who had. Both of these studies, along with undertaking a module relating to visual programming within modular programming environment software such as Max/MSP and Patchwork Graphic Lisp (PGWL), influenced my desire to conduct research into the field of HCI in music. The two projects entailed composing and mixing for playback via multi-channel installations (5.1 the former, 8-channel the latter); the first focussing on differing approaches to the open interpretation of musical works through improvisatory performance techniques, including loop-based live remixing with a button-matrix MIDI controller and the use of generative algorithms to transform and evolve the source material of original works; the second involving the composition of multiple accompanying soundscapes to specific sequences of images and interview recordings to be isolated from each other at various locations within the installation space.

The idea to expand upon the conceptual ideas and design-aesthetics of many pre-existing ICMSs in *ScreenPlay* was fuelled, in part, by my involvement in a workshop with Francesco Giomi - an Italian composer and conductor of experimental electronic music - who has developed an extensive code of hand signals that he uses to direct performers in improvisatory performance in order to craft variation in melody, harmony, and form and structure: the key aim of the inclusion in *ScreenPlay* of the Markovian generative and topic-theory-inspired transformative algorithms. The decision to utilise elements of topic theory in order to facilitate this functionality is due to the textural/timbral potentialities associated with electronic music through synthesis and audio effects processing, as well as the suitability of using metaphorical language to express each of the topical opposition transformations to both novice and expert users alike: "stability-destruction"; "open-close"; "light-dark"; "joy-lament".

In order to properly frame and contextualise the research project I asked the following question: is it possible to conceive of an ICMS that achieves in its design and implementation that which has not yet been accomplished by current and pre-existing systems; catering to both novice and experienced users and musicians, and using only existing technology so as to maximise its impact in the fields of HCI in music and music technology? *Reactable* (Jordà, Kaltenbrunner, Geiger, & Alonso, 2003-present), *NodeBeat* (Sandler, Windle & Muller, 2011-present) and *Incredibox* (So Far So Good, 2011-present) are just a few examples of ICMSs that each exhibit the characteristics of only one of the three overarching approaches to ICMS design: *sequenced*, *transformative* and *generative* (Rowe, 1994; Drummond, 2009). A common hindrance to the vast majority of ICMSs, *ScreenPlay* seeks to combat this exclusivity of focus through the encapsulation and evolution of the fundamental principles behind the three system design models in what is a novel approach to ICMS design, along with the introduction of new and unique concepts to HCI in music in the form of the aforementioned topic-theory-inspired transformative algorithm and its application alongside the Markovian generative algorithm in breaking routine in collaborative improvisatory performance and generating new musical ideas in composition, as well as providing new and additional dimensions of expressivity in composition and performance. While the multifunctionality of the system, which allows it to exist as both a multi-user-and-computer interactive performance system and single-user-and-computer studio compositional tool - including the affordance of dedicated GUIs to each individual involved in collaborative, improvisatory performance when in multi mode and the technicality of hosting up to sixteen separate users through a single instance of Ableton Live in order to achieve this - is another of *ScreenPlay*'s novel achievements in respect to ICMS design.

2. Literature Review

2.1 Philosophical/Aesthetic Context for Open Interpretation and Interactivity in Contemporary Arts

The concepts underpinning contemporary examples of interactivity in music are, predominantly, those essential to post-structuralist theory, including its approach to the open interpretation of works within the arts. This idea is summarised by John Storey (1997, p. 89), who states that 'Post-structuralists reject the idea of an underlying structure upon which meaning can rest secure and satisfied. Meaning is always a process … [it] is a momentary stop in a continuing flow of interpretations of interpretations.' Furthermore, Storey cites in support of this the contention between Ferdinand de Saussure's structuralist theory that language be divided into *signifier* - the written or verbal word - and *signified* - the associated mental image/"sound image" or physical object/action/characteristic etc. with said word (Saussure, 1916/1959 [3rd ed., English translation], pp. 66-67; Bredin, 1984, p. 67; Storey, 1997, p. 73) - and that of post-structuralism whereby 'Signifiers do not produce signifieds, they produce more signifiers' (p. 89). The 'polysemic nature of signs' (p. 83) was first recognised by Roland Barthes in *Mythologies* (1957/1972 [English translation]/1991 [25th ed.]) when he initially coined the concept of secondary signification, or "myth" (1991, p. 113). In *Elements of Semiology* Barthes explains the link between primary/secondary signification and denotation/connotation, declaring that 'The signifiers of connotation … are made up of *signs* (signifiers and signifieds united) of the denoted system' (1983, p. 91); further elaborating with the idea that the signifieds of connotation are formed by ideology, and the signifiers by rhetoric (p. 92) - indicating that myth is the agent through which an author of a work is able to impose a particular slant or bias on the message it conveys (1991, p. 115). A point best summarised by Boer, who states that 'myth is not determined by the content of its message, but the way the message is told.' (2011, p. 215)

Umberto Eco (1989) expands on these ideas by outlining the distinction between an *open work* and *work in movement*: an open work being one which, despite being complete in its structural entirety, is open to an infinite number of possible interpretations by the audience - usually subject to the influence of internal and external factors such as the mood, tastes and intuition of the addressee as well as the environment and time-frame within which a work is being observed; while a work in movement can be defined as one which requires from the observer a level of collaboration with the creator in piecing together the intentionally disjointed fundamental components of the work. Eco also makes clear the disparity that exists between the openness in perception of any real-world interaction with a work or object that, by design, has no intentionally subjective openness and that of the pre-meditated, infinitely variable open form characteristic to open works and works in movement. The work of Luigi Pareyson (1960, cited in Eco, 1989, pp. 21-22) is referenced in support of this, who states that any of the infinite permutations existent within the finite structure of an open work or work in movement is tantamount to a complete embodiment of the work so long as there is a recognition by the addressee that numerous other permutations exist and must be actively sought out in order to deepen the understanding of the work.

Deleuze & Guattari's book *A Thousand Plateaus* (1980) could be considered a literary example that is simultaneously representative of the qualities attributed to both an open work and a work in movement. The chapters in the book are 'present[ed] … as a network of "plateaus" that are precisely dated, but can be read in any order' (Massumi, 2005, ix, in Deleuze & Guattari, 2005). In effect, the reader is free to re-configure these "plateaus" in a multitude of different formations to result in various embodiments of the complete work. However, the book has also been realised in its totality by the authors, meaning that it can only be considered complete once all of the chapters or "plateaus" have been explored by the reader. *A Thousand Plateaus*, in many ways, epitomises the post-structuralist ideology that coherency in form and structure is secondary to content and openness in a work. This is achieved via an overarching, network-based construction founded on the concept of *nomad thought* - a school of thinking in direct opposition to that of *state philosophy* whereby the field of perception and scrutiny can be described as '"smooth," or open-ended' as opposed to '"striated" or gridded.' (p. xiii)

It is this ideology, so evident in *A Thousand Plateaus*, that calls into question the integrity and esteem of the author. According to Barthes (1977), the structuralist view on the relationship between an author and his or her work is that a work's significance can be attributed to the character, background or circumstances etc. of the individual responsible for creating it and, additionally, that such a standpoint works in aid of simplifying the critical process. In support of this he cites the regularity with which the inspiration and motivation behind Van Gogh's work is associated with his mental state. In contrast Barthes argues that it is, in fact, the audience within which the meaning of a work resides: meaning is not predetermined by the creator but is instead interpreted by the observer in any number of ways during the moment in which he or she examines the work. This is a point echoed by Storey, who states that 'A text is a work seen as inseparable from the active process of the intertextuality of its many readings.' (1997, p. 90)

Storey also notes that Barthes, in relation to literature, points out how the context of publication of a work and its target demographic can influence the interpretation of that work in the eyes of the reader depending on their personal beliefs and prejudices, social/cultural heritage and political inclination (Barthes, 1991, pp. 125-126; 1977, p. 29; p. 46; Storey, 1997, pp. 84-86) (much in the same way as Eco's concept of an open work); as has already been alluded to when discussing Barthes' notion of the role played by ideology in secondary signification/connotation/myth. In terms of HCI in music, and *ScreenPlay* in particular, this is an important point in relation to graphical user interface (GUI) design: a gamer will likely have differing expectations as to what may be an appropriate manner in which to visually represent the control surface of an ICMS and its affordances to the user - to be discussed in more depth in section six of the Literature Review (Categorisation and Approach to Design of ICMSs) - than would an electronic musician, classical musician, or non-musician etc.. Likewise - and in spite of being conceived without any intention of being applied to the open interpretation of works within the arts or giving any consideration to musical relevance - structuralism's binary opposition model can also be applied to ICMS design and the relationship between anticipated and intended functionality of GUI control surfaces/screen-based interfaces. This is due to the fact that not only is the binary opposition model exemplified by Saussure's signifier/signified theory but is inclusive also of the syntagmatic (combination) and paradigmatic (substitution) series (Saussure, 1959, pp. 122-125) - syntagmatic being 'the linear relationships between linguistic elements in a sentence'; paradigmatic being 'the relationship between elements within a sentence and other elements which are syntactically interchangeable' - also referred to by Saussure as *associative relations* (Appignanesi & Garratt, 2013, p. 60; Saussure, 1959, p. 123) - which, in turn, are directly linked to the opposing ideas of metonymy and metaphor respectively (Jakobson & Halle, 1956, pp. 76-82). In addition, the learned cultural codes and conventions upon which Saussure's signifier/signified binary opposition model is reliant (Saussure, 1959, pp. 65-70) bear particular relevance to the choices made in relation to both the mode of interaction with the *ScreenPlay* GUI and its topic-theory-inspired transformative algorithm; the topical oppositions of which are, in fact, themselves binary oppositions.

The manifestation of Structuralism's own view on the interpretation of works within music, however, is highlighted by the ideology, championed by many composers and philosophers during the first half of the 20th century, that the written score should be considered the primary musical artefact of a musical work and the performer of a piece has an ethical obligation to faithfully recite it without the addition of any improvisatory flair or embellishment - in other words, to '"let the music speak for itself"' (Taruskin, 1982, p. 339). Igor Stravinsky (1947) states that 'The sin against the spirit of the work always begins with a sin against its letter' (p. 124) and also distinguishes between two types of performer: the *executant* and the *interpreter*. An executant is described as the kind of performer who strictly obeys the letter of the score, whereas the interpreter is a performer who adds supplementary and (in the eyes of Stravinsky) unnecessary ornamentation against the direction of the score: 'the conflict[ions] of these two principles … impose themselves between the musical work and the listener and prevent a faithful transmission of its message.' (p. 122)

Over time, the widespread increase in availability of audio technology and the rise of post-structuralism has seen such ideas disappear, in large part, from music as a whole. Although written scores are still commonplace today within orchestral/big band concert situations etc., the movement away from a structuralist approach to music recital and consumption began with the recognition of the recording taking precedent over the score as the primary musical artefact; first through the compositions of Pierre Schaeffer and Pierre Henry, which contributed to the conception of Musique Concrète (Schaeffer, 1952/2012 [English translation]; Moorefield, 2005), and later in popular music; while the free improvisation of jazz also played a significant role in the shifting of this balance. Modern-day popular electronic music, through its approach to performing and listening to live music, is exemplary of the ideologies posited by post-structuralism with regard to openness in a work. Rather than playing the original recorded version of a piece during a performance 'DJs often favour remixes because they keep their playlists fresh … but still deliver a level of familiarity that they can be confident crowds will respond to' (This is the Remix, 2010, p. 25). In fact, prior to the emergence of "live" electronic music performance (as opposed to playing records/CDs/WAVs as part of a DJ set) made possible by technological advancements in both hardware and software, many producers would create one-off "VIP mixes" of their compositions and distribute the recording to a select few of their counterparts in order to provide audiences with yet another unique take on the original work among the numerous official remixes commissioned by the record label.

In recent years, the integration between Ableton *Live* software (2001-present) and button-matrix MIDI controllers such as the Akai *APC40* (2009), Novation *Launchpad* (2009) and DJ TechTools *Midi Fighter* series (2009-present) has paved the way for spontaneous, improvisatory live performances of recorded works to be fluidly and efficiently crafted through a process of breaking down a composition into its constituent parts and further dividing these up into individual loops/clips to be played back in any order and number of combinations - resulting in the ideal solution for artists looking to construct imaginative and bespoke live performances. The idea of the "VIP mix" has also evolved so that producers/DJs will now distribute the stems of their compositions to allow others to individualise and recontextualise a work in a manner unique to their particular performance-style. As has the process of composition whereby similar techniques to those implemented in performance are utilised to capture an essence of "liveness" in the recording. The acceptance of the live remix in popular electronic music culture has rewarded DJs with 'the status of artist, and this has necessitated a redefinition of such familiar concepts as musical instrument, performer and the role of audience in performance' (Fikentscher, 2000, p. 52). As such, the recent progression towards widespread interest and exploration on the part of the audience and composers/programmers respectively into interactive music systems (particularly games and album/single releases) within popular electronic music is both obvious and logical. (Meikle, 2016)

2.2 The beginnings of Computer-Aided Algorithmic Composition (CAAC) and Generative Music

Generative music, pioneered by the likes of Iannis Xenakis (1963/1971 [English translation]/1992 [revised ed.]), Lejaren Hiller and Leonard Isaacson (1958; 1959), David Caplin and Dietrich Prinz (Caplin, 2008, 2009a, 2009b, 2009c, 2009d, 2011, cited in Ariza, 2011; Prinz, n.d.) and Harriet Padberg (1964), has formed the foundation upon which the entirety of HCI in music has been built, and can be separated into two main categories. The first is 'aleatoric or "stochastic" music, in which events are generated according to the statistical characterization of a random process.' The second is 'music where the computer is employed to calculate permutations of a set of predetermined compositional elements' (Dodge & Jerse, 1997, p. 341). These random and deterministic processes can then, again, be divided into two further sub-categories. The former can be manipulated to result in either 'independent observations' from every calculation, or so that 'previous results influence the current outcome'. The latter encapsulates both *motivic* (based around the concept of applying independent variations to musical motifs/motives - 'the smallest melodic/rhythmic fragment of a musical theme - often only two or three notes') (p. 382) and *serial* (the application of transformative operations to a set of twelve tone-rows - each of which cannot contain any duplicate notes) (pp. 391-392) compositional techniques. Variation of the fundamental motifs and tone-rows of motivic and serial composition can be achieved in a number of ways. It is common in motivic works to make use of repetition, transposition, alteration, inversion, retrogression and "registral displacement" (displacement of pitch-values to another octave) with regard to melodic contouring, as well as augmentation and diminution of rhythmic features (pp. 383-388). "Classical" serial technique commonly involves the application of transposition, inversion, retrogression and retrograde inversion operations to the pitch-classes of the tone-rows (Babbitt, 1955; 2011, pp. 38-48), whereas more contemporary composers often implement the M5 and M7 operations, which '[reorder] the pitch-classes by multiplying each member of the row by 5, [or 7], modulo 12' (Dodge & Jerse, 1997, p. 395). Composers such as Olivier Messiaen, Pierre Boulez and Luigi Nono all experimented with rhythmic serialization through the use of 'durational rows … analogous to a series of pitch-classes', which 'can [then] be subjected to the same permutations as the pitch row' (p. 395; Smith-Brindle, 1966, pp. 163-167). On the other hand, Milton Babbitt and Charles Wuorinen used the *time-point system* (Babbitt, 1962; 2011, pp. 109-140; Wuorinen, 1979) ('the assignment of the pitch-class numbers to the corresponding rhythmic division of the measure' (Dodge & Jerse, 1997, p. 396)), and John Melby developed his own method for serializing rhythm (pp. 397-398).

Throughout his extensive catalogue of works Xenakis experimented with a number of different approaches to algorithmic composition. However, he maintained a disdain for the deterministic values of serialism and 'judged that the serial system had two weaknesses, the series and the polyphonic structure … he saw no reason to limit [the] number of tones to 12 or 13. "Why not a continuous spectrum of frequencies? … of timbres, intensities, durations?"' (Matossian, 1986, p. 85; Xenakis, 1955, cited in Xenakis, 1966, p. 11). Despite this, Xenakis' first major composition (excluding his so-called "early works" (Solomos, 2002)), *Metastaseis* (1953-54), does hold tenuous links to serialist practice through its use of twelve tone-rows; although these are not manipulated serially and - influenced by his plans for the design of the Couvent de La Tourette, which incorporated Le Corbusier's *Modular* (Matossian, 1986, p. 60), the basis of which was founded on the Golden Section and its inherent relationship with the Fibonacci series (Wheatley, 2007, p. 15) - Xenakis instead applied the Fibonacci series (a simple numerical series where the next number is equal to the sum of the two that preceded it) in articulating both pitch and rhythm (Matossian, 1986, p. 60). Harriet Padberg's work was also founded in serial technique, along with the use of text-to-pitch mapping procedures (1964, pp. 19-20) similar to the vowel-to-pitch mapping seen in Guido D'Arezzo's *Micrologus* (c.1026); although she fails to recognize this connection in her writing (Ariza, 2011, p. 48)[1]. Aside from *Metastaseis* - and a few more possible exceptions with deterministic properties including *Duel* (1959) and *Strategie* (1956-62); both of which exploit games theory through the provision of 'a set of musical tactics or moves [to the conductors of two opposing orchestras] that are graded according to the suitability of their choice in response to, or simultaneously with, the opponent's tactic' (Matossian, 1986, p. 139; Xenakis, 1992, pp. 110-130), and *Nomos Alpha* (1966) through its use of sieve theory (1992, pp. 219-236; see also pp. 194-200/1970, pp. 2-19 for a general analysis of sieve theory) - the vast majority of Xenakis' works are rooted in aleatoric/stochastic random processes. His second work, *Pithoprakta* (1955-56), for instance, makes use of probability theory (Matossian, 1986, p. 89) and is also influenced by the Maxwell-Boltzmann distribution (Xenakis, 1992, pp 15-21; Matossian, 1986, p. 92).

The first example of CAAC was Hiller and Isaacson's *Illiac Suite* string quartet (1956; 1959; Dodge & Jerse, 1997, p. 374; Ariza, 2011, p. 40); Caplin and Prinz were the first to use a computer to formulate new musical structures as opposed to purely generating sound, through their implementation of the *Musikalisches Würfelspiel* dice games attributed to Mozart in 1955 (Ariza, 2011, p. 40; p. 44); and Harriet Padberg was not only the first woman to have research and software related to CAAC published in 1964 but her dissertation, entitled *Computer-Composed Canon and Free Fugue* (1964), was the very first of its kind conducted by anyone (Ariza, 2011, p. 40; p. 47). Xenakis' first algorithmic composition using a computer (the IBM 7090) was entitled *ST/48-1.240162* (1956-62), while he also wrote a thesis in which he dissected his earlier work, *Achorripsis* (1956-57), which was originally calculated by hand, with the aim of outlining the fundamental rules of the process of composition to be applied to a computer program (Matossian, 1986, pp. 159-160). Subsequent works composed by Xenakis using a computer included *ST/10*, *ST/4*, *Morsima/Amorsima* and *Atrées* (1956-62).

[1] Coincidently, Padberg also shared Xenakis' interest in microtonality; for which she developed a computer system along with Max Matthews in 1968. (Hiller, 1970 & Padberg, 2008, cited in Ariza, 2011, pp. 47-48)

> Compared to the disappointing clumsily contrived computer compositions at the time and of later years, the value of Xenakis' contribution [to the field] should not be underestimated. With a simple 45-minute programme on the IBM 7090, he succeeded in producing not only eight compositions which stand up as integral works but also in leading the development of computer aided composition. (Matossian, 1986, p. 161)

Xenakis' thesis on *Achorripsis*, however, was not his first attempt to formalize an aspect of music-making, having previously done so in his book *Formalized Music: Thought and Mathematics in Composition* (1963/1971 [English translation]/1992 [revised ed.]). Here he used set theory as the basis to outline a skeleton-approach to algorithmic composition dependent on logic and group theory; of which *Herma* (1960-61) is the musical embodiment (pp. 170-177; Matossian, 1986, pp. 148-151). He was also responsible for conducting pioneering work within other areas of composition/performance that have come to directly influence the evolution of HCI in music, such as experimentation with spatialisation in *Eonta* (1963-64) (which also makes use of the Olympic system - a compositional technique, comparable in ways to complexity theory, where intricate musical phrases are "relayed" between multiple performers to make possible what is not in unison) (Boulez, 1964, cited in Matossian, 1986, p. 178), *Terretektorh* (1965-66) and *Nomos Gamma* (1967-68), as well as his application of "cinematographic painting" in creating a number of audiovisual works, or "polytopes"; examples of which include *Polytope of Montreal* (1967), *Hibiki-Hana-Ma* (1969-70), *Persepolis* (1971), *Polytope of Cluny* (1972) and *The Legend of Er* (1977) - composed specifically to be performed in the Diatope (a temporary, moveable pavilion/performance space designed by Xenakis as part of his architectural work and erected in Paris (Matossian, 1986, pp. 214-218; 224-227; Harley, 1998; Sterken, 2001; Oswalt, 2002).

In relation to the transformative/generative operations implemented into the system programming of *ScreenPlay* the various approaches to algorithmic composition used by Xenakis in the examples discussed above are of little direct influence; as is also true of other approaches not touched upon by Xenakis in the works mentioned, such as simulated annealing (Iazzetta & Kon, 1995), evolutionary computation and modelling/emulation of biological processes (Alsop, 1999; Brooks & Ross, 1996; Edwards, 2011; Miranda, 2003). His experimentation with Brownian movement/random walk in works such as *Synaphai* and *Persephassa* (1969) - which refined the rhythmic schemes developed in *Synaphai* (Matossian, 1986, pp. 230-233) - as well as *Gmeeorh* (1974), *Noomena* (1974), *N'Shima* (1975), *Khoaï* (1976) and *Erikthon* (1974) (Matossian, 1986, pp. 230-236) is, however, of a great deal more immediate relevance. Random walks and fractional noises are both Markovian processes (resulting in the formation of Markov chains), meaning that the value of the current outcome is dependent on those which have preceded it. First-order Markov chains - of which Caplin and Prinz's experimentation with transitional probabilities is reminiscent (Hiller, 1970, cited in Ariza, 2011, p. 46) - are not ordinarily overly suited for use within transformative/generative ICMSs, due to the fact that the outcome of each calculation is only subject to the influence of the result immediately prior to it - i.e. they have "short memories". Second and third-order Markov chains are much more appropriate, due to their longer "memories" and dependence on the previous two or three results respectively when generating new values. The fractional noises - including white noise, brown (or brownian)/1/*f*2 noise and 1/*f* noise - are all examples of random walks/Markovian processes. White noise has no "memory", brown noise has an extremely short "memory", while '1/*f* noise has the best memory of any noise … and produces patterns said to be "self-similar"' (Dodge & Jerse, 1997, p. 369). As such, 'One-over-f (1/*f*) noise is generally agreed to produce the most aesthetically pleasing quality [when applied to music] of the three types of noise' (p. 370). 1/*f*0.5 and 1/*f*1.5 also have a reasonably long memory and so too are worthy of consideration in this context (p. 369). *ScreenPlay* makes use of both second-order and first-order Markov chains - despite their short "memory" - applied to pitch values and other musical parameters respectively (velocity, duration etc.) in order to strike a pleasing balance between recognisability and variation and unpredictability.

As well as Xenakis' use of random walks in his compositional process for a number of works, his stochastic approach to sound design is also apt in relation to *ScreenPlay*; with its use of a topic-theory-inspired transformative algorithm to alter not only pitch-values and rhythmic durations but also textural/timbral sonic properties (Xenakis, 1992, p. 246; Serra, 1993; Harley 2002). As is his general outlook on the use of mathematics in music influential to HCI in popular electronic music as a whole, being that he always showed a willingness to alter the results of his calculations in the case he felt doing so would improve the artistic quality/integrity of a piece (Matossian, 1986, p. 156). In fact, Xenakis was so ahead of his time that, in many ways, he epitomises the computer-musician of today: his multi-faceted skill-set (comprising architecture, mathematics, composition/arrangement and computer programming) equivalent to the array of talents required of the modern-day electronic musician (also inclusive of composition/arrangement and computer programming - albeit of a slightly different kind - as well as sound-design and production skills, such as mixing and mastering).

It should be remembered that of equal importance to the work of, among others, the individuals mentioned above in the progression of algorithmic/generative music are the technological advancements in computer hardware that facilitated it; such as the integrated loudspeakers of the Ferranti Mark I (1951), Mark I* (1951-57), Mercury (1957) and CSIRAC (1949-64; CSIRAC, n.d.) used by Caplin and Prinz (Prinz, n.d.; Ariza, 2011, p. 46), the IBM 1620 (1959-70; 1620 Data Processing System, n.d.; da Cruz, 2009) and 7072 (1960) used by Padberg (Padberg, 1964, cited in Ariza, 2011, p. 48), and the 7090 (1959; 7090 Data Processing System, n.d.) used by Xenakis (Xenakis, 1992; Matossian, 1986, pp. 157-162). It is by using these computers that they - along with many others between 1955 and '65, such as Hiller and Robert Baker with their Music-Simulator-Interpreter for Compositional Procedures (MUSICOMP) (Baker, 1963, cited in Ariza, 2011, p. 41; Hiller, 1967; 1969) and Gottfried Michael Koenig's *Project 1* (1970) - were able to '[implement] and extend well-known approaches to procedural algorithmic composition that had formerly been carried out without computers' (Ariza, 2011, p. 41). The same pattern of development can be seen reflected in the evolution of HCI in music nowadays, with the rapid developments in ICMS design being driven not only by the individuals programming the systems but also by increasing technological advancements in terms of improved hardware and its affordances, as well as the mass availability and affordability of sensory/touchscreen technology and other technologies associated with the field.

Xenakis himself even developed in the 1970's an early ICMS of sorts in the form of UPIC (Unité Polyagogique Informatique CEMAMu) (Xenakis & Saint-Jean, 1977; Xenakis, 1992, pp. 329-334; Matossian, 1986, pp. 240-242; Marino, Serra & Raczinski, 1993; Di Nunzio, 2014), which was a unique compositional tool/instrument in the form of a large interactive table/tablet-like surface onto which waveforms and compositions could be drawn using an electromagnetic pen. The piece *Mycenae Alpha* (1978; Matossian, 1986, p. 242) was Xenakis' first using the system. Development of UPIC continued long after its initial introduction in 1977, with significant iterative releases coming in 1983 with updated hardware, the first real time version in 1988, and the first version for PC in 1991. The implementation of screen-based interfacing and, in particular, touchscreen devices such as smartphones and tablets by many of the ICMSs we see today - not least *ScreenPlay* - goes to show how UPIC demonstrated at an incredibly early stage the potentialities of the use of this technology in HCI in music.

2.3 Early and Experimental Interactive Works and Systems

Long before the emergence of ICMSs in popular electronic music culture the potentialities of HCI in music were also being explored by pioneering individuals like Michel Waisvisz. *The Hands* (1984; 2006) was a sensory-based interactive system responsive to non-musical gestures transmitted via wearable controllers attached to the fingers and hands of the user, and thus showcased a loose basis for the design of ICMSs to be used by non-expert musicians - a key notion in the design and functionality of *ScreenPlay*. Laetitia Sonami's *Lady's Glove* (1991-2001; n.d.), which was developed and manufactured in conjunction with Bert Bongers in its fourth and fifth generations, is conceptually similar to *The Hands* - utilising hand/wrist-mounted controllers to capture gestural, non-musical input data used to influence the musical output of the system - yet in some ways is more refined in terms of operational simplicity and efficiency; especially the later incarnations with the added influence upon design of Bongers. Despite the dependence of these systems on non-musical control-gestures, which have been inherent in the evolution of ICMSs aimed at non-expert users, it can be assumed that their fluid and efficient operation would be subject to somewhat of a learning curve due to the fact they were both designed to be used in performance only by their respective creators. The work and performances of electronic ensemble *HyperSense Complex* (Riddell, Langley & Burton, 2002-2005; Riddell, 2005; n.d.; Langley, n.d.; Burton, 2003) also exemplify sensory/gestural musical interaction for trained/professional performers and musicians utilising wearable motion-sensitive controllers.

Another electronic ensemble, *The Hub* (Bischoff, Perkins, Brown, Gresham-Lancaster, Trayle & Stone, 1987-present; Early Computer Network Ensembles, n.d.; Brown, n.d.), is responsible for advancing the field of "Computer Network Music" - a genre of electronic/electroacoustic music that explores the potentialities of enabling multiple performers to collaborate and interact with each other in improvisational composition and performance through a shared, connective ICMS via individual user-interfaces (UIs) or instruments. In this instance *The HUB* itself acts as the central computer juncture through which each of the participating members can connect via a wired local network. *Global String* (Tanaka & Toeplitz, 1998-2001; Tanaka, n.d.; Plohman, 2000) is another example of an ICMS representative of Computer Network Music. In contrast to *The HUB*, however, the system was designed not only for use by a specific ensemble of musicians involved in its development but also as a gallery installation for public use. In developing *Global String* Tanaka and Toeplitz expanded upon the ideas established by *The Hub* by using the internet as the central juncture through which a number of remote users at multiple sites around the world could interact with the users in the other installation performance-spaces. The installation consisted of a physical string connected to a virtual string network, which would transfer analogue pulses of the real string measured by vibration sensors and converted to digital data over the virtual string network to the other installation locations. The responses performed by these users would be transferred back over the virtual string network and provide audible and visual feedback directly to the other performance spaces. The concept of Computer Network Music is key to *ScreenPlay*, but its implementation falls somewhere in between that of *The HUB* and *Global String* to produce an experience when interacting and "playing" with the system that is accessible to and enjoyable for both novice musicians/users and those with a high level of experience, proficiency and expertise.

Further examples of interactive sound-art installations designed for public gallery spaces include *Gestation* by Garth Paine (1999-2001; 2013) and *Bystander* by Ross Gibson and Kate Richards (2004-2006; n.d.) - part of the larger project *Life After Wartime*. Both *Gestation* and *Bystander* interpret and respond accordingly to the number and movements of individuals within the multi-channel installation environment. *Bystander*, however, incorporates the use of an additional parameter by responding dynamically to the collective attentiveness of the audience. By collecting this data and applying it as an influential factor in the generation and evolution of the audiovisual output from the system the audience can systematically learn during the process of experiencing and interacting with the system how to impart a certain level of control over the resulting output.

> The central premise of *Bystander* is that the more quiet and attentive the audience, the more aesthetically coherent and semantically divulgent the "world". Ideally visitors can gain the "trust" of the space and perform a dance of intimacy with the "world" and its complex narrative matrix. (Gibson & Richards, n.d., p. 1)

Although both systems allow for the generation of musical output without assuming or requiring any prior knowledge of the operational protocols on behalf of the participants, they offer little or no direct and precise control over the subsequent soundscape to any one individual within the collective group whose movement through the various sections of the installation space over time is expressed in the composition generated by the algorithms encoded into the system architecture. When coupling this with the lack of feedback provided to the audience defining to some extent how the musical output of the system is influenced by the input data - especially in *Gestation* - it could be construed that both systems are more randomly *reactive* than *interactive*. This is also true of *La Maison Sensible*, or *The Sensitive House* (Lasserre, met den Ancxt, Ajima & Nagemi, 2015; Lasserre & met den Ancxt, 2015; Emory, 2015), which is another audiovisual gallery installation; although it is instead designed to respond to the physical interactions between the audience/users and the walls/objects within the performance environment. Because of the lack of genuine, communicative two-way interaction between the users and these systems their design has little bearing on that of *ScreenPlay*, which aims to supply a deeply engaging experience to each of the users interacting with it at any one time.

This is not, however, the case with the works of schnellebuntebilder and Universal Everything - in particular *STEPSEQUENCER* (Timpernagel et al., 2013-14; STEPSEQUENCER, n.d.; synthhead, 2014a) and *1000 Hands* (Pyke et al., 2013; Kaganskiy, 2013). *STEPSEQUENCER* is an audiovisual installation designed to be controlled simultaneously by a small number of participants via both physical and digitally-projected control-objects located on the floor of the performance space in a manner reminiscent to that of the classic arcade game *Dance Dance Revolution*, or *Dancing Stage* (Konami, 1998-present; DDR, n.d.). It is the precise control afforded to the users over specific aspects of the musical output by the various control objects and the focus of the system on generating popular electronic/dance music in response to the actions of the user that is influential in the approach to designing *ScreenPlay*. The majority of the installations produced by Universal Everything focus primarily on generating visual responses to the interactions of the audience as opposed to audible ones. Despite this, the reliance of *1000 Hands* on the use of the audience's own touchscreen devices to interface with the system by drawing images and shapes that are then transmitted to and shown on the display of the installation is paramount to the design of *ScreenPlay*, which is largely focussed on exploiting the possibilities of screen-based interfacing. All of the interactive sound-art installations described above -

particularly *La Maison Sensible*, *STEPSEQUENCER* and *1000 Hands* - are examples of what Steve Benford describes as "mixture reality performance" (Benford, Giannachi, Boriana & Rodden, 2009; Benford, 2010, p. 57), which 'combine[s] physical and virtual spaces in various ways to create mixed-reality stages and that also combine elements of live performance with interaction with digital technologies.'

There are numerous other approaches to ICMS design that have been deemed inappropriate in the development of *ScreenPlay*. *Inter-Harmonium* (Miranda, n.d.a; Miranda & Brouse, 2005a), *BCMI-Piano* (Miranda, n.d.b; Mirada & Brouse, 2005a; 2005b) and *Eunoia* (Park, 2013; Park, n.d.; synthhead, 2014b; Chow, 2013) are all exemplary of brain-computer interfacing (BCI) systems and are reliant upon a technique known as Electroencephalography, or EEG, which is used to measure brain-patterns as voltage fluctuations by attaching sensors to the scalp.

> EEG is a difficult signal to handle because it is filtered by the meninges, the skull and the scalp before it reaches the electrodes. Furthermore, the signals arriving at the electrodes are sums of signals arising from many possible sources, including artefacts like the heartbeat, muscle contractions and eye blinks. (Miranda & Brouse, 2005b, p. 2)

As a result, the data collected by the EEG can only be used to modify the sound generated by these systems in a very general and flexible way. It is because of this that the technique in neither suitable nor applicable to the *ScreenPlay* system design-model, which is intended to provide multiple users with full and precise control over a particular sound-source within the arrangement of the musical output from the system; or a single user control over numerous parts/instruments.

Pieces such as *Maritime* by Robert Rowe (1992; 1999; Drummond, 2009), *Voyager* by George Lewis (1993; 2000; Drummond, 2009), *Music for Clarinet and ISPW* by Cort Lippe (1991; 1993) and *Pluton* by Philippe Manoury (1988; Puckette & Lippe, 1992) are examples of *score-driven* ICMSs, or "score-followers". These type of systems are responsive solely to musical input from acoustic instruments and are often specifically created for the performance of a particular composition. For a modern-day interactive system to be designed in this way is very uncommon as most are tailored towards facilitating the composition and performance of electronic music in fun, exciting and unique ways that are able to captivate the creativity and imagination of both novices and experts alike. In a similar vein, and one which epitomises the changing approach to system design over time from "score-followers" to the systems and games we encounter today, the *HyperInstruments* project (Machover, 1986-present; Machover & Chung, 1989; Hyperinstruments, n.d.) originally focussed on the expansion of traditional acoustic instruments to allow for an extension of range in performance techniques and possibilities for professional musicians. Since 1992, however, much attention has been devoted to the development of sophisticated interactive music systems for non-expert musicians, such as *Drum-Boy* (Machover, n.d.a) and *Joystick Music* (Machover, n.d.b). (Meikle, 2016)

2.4 Web and Touchscreen-based Interactive Music Applications and Games

The internet and, in particular, the ever-growing presence of touchscreen devices, such as smartphones and tablets, has paved the way laid out by the above systems for the expansion of musical interaction into mainstream popular culture from the relatively niche area of experimental/academic art and research. *Symphone* (Lindetorp, n.d.) is a simple web-based ICMS that allows participants to use their smartphones as a sound-source to build up an orchestral piece of music comprised of pre-recorded clips for each instrument within the orchestra. Users are randomly assigned an instrument upon connecting with the system, while any unassigned instruments are played back via the central computer within the system. Whether or not a particular instrument/part is playing is left to the discretion of the user to whom it has been allocated. This is the only control over the output of the system provided to the participants and, as such, the functionality of *Symphone* is fairly basic in comparison to the majority of other web- and screen-based systems.

NodeBeat (Sandler, Windle & Muller, 2011-present), *Kinetic* (humbleTUNE, 2011-present) and *Bloom* (Eno & Chilvers, 2008) are all relatively similar ICMS applications for Android/iOS based around gravity mechanics/physics modelling and generative algorithms. Both *NodeBeat* and *Kinetic* are fundamentally dependent upon the principle of motion, with the former generating musical output from the interactions between moving "nodes" and "generators", which float around the display and form temporary connections when coming within close proximity of one another, and the latter generating sound as a result of node-/ball-like objects impacting and bouncing off the four sides of the screen. In *NodeBeat*, the user is able to influence the generation of musical output in a number of ways, including: key signature/scale and lowest octave, oscillator wave-shape and ADSR envelope shape, reverb/delay level; the number and variable velocity and connection-proximity of nodes/generators as well as disabling movement altogether for either nodes and/or generators and enabling "gravity", which uses the accelerometer within the smartphone or tablet to enable users to manually influence the directional movement of the nodes/generators; tempo and quantization value. The background of the display is also playable as a key-signature-locked keyboard. *Kinetic* offers a similar but less in-depth level of control, while the GUI background of *Bloom* is also playable as a keyboard but the generation of musical output if subject entirely to evolutionary algorithms. The simple mode of interaction, engaging GUI/animations and musical constraints of these systems are attributes that lend themselves well to supporting intuitive interaction for non-expert users; however, the drawback of imposing these limitations in order to achieve this goal is that systems such as these struggle to captivate more experienced users/musicians beyond the point of initial intrigue. *Svärm* (humbleTUNE, 2013-present) and *Pixel Tune* (humbleTUNE, 2010-present) are two other systems with the same target audience that operate in a slightly different way. *Svärm* is designed only to generate animated visuals in response to any music inputted into the system by the user, while *Pixel Tune* uses generative algorithms to respond musically to the visual input drawn into the interface by the user.

Two more web-based ICMSs aimed at novice musicians, which are more closely related to *Symphone* due to their reliance on sample/loop playback as opposed to algorithmic generation, are *Incredibox* (So Far So Good, 2011-present) and *Patatap* (Brandel, 2012-present; Brandel, 2015). These systems also differ from any of the other web/screen-based ICMSs previously described due to the fact that the generated musical output is much more akin to popular electronic music than the orchestral music of *Symphone* and the Ambient music of *NodeBeat*, *Kinetic* and *Bloom*. At the time of writing, *Incredibox* exists in four iterations comprised of different loop-based material, but the user interacts with all of them in exactly the same way: by choosing between multiple cartoon characters - each of which has assigned to it a particular loop - in order to create a lineup of characters who appear to "sing" the instrumental sounds/effects and lyrics of the arrangement as it plays. *Patatap*, on the other hand, requires users to trigger short samples using their computer keyboard - different banks of which can be accessed by pressing the spacebar.

Finally in this category, there are a large amount of interactive music games available for iOS/Android touchscreen devices which draw influence from the classic Xbox game series that includes titles such as *Guitar Hero* (Harmonix, Neversoft, Budcat Creations, Vicarious Visions & FreeStyleGames, 2005-present), *Rock Band* (Harmonix & MTV Games, 2007-present) and *DJ Hero* (FreeStyleGames & Exient Entertainment, 2009). Specifically, *Cytus* (Rayark Inc., 2012-present) and *Dynamix* (C4Cat Entertainment, 2014-present) are almost carbon-copies of these games, despite the musical content being far more electronically-/dance-oriented. *Deemo* (Rayark Inc., 2013-present), *Beat Beat Volcaloid* (Kestrel Games Studio, 2013-present) and *Full of Music* (Handicrafter, n.d.-present) all also operate in the same manner, although the first two bring a story-based structure to the format and the latter allows users to play along to their own music collection. The global appeal of these games over the past decade demonstrates more clearly than anything else the potential within mainstream popular culture for HCI in music to thrive. (Meikle, 2016)

Other than the mode of interaction of the the touchscreen-based systems discussed here, there is little of direct influence upon the design of *ScreenPlay*; save for the loop-based approach to musical interaction exhibited in *Incredibox*, which is expanded upon within *ScreenPlay* by allowing users to initially record their own material into a fixed number of loops before/whilst using them to construct the arrangement of the composition/performance as opposed to providing them with a fixed set of material with which to work.

2.5 HCI in Popular Electronic Music Records/Releases and Music-Creation Applications/Virtual Instruments

Following on from the simple systems and games described above, there is another class of interactive applications and games offering a far greater level of musical complexity and precision. The most prominent of these is *Reactable Mobile* (Jordà, Kaltenbrunner, Geiger, & Alonso, 2003-present) and the recently released *ROTOR* (Jordà, Kaltenbrunner, Geiger, & Alonso, 2016), which is an evolution of *Reactable Mobile* from the same company that introduces physical potentiometers for improved control, some additional features/functionality to the software, and a streamlined GUI (ROTOR, n.d.). First conceived as a touchscreen-based tabletop hardware instrument before being developed as an application for Android and iOS devices, the *Reactable* systems are better defined as digital modular synthesizers than ICMSs due to a lack of two-way communication functionality between the user and computer in terms of the computer contributing in conjunction with the user to the generation of musical material. Because of this, *Reactable*, *Reactable Mobile* and *ROTOR* are primarily aimed at experienced electronic musicians and, as a result, the ability to interact with the system to the full extent of its possibilities is subject to one's knowledge and experience-level in relation to the GUI; which requires users to form connections between different objects or "crystals" - each with a specific function (oscillator, filter, sequencer etc.) - in order to generate sound. This is, however, something that Reactable have recently moved to address through the introduction of a new tabletop hardware instrument, the *Reactable Experience*, developed for implementation in museum and gallery installations as well as other public spaces such as hotel lobbies etc. (The New Reactable Experience, n.d.). Despite providing the possibility to compose, perform and record entire musical works, the *Reactable* systems still have numerous deficiencies in terms of ICMS design that are accounted for in the design of *ScreenPlay*.

Similarly, *Audulus* (Holliday, 2011-present; Subatomic Software Audulus, 2014), *Jasuto* (Wolfe, 2008-present) and *Nanoloop* (Wittchow, 1998-present) are all systems designed with music creation in mind but that do not incorporate the use of any two-way communicative capabilities between user and computer. *Audulus* and *Jasuto*, like *Reactable*, are both modular in their design; allowing users to connect different sound-source/effect objects etc. together to create virtual instruments and the like. *Jasuto* is focussed entirely on synthesis, while *Audulus* offers greater potential for experimentation not only with sound-design but also control of external instruments, MIDI devices etc. and is more like a stripped-down, simplified version of the modular programming environments Max/MSP (Puckette, 1988-present) and Pure Data (Puckette, 1996-present). Both of these systems aim to provide intermediate-level electronic musicians with an introductory route into the areas of modular sound-design and visual programming respectively - in particular with *Audulus*, which allows for novice programmers with little or no experience to explore the creative potentialities of working within a modular programming environment and incorporating this into their music-making process without the need to undergo the extensive learning-curve required to gain a relative level of knowledge and proficiency with regard to the visual programming languages used in Max/MSP and Pure Data.

Nanoloop (Wittchow, 1998-present), on the other hand, is a modernised version of the Game Boy/Game Boy Advance game *Nanoloop 1*/*Nanoloop 2* ported to Android and iOS, which enables users to build up short, loop-based compositions using a simple FM synthesizer, sequencer and sampler; while there are also a multitude of examples of music-creation apps and games designed for up-to-date handheld consoles and Android/iOS devices with more experienced electronic musicians in mind. Included in this category are: KORG's *DS-10* (Cavia Inc., Sano & Mitsuda, 2008), *DSN-12* (KORG Inc., 2014; synthhead, 2014c) and *M01D* (KORG Inc., 2013) for Nintendo DS/3DS, *iMS-20* (KORG Inc., 2010-present), *iM1* (KORG Inc., 2015; Rogerson, 2015; synthhead, 2015) and *iElectribe* (KORG Inc., 2010; Korg iElectribe, 2010) for iOS, and Arturia's *iProphet* (2014; Arturia iProphet, 2014; synthhead 2014d), which are fully functional virtual-analogue/digital emulations of their classic hardware counterparts; an abundance of traditional virtual-analogue/digital synthesizers from small, independent developers such as *Heat Synthesizer* (Schneider, 2013-present) and *FM Synthesizer*/*SynprezFM II* (Desprez, n.d.-present) for Android; and experimental/forward-thinking synthesizers, which aim to take advantage of the potentialities afforded by touchscreen-based interfacing, like *Arpio* (Randon, 2014-present), *Ether Surface* (Batchelor, 2014), *Ethereal Dialpad* (Smith, 2011) and *Photophore* (Dika, 2014-present; synthhead, 2014e), also for Android and iOS.

Again, none of these systems can be described as "interactive" in terms of facilitating communicative collaboration in composition and performance between human and computer; there are, however, a number of systems that aim to achieve this in interesting and engaging ways. *FRACT OSC* (Flanagan, Nguyen & Boom, 2011-present) - for Windows/Mac - and *PolyFauna* (Yorke, Godrich. & Donwood, 2014) - for Android/iOS - are both examples of open-world musical exploration games whereby, as you move through and interact with the virtual environment, the generated musical output evolves in accordance with your actions. *FRACT OSC* also includes elements of traditional synthesis and problem solving that enhance the overall level of immersion when experiencing the game. *Synthesizer 7DRL* (TheBroomInstitute, 2015; Hybrid RPG and Synthesizer, 2015) is another web-based ICMS in the form of a complex role-playing game (RPG) based around the fundamental concepts of subtractive synthesis.

In addition, there are also various other ways in which HCI manifests within mobile applications and games; both in the context of music and outside of it. For example, *80 Days* (Inkle Studios Ltd., 2014) is an interactive novel-based game, while *Navichord* (Kutuzov, 2014; synthhead, 2014f) is an educational tool for learning the fundamentals of music theory and harmony.

The increased prevalence in recent years of touchscreen technology in everyday life has, of course, also contributed to a sharp rise in the development of applications to be used in tandem with, or even take the place of, professional audio software and outboard gear such as

DAWs (digital audio workstations), hardware synthesizers and MIDI controllers as an integral part of the composition, production and performance of electronic music. Novation have released an iOS version of their *Launchpad* MIDI controller (2009), *Novation Launchpad* (Focusrite Audio, 2013), while plenty of third-party developers have released their own imitations of the hardware/software, such as *Launch Buttons* (Nowak, 2015) for Android. Even more developers, however, have exploited the advantages software holds over hardware in order to improve upon the original concept of the hardware button-matrix MIDI controller by allowing for users to create entirely unique and fully customizable control-surface layouts from scratch. Such applications include *TouchOSC* (Fischer, 2008-present) (the app used to design and host the GUI for *ScreenPlay*), *LIVKONTROL* (Imaginado, 2011-present; synthhead, 2014g), *touchAble* (Blomert, C., Garcia, Keppmann, Blomert, P. & Kapp, 2010-present) and *Lemur* (Slater et al., 2011-present) - a software iteration of JazzMutant's famous *Lemur* (Largillier, Joguet & Olivier, 2007) MIDI/OSC multi-touch hardware controller that, along with the discontinuation of the hardware version, serves perfectly to exemplify the changing music production/performance market and thus the development priorities of pro-audio companies. As well as MIDI/OSC controllers, there are a number of iOS and Android applications that aim to negate entirely the need to work within a DAW when producing electronic music. Included in this category are Akai Professional's emulated version of the famous *MPC* series (1988-present) of hardware samplers, *iMPC* (2012-present), and Native Instruments' emulation of their *Maschine* (2009-present) range of grooveboxes, *iMaschine* (2011-present). There are also more DAW-like examples; namely Image-Line's *FL Studio Mobile* (2011-present), KORG *Gadget* (2014-present; Aisher, 2014; Nagle, 2014), *Caustic 3* (Single Cell Software, 2013-present) and *G-Stomper Studio* (Planet-H, 2013-present). Although the majority of these applications are not so much "interactive", they all promote music-making for non-experts in some capacity. Be it through their design and intended functionality or their market-placement in terms of price-range when compared to that of professional audio hardware and software, as well as only requiring the use of technology most novice musicians will already possess (as opposed to potentially expensive, specialised computer equipment) - a smartphone or tablet - applications like these make taking the first steps as an electronic musician accessible to anyone.

Likewise, the influence professional audio hardware and software has had on the development of mobile music-making solutions is reflected in that of applications aimed at providing an introduction to electronic music production for beginners in the design of more recent professional-level hardware design. For instance, the Ableton *Push* (2013), Novation *Launchpad Pro* (2015) and Native Instruments *Komplete Kontrol S-Series* MIDI keyboards (2014; Griffiths, 2014) all utilise intelligent backlighting of pads/keys to denote the notes and root-notes within a chosen key signature/scale, while the Ableton *Push* even allows for the entire button-matrix grid to be "locked" in key (a design concept elemental to the *ScreenPlay* GUI), meaning chromatic notes are not available unless chosen by the user and standard triads within the scale can be formed using the same hand-shape anywhere on the control-surface.

One of the more significant developments in terms of HCI in popular electronic music in particular has been the emergence of official music releases from established artists being packaged as interactive applications and games as opposed to the traditional recorded format. Possibly the most important and well-publicised of these is *Biophilia* by Björk (2011), who has not only provided a platform for interactive performance techniques in more popular veins of electronic music through extensive performances utilising the *Reactable* modular synthesizer, but has also been a pioneering figure in composing and designing ICMSs for screen-based interfaces aimed at providing non-expert musicians with the freedom to actively engage in creating and influencing musical pieces as they listen to them through her work on *Biophilia*. Each composition embodies its own unique interactional model and interface; many of which rely on generative algorithms inspired by biological and physical processes found in nature to provide appropriate responses to the input of the user. Lady Gaga also released the album *ARTPOP* (2013) as an interactive application for Android and iOS, Skrillex has created an audiovisual interactive website for the single *Doompy Poomp* (2014; Division Paris, Skrillex & Creators Project, 2014) - enabling users to "remix" the music video using the keys on their computer keyboard to trigger short GIFs as the track plays - and there are numerous websites such as *DaftPunKonsole* (Dellidj, n.d.) and *iDaft* (Najle, 2010-present) that implement the same technique to enable users to re-imagine popular Dance songs - in this case *Harder, Better, Faster, Stronger* (Daft Punk, 2001) and *Technologic* (Daft Punk, 2005) - by triggering loops and one-shot samples over the top of an underlying groove. In addition, the concept has transcended to more "underground" electronic music genres, as evidenced by the Teengirl Fantasy EP *THERMAL* (2014; Teengirl Fantasy & 4real, 2014; DJ Pangburn, 2014) - a web-based application enabling users to enter and interact with in different ways a unique virtual world for each song on the EP (four in total) - and the *Sword and Sworcery* EP by Superbrothers (Superbrothers, Jim Guthrie & Capybara Games, 2011-present) - 'an exploratory action adventure [game] with an emphasis on audiovisual style' (Superbrothers

Sword & Sworcery, n.d.), much like *FRACT OSC* (Flanagan, Nguyen & Boom, 2011-present) and *PolyFauna*. (Yorke, Godrich. & Donwood, 2014)

There are also examples, such as *Reactable Gui Boratto* (Boratto, Jordà, Kaltenbrunner, Geiger & Alonso, 2012) and *Reactable Oliver Huntemann* (Huntemann, Jordà, Kaltenbrunner, Geiger & Alonso, 2012), of established dance music producers and DJs releasing select compositions for use with pre-existent music-creation systems, whereby users are able to simply watch back the recorded performances of the artists themselves or can interact with them on whatever level they choose. This could be anything from the evolution and development of the form and structure, sounds and effects, or the addition of newly synthesized melodic, harmonic and rhythmic material, to a complete reassembly of the constituent parts with the addition of new parts and lines in order to create an entirely unique reinterpretation of the piece. The same is true of sound packs released by artists such as Mad Zach for use with Ableton *Live* (2001-present) and *Traktor* (Native Instruments, 2000-present) along with specific MIDI controllers like the *Midi Fighter* (DJ TechTools, 2009-present) or Ableton *Push* (2013) and, more recently, the introduction of Native Instruments' new audio format *Stems* (Ramley, 2015), which allows musicians to purchase music in the form of its individual constituent parts/stems (drums, percussion, bass line, vocals etc.) with the aim of providing amateur remixers/bootleggers with higher quality materials and thus promoting ingenuity and creativity in the popular electronic music scene.

Aside from displaying yet another evolutionary step forward from the VIP mix, as discussed in the first section of this chapter, the release of such material effectively transforms previously *reactive* music-making software into an ICMS through the two-way collaboration that is made possible between the user and the composer. Coincidently, it also addresses the issue pointed out by Roland Barthes in his paper *Musica Practia* when discussing his theory of 'two musics … the music one listens to [and] the music one plays' (1977, p. 149). Barthes argues that 'passive, receptive music, sound music, [has] become *the* music (that of concert, festival, record, radio): playing has ceased to exist' and that, in contrast to the popular music of the late nineteenth and early twentieth centuries - an era made famous by the music publishers of Tin Pan Alley in New York City - when the written score, rather than the recording, acted as the primary musical artefact and simplified piano/vocal lead-sheets were the common unit of sale, 'The amateur, a role defined much more by a style than by a technical imperfection, is no longer anywhere to be found' (p. 150). Not only do interactive releases encourage novice musicians to engage in actively exploring their creativity and musicality, they also go some way to replacing some of the "special" qualities (aesthetics, artwork etc.) that are missing from the sterile experience of purchasing music from digital downloads stores. (Meikle, 2016)

2.6 Categorisation and Approach to Design of ICMSs

> Interactive systems enable compositional structures to be realised through performance and improvisation, with the composition encoded in the system as processes and algorithms, mappings and synthesis routines.
>
> The challenge facing designers of interactive instruments and sound installations is to create convincing mapping metaphors, balancing responsiveness, control and repeatability with variability, complexity and the serendipitous. (Drummond, 2009, p. 132)

Every one of the systems outlined above can be categorised as one of either *sequenced*, *transformative* or *generative*. They will all also communicate and interact with the user/performer as would either an *instrument* or a *player* - i.e. responding to the input data with a sense of reactive predictably or independent autonomy respectively (Rowe, 1994). *Sequenced* systems (*Incredibox* (So Far So Good, 2011-present), *Patatap* (Brandel, 2012-present; Brandel, 2015)) are usually tailored towards a lone user and allow for the full orchestration and arrangement of system-specific or pre-existing compositions; the constituent parts of which are broken up into short phrases and clips. Interactive systems based on this design model are often devoid of any form of influential response from the computer in relation to the input from the user over the resulting musical output. *Transformative* and *generative* systems are ordinarily reliant upon an underlying algorithmic framework to generate an appropriate musical response to the input information from the user and are more suited than sequenced-response systems to facilitating multiple-user input. ICMSs based on the *transformative* or *generative* design models (*NodeBeat* (Sandler, Windle & Muller, 2011-present), *Bloom* (Eno & Chilvers, 2008)) are often melodically and harmonically simplistic, incorporating only a few different parts/lines, and offer limited influence to the user(s) over the musical output; while all of *transformative*, *generative* and *sequenced* response-type system designs are hampered by considerable stylistic constraints - i.e. it is possible to program/compose an ICMS for practically any genre of music, but it is very rare for these systems to provide the user(s) the opportunity to control or alter this to their personal taste. By incorporating aspects of all three categories into its design *ScreenPlay* aims to address this issue as well as the specific deficiencies of each individual category. Additionally, Rowe also distinguishes between *score-driven* and *performance-driven* systems - although contextually this is relevant only to *transformative* interactive systems responsive to musical input from a traditional or modified acoustic instrument (*Maritime* (Rowe, 1992), *Voyager* (Lewis, 1993) etc.) and therefore not the development of *ScreenPlay*, as most modern ICMSs aimed at non-expert users and musicians are always *performance-driven*.

One theory relating to the design of ICMSs paramount to that of *ScreenPlay* is Tina Blaine and Sidney Fels' 11 criteria for collaborative musical interface design (2003). The 11 criteria are: focus, location, media, scalability, player interaction, musical range, physical interface/sensor, directed interaction, learning curve, pathway to expert performance and level of physicality between players. A full and detailed definition for each criterion and how they have been incorporated into the design of *ScreenPlay* is provided in the Methodology and Implementation chapter of the thesis.

Closely related to Blaine and Fels' 11 criteria for collaborative musical interface design, and in particular that of *focus* and "audience transparency" (2003, p. 414; Fels, Gadd & Mulder, 2002) - the ease with which members of the audience are able to discern the connection between the actions of the user(s)/performer(s) when interacting with the system and the resulting musical output - is Steve Benford's approach to the classification of 'high level design strategies for spectator interfaces' (2010, p. 55) as *magical*, *expressive*, *secretive* and *suspenseful*. Often the most useful of these design strategies within the context of HCI in music is *suspenseful*, which 'involves revealing manipulations to the spectator while hiding their effects'; meaning that prospective users 'waiting in line to experience an installation … can benefit from seeing in advance how others use it … but without seeing the "payoff" until it is actually their turn' (p. 57). The *expressive* strategy, which 'aims to make both manipulations and effects as visible as possible' (p. 56), can also be useful when included in the UI design of an ICMS despite most commonly being attributed to the mode of interaction of traditional instruments like the piano or guitar. *Secretive* tactics stand in direct opposition to *expressive* by obscuring from view of the audience both the gestural manipulations and their resultant effects upon the system. As such, they have little relevance to HCI in music and are more suited to the design of everyday computer systems such as ATM machines and Chip and

Pin card readers. The final design strategy proposed by Benford is *magical*, which 'involves revealing effects but hiding away the manipulations' (p. 56). The implications of which bear relevance to live musical performance by '[enabling] a performer to hide clumsy interactions that might detract from the overall aesthetic of the performance … [like] "meta-level" control of instrument settings, rather than the immediate production of musical sound, often accomplished by foot-switches, pedals, and similar devices'. The *magical* strategy can also relate to the Blaine and Fels' *media* criterion, which relates to the use of audiovisual elements within an interactive system, when the intention is to distract users through the use of visuals and animation from the actions of others in influencing the generated musical output of the system. Of the four approaches outlined above the most applicable to the design of *ScreenPlay*'s GUI is *expressive*; due to the resemblance between the way in which users interface with the system and that of traditional acoustic instruments such as the piano, as well as the visual representation of control objects that are traditionally associated with electronic music production.

Linked directly to these design strategies for addressing the audience is what Benford describes as the "framing" of the system (Benford et al., 2006; Benford, 2010, p. 56) - which 'makes a distinction between the audience who are within the performance frame … and are able to interpret the action appropriately', and those who are outside of it 'and may be less able to interpret what is taking place.' This differentiation 'become[s] increasingly blurred in public settings' and is an important consideration when designing interactive gallery installations or similar ICMSs. He also highlights the importance of "feedthrough" information (Dix, 1997, pp. 147-148; Benford, 2010), which applies not only to musical/interactive/performative circumstances but also to many non-musical, everyday scenarios but, in essence and within the context of HCI in music, systematically provides each user/performer within a collaborative interactive environment with visual feedback via their own dedicated UI containing data and information relating to the actions of their fellow participants and how this is impacting on the overall response of the system.

Additionally, Benford (Benford et al., 2005; Benford, 2010) suggests ICMS developers should aim for the control-gestures required to operate their system's UI to always include the characteristics of three specific types of motion: *expected* - actions/movements the user(s) may anticipate as useful due to their initial, intuitive perception of the system's operational protocols, which may be subject to the system's metaphorical connotations and/or physical affordances; *sensed* - actions/movements the system hardware is able to detect;

desired - the specific actions/movements used to impart control/influence over the output of the system. This approach to design was originally devised by Benford in relation to sensory-based ICMSs, but is equally apt when designing GUIs for touchscreen-based systems and, of course, is incorporated into that of *ScreenPlay*. Much like the *expressive* design strategy described above, *expected* control gestures are particularly prominent when interacting with the GUI of *ScreenPlay*; again, because of its similarities in mode of interaction and aesthetic appearance to that of common instruments and control objects respectively. *Expected* gestures are also heavily reliant upon the learned cultural codes and conventions discussed in the first section of this chapter in relation to Ferdinand de Saussure's signifier + signified = sign binary opposition.

Of equal importance when developing a GUI for a touchscreen-based ICMS is to give ample consideration to its affordances, which are defined by the design-choices made in relation to Blaine and Fels' *Physical Interface/Sensor* criterion. In very broad, non-musical terms, 'The affordances of [an] environment are what it offers [an] animal, what it provides or furnishes' (Gibson, 1979, p. 126). More musically relevant, however, is the definition given that 'Affordances refers to the properties of an object and the person. It's the relationship between the object and the person, and what the person can do with that object.' (Norman, 1994). Norman (2002; 2004) expands on this definition by distinguishing between *real affordances* (those expressed by the physical attributes of an object) and *perceived affordances* (those that exist through the metaphorical/symbolic representation of language and imagery), as well as laying out four principles for screen-based interfaces in relation to their affordances: 1) 'Follow conventional usage, both in the choice of images and the allowable interactions'; 2) 'Use words to describe the desired action'; 3) 'Use metaphor'; 4) 'Follow a coherent conceptual model so that once part of the interface is learned, the same principles apply to other parts' (2004). 'In graphical, screen-based interfaces, all that the designer has available is control over perceived affordances.' The display of a computer, tablet or smartphone affords touching regardless of whether or not the screen is touch-sensitive: this is a *real affordance*, as it relates to human interaction with a physical object. The *perceived affordances*, in relation to screen-based interfaces, are those which evoke or signify that touching or clicking on a particular graphical object displayed on the screen will result in a meaningful interaction. Norman states:

> it is wrong to argue whether a graphical object on the screen "affords clicking." It does. The real question is about the perceived affordance: Does the user perceive that clicking on that location is a meaningful, useful action to perform? (2004)

As would be expected, all four of Norman's principles are accounted for in the design of *ScreenPlay*, and their definitions and implementation will be discussed at length in the Methodology and Implementation chapter of the thesis. The use of metaphor in particular is of vital importance in imparting control over the topic-theory-inspired transformative algorithm that underpins the system design-model.

2.7 Timbre and Spectromorphology

As has been touched upon previously, Xenakis - influenced by the work of his teacher, Messiaen, in which 'colour and texture frequently predominate over pitch' (Matossian, 1986, p. 61) - focussed a great deal of attention on the use and manipulation of timbre in his works; as opposed to an over-reliance on purely tonal and rhythmic characteristics to shape a composition. Specifically, in his work *Metastaseis* (1953-54), Xenakis implemented traditional articulation techniques such as glissando, pizzicato etc. in a non-traditional way 'by scoring them for large sections of the orchestra playing en masse at varying speeds, on different pitches, in aggregates or various combinations to achieve the textural fields fundamental to the non-hierarchic and multi-directional character of the music' (Matossian, 1986, p. 61; Xenakis, 1992, pp. 9-10). While similar compositional techniques were also applied in a number of his other works (1992), as well as exploring the use of timbre in vocal music through the use of phonemes and syllables in *Nuits* (1967; Matossian, 1986, pp. 207-208; Souster, 1968), he also applied the inverse of this textural-/timbral-centric approach when composing *Herma* (1960-61), in which he purposefully limited and/or removed the use of different modes of attack, accents and rhythmic elements in favour of applying sieve theory to pitch-selection. The omission of these features, ultimately, was not detrimental to the resulting musical work; however, ordinarily 'it is these very choices which produce a living music, closely tailored to the instrument instead of a dry sampling of pitches which might have been the result of such a system in other hands' (Matossian, 1986, p. 156). The values texture and timbre brought to orchestral music were also appreciated by Varèse, who experimented with 'Effects like timbral shifts in a single note, or complex attacks created by pitched and non-pitched instruments simultaneously … as well as a very special concern with dynamics that could be compared to the techniques of amplitude envelope shaping in electroacoustic music.' (Guedes, 1996, p. 7)

As a theoretical approach to analysing the textural and timbral qualities within music, or more specifically, 'the interaction between sound spectra … and the ways they change and are shaped through time' (1997, p. 107), Denis Smalley developed the concept of *spectromorphology*. Spectromorphological analysis/evaluation is chiefly concerned with the ideas of *source-cause interaction*, *source bonding* and *sounding gesture* - all three of which are linked. Through use of the *spectromorphological referral process* (reversal of the source-cause interaction chain) (p. 111) the listener is able to attempt to identify both the sounding-body and gesture-type used to create any given sound. 'in traditional music, sound-making and the perception of sound are interwoven, in electroacoustic music they are often not connected' (p. 109) and, thus, analysis/identification of the source-cause interaction properties of a sound can be more difficult. However, the concept of *technological listening* (again, attributed to Smalley), whereby the technology or technique behind a sound/piece of music is perceived by the listener as opposed to the music itself, is often applied by the listener (consciously or otherwise) to the process of spectromorphological analysis within the context of electronic/electroacoustic musics (p. 109).

Source bonding, 'the *natural* tendency to relate sounds to supposed sources and causes, and to relate sounds to each other because they appear to have shared or associated origins' (p. 110) is reliant upon what Smalley calls intrinsic and extrinsic "threads". Intrinsic threads can be defined as 'sound events and their relationships as they exist within a piece of music', while the extrinsic relate to the 'foundation in culture' outside of a musical work that allows the intrinsic thread to exist/have meaning (p. 110). According to Smalley, 'intrinsic spectromorphological description … should be capable of helping a listener to pinpoint those musical qualities which are carriers of meaning.' (p. 111)

Sounding gesture is the association of sounds with their likely sources that are culturally embedded in the consciousness of the listener, much like the learned cultural conventions upon which Structuralist binary opposition models are founded. As the likelihood that a sound-source will be recognisable from the sound itself becomes more and more remote, Smalley refers to his concept of *gestural surrogacy*. *First-order surrogacy* relates to instances in music where both the sound-source and gestural cause of a sound are recognisable when hearing it. *Second-order surrogacy* deals primarily with 'traditional instrumental gesture', although it also applies to instrumental simulation. *Third-order surrogacy* is adopted by a sound when the reality of its source and/or gestural cause are difficult to distinguish by the listener upon hearing it. Finally, there is *remote surrogacy*, which applies only when both the 'Source and cause become unknown and unknowable as any human action behind the sound disappears' (p. 112). Related to *gestural surrogacy* as a mode of musical analysis and, in particular, third and fourth-order surrogacy, is the idea that 'If gestures are weak, if they become too stretched out in time, or if they become too slowly evolving, we lose the human physicality. We seem to cross a blurred border between events on a human scale and events on a more worldly, environmental scale' (p.113). Because of this Smalley concludes that 'A music which is primarily textural, then, concentrates on internal activity at the expense of forward impetus.' (pp. 113-14)

Despite the textural/timbral shaping of *ScreenPlay*'s musical output carried out by the topic-theory-inspired transformations the theories relating to spectromorphology discussed thus far do not directly influence the design but, instead, are passively relevant through the way in which they present a general association with all musics - in particular those which depend heavily on synthesized and sampled/re-sampled, or "non-human", sounds (traditional instrumentalism could also be described as non-human, but these sounds and their associations are much more culturally ingrained than the wide variety of textural/timbral possibilities related to electronic sound-creation) such as electronic and electroacoustic music. The same is true of other topics touched upon by Smalley, such as audience anticipation when listening to music (pp. 112-115) and motion and growth processes (pp. 115-117) within the context of electroacoustic music where 'traditional concepts of rhythm are inadequate to describe the often dramatic contours of electroacoustic gesture and the internal motion of texture which are expressed through a great variety of spectromorphologies' (p. 115). Smalley also covers a few other areas specific to how humans hear pitch differences and the movement between them, as well as electroacoustic music, which are, again, not directly applicable within the context of *ScreenPlay*.

There are, however, certain concepts outlined in Smalley's work that bear a great deal more relevance to ICMS design, and that of *ScreenPlay* in particular. For instance, there exists a direct connection between the four qualifiers of spectral space described by Smalley (emptiness-plenitude, diffuseness-concentration, streams-interstices, overlap-crossover) - in particular the first two - with the application of topical oppositions made available to the user(s) by *ScreenPlay* in order to encourage creative expansion and structural development in both composition and performance (p. 121), and in which texture/timbre also play an important role. Additionally, there are considerations to be made in terms of ICMS installation design when related to Smalley's theory of space and spatiomorphology - 'Spatial perception is inextricably bound up with spectromorphological content' - 'higher pitches can be thought of as spatially higher, and lower pitches lower' (p. 122), which is applicable to *ScreenPlay* because it can exist as both a collaborative, interactive installation focussed on group improvisation in electronic music, as well as studio compositional tool to aid in the creative process. Also of interest when discussing ICMS installation design is the differentiation between the *composed space* ('the space as composed on to recorded media' (p. 122)) and the *listening space*. The listening space can be divided into two categories: personal, where the listener is generally close to and directly facing the sound-source; and public, where the 'listener could be in any one of a variety of distant or off-centre positions relative to a frontal reference-image [or within a multichannel installation].' In any case, 'spectromorphology becomes the medium through which space can be explored and experienced. Space, heard through spectromorphology, becomes a new type of "source" bonding.' (p. 122)

2.8 Topic Theory

Topic theory is implemented in the design of *ScreenPlay* as a way of generating new musical material and affecting the texture/timbre of the existing material in response to topical opposition transformations controlled by the user(s) in order to break routines of form, structure and style in collaborative, improvisatory performance as well as to aid compositional problem-solving. Particularly prevalent in Classical and Romantic music, topic theory is closely related to Structuralist binary opposition models, such as Ferdinand de Saussure's linguistic theory of signifier and signified (Monelle, 2000), through the way in which specific musical identifiers are utilised deliberately by the composer to evoke certain emotional responses and cultural/contextual associations from the audience; these connections being exemplary of metaphorical binary oppositions. In *ScreenPlay* these metaphorical binary oppositions are used in reverse to verbally describe to the user(s) the audible effects that a variety of topical oppositions made available to them will have on the resultant musical output of the system; while the topical oppositions themselves are also, in fact, binary oppositions. Monelle states that 'There are two semantic levels in song, one verbal, the other wordless. The wordless level is present in all music, of course. Music is wordless song' (2000, p. 9). Taking the example "horse" in relation to Saussure's signifier/signified theory: 'A musical horse carries a much greater burden of imaginative signification than the simple linguistic term. The musical horse is noble, masculine, adventurous, warlike, speedy' (Monelle, 2006, p. 24); 'the musical horse provides more information than the linguistic horse because it adds the imaginative dimension' (p. 25). In other words, the musical signified has the potential to transmit a far more detailed account of itself than the linguistic signified due to the 'imaginative dimension' (p. 25) and its capacity to evoke descriptive details of itself without the need for additional adjective/verbal discourse - as is the case with the literary signifier/signified. 'Music cannot be translated into language; on the contrary, it chastens language by drawing out its limitlessness' (Monelle, 2000, p. 13). Also noted by Monelle, which is perhaps indicative of the reasoning behind the move away from the score as the primary musical artefact in many genres of music in favour of the recording (discussed in the first section of this chapter), is that analysing music within this framework serves to reveal a musical *text*, which he describes as being able to provide 'a great deal more [information] than merely the score.' (p. 11)

As a result of the difference between linguistic and musical signifieds, Peirce expands the analytical application of the signifier, signified, sign linguistics model to music by extending the definition of the sign to include *icon*, *index* and *symbol* (1940, pp. 102-103; pp. 104-115).

> Iconic signs *resemble* their object, as a silhouette of a man with a spade may mean "road up" … Symbolic signs depend on learned cultural codes; thus, the word "tree" has nothing in common with a tree, but is understood by a speaker of english to carry this signification … [The] *index* [is] a sign that signifies by virtue of contiguity or causality, as when a hole in a pane of glass brings to mind the bullet that passed through it and caused it. (Monelle, 2000, pp. 14-15)

'musical meaning is "expressive" or related to the "emotions"' (p. 11) but, despite this fact, signification is not dependent on the perception of individual listeners. This argument is compounded through Monelle's belief that 'the musical topic … clearly signifies by ratio facilis … [Umberto Eco's theory, 1976] in which signification is governed by conventional codes and items of expression are referred to items of content according to learned rules' (p. 16) and echoed by Smalley when discussing spectromorphology:

> Spectromorphological thinking is basic and easily understood in principle because it is founded on experience of sounding and non-sounding phenomena outside music, a knowledge everyone has - there is a strong extrinsic-intrinsic link … [music] must have some shared natural-cultural basis if [it is] to make sense to listeners. (1997, p. 125)

Monelle identifies and discusses at length in *The Musical Topic* (2006), 'The three great topical genres, hunting, soldiering, and shepherding' (p. ix), in addition to providing multiple examples in different works of certain specific topics (including: the *pianto*, *Dance of Death*, *Sturm und Drang* (storm and stress) and *locus amoenus*). Although none of this information has a great deal of relevance to conventional modern-day ICMS design it is of importance in relation to the use of topic theory in *ScreenPlay* - particularly the musical characteristics and metaphorical associations of certain specific topics. Monelle also points out that each topic (using the *hunt* as an example) 'carries a "literal" meaning, together with a cluster of associative meanings'; which is a worthy consideration given the algorithmic implementation of aspects from topic theory in *ScreenPlay* (p. 3). This view is more in line with that of Agawu (1991) who, although agreeing with Monelle's sentiment discussed in the previous paragraph that topics are reliant upon learned cultural conventions - even taking it a step further to suggest that there existed a topical vocabulary shared by both audience and composer during the Classical period that would be exploited by the composer knowing the recognition of musical meaning/association was reliant upon a knowledge of this (p. 33), goes on to state that 'Topics … are points of departure, but never "topical identities." … even their most explicit presentation remains on the allusive level. They are therefore suggestive, but not exhaustive' (p. 34). This point that the association/response evoked from different individuals within the audience cannot be guaranteed due to their own views/experiences/heritage/culture etc., along with the assertion that 'a given topic may assume a variety of forms, depending on the context of its exposition, without losing its identity' (p. 35), makes the application of topic theory in the context of *ScreenPlay* as a way of generating new musical ideas and breaking routines of form and structure ideal, given that the source material upon which transformations are applied cannot be predefined within the rules of the transformative algorithm as a result of it being generated by the user(s) of the system.

Also of great relevance in the context of *ScreenPlay* (at least theoretically) is the assertion of Agawu (1991) and Ratner (1991, p. 615), highlighted by Monelle, that topics 'work best' in Classical music (2006, pp. 7-8). This may have been true at one time but nowadays it could certainly be argued that the compositional implementation of topics is perhaps best suited to electronic music, due to the vast array of potential spectromorphologies that are able not only to emulate and replicate the sonic characteristics of traditional/acoustic instruments but also to invoke imagery of other-worldly environments/creatures/machines and mechanisms made possible by sound-generation techniques such as synthesis, sampling and resampling etc.. As well as resonating (in part) with Denis Smalley's idea of *technological listening* (1997), the aspects of electronic music discussed above also display similarities with Latin authors of the Middle Ages:

> [who] learned their craft by studying rhetoric and poetry of the ancient Roman world … [resulting in the] literary topoi adopt[ing] significations from elsewhere and from other times[;] they did not refer to any aspect of the real social world of their time, but rather to an imaginative world. (Monelle, 2006, p. 12; see also Curtius 1948/1953 [English Translation], pp. 183-185: "Exotic Fauna and Flora")

Monelle drew two conclusions from this, which both reflect favourably on the application of topic theory in electronic music for the reasons outlined above. 'First, in the case of topics, the signifier and signified are not necessarily contemporary or local to each other … Second, the topical signified may be wholly imaginary, a reflection of cultural fantasies' (2006, pp. 12-13). He continues by saying that 'Theorists of the literary and musical topic, therefore, must take care not to assume that signifier and signified are necessarily contemporaneous, or even that the signified was ever part of the social and material world.' (p. 13)

Agawu is predominantly concerned with how topics function within/affect Classical (or, as he refers to it, 'Classic') music, rather than their signification - 'not "what does this piece mean?" but, rather, "*how* does this piece mean?"' (1991, p. 5) This is an ethos much more in line with the aesthetic approach to interactive electronic music composition and performance being explored through the development of *ScreenPlay*, despite Monelle's assertion that 'The primary concern of the topic theorist is to give an account of each topic in global terms, showing how it reflects culture and society, not to focus on music alone' (2006, p. 10). Agawu applies this take on topic theory to the analysis of form and structure in Classical music in relation to his *beginning-middle-end* paradigm, of which he says 'there are specific attitudes to a work's beginning, its middle, and its ending, and that these strategies are an important clue to the dramatic character of Classic music' (1991, p. 51). The suggestion here is that certain topics are more suited to certain sections of Classical music and thus will more often than not appear in these sections. In support of this Agawu cites a number of theorists, including Johann Mattheson who believes that, in order to best represent the rhetorical strength of their musical ideas, composers should 'begin with their strongest arguments, present the weaker ones in the middle, and close with stronger ones again' (Mattheson & Lenneberg, 1958, p. 201).

Although the use of form and structure in *ScreenPlay* is not relevant in this context - given that the musical output of the system is freely formed by the user(s) from the clips they have recorded and the fact that very few potential users would have a theoretical understanding of topic theory and its application to form and structure - the initial point made by Agawu with regard to *how* topics mean taking precedent over *what* they mean does apply. This is because the user is made aware of the literal meaning of the topical opposition transformations available to them by the labelling provided on the TouchOSC GUI. What is important is how these literal meanings are translated into the musical output of the system so that the user can make an informed decision over the effect a particular transformation will have on the musical output. In essence the application of topic theory in the topic-theory-inspired transformative algorithm is a reversal of roles of music and meaning in the traditional sense.

2.9 Breaking of Routine in Collaborative, Improvisatory Performance and Aiding Composition through the Development of New Musical Ideas

A final consideration of importance to the design of *ScreenPlay* is that of facilitating changes in form, structure and style upon collaborative, improvisatory performance, and problem solving through the generation of new musical ideas during the compositional process; as has been touched upon in previous sections. In 1975 Brian Eno and Peter Schmidt published a collection of idea cards for combating creative block during the process of music-making, *Oblique Strategies* (1975; Oblique Strategies, n.d.). This is still a major problem today for many electronic music composers/producers and can perhaps be even more pronounced due to a lack of instrumental interaction - although there are many tactile MIDI controllers that aim to remedy this situation in various ways - as evidenced by the recent publication by Ableton of the book *Making Music: 74 Creative Strategies for Electronic Music Producers*, written by their Head of Documentation, Dennis DeSantis (2015).

The main patch of *ScreenPlay* is programmed as a number of Max for Live (Max/MSP running within Ableton Live) MIDI Devices, meaning that not only can it be set up and run as an interactive installation promoting collaborative, improvisatory performance for novice musicians (as well as those with more experience), but it can also be used by any computer-musician working within Ableton Live and running Max for Live as a tool for composition. In this context, the system can either receive MIDI input from the dedicated, touchscreen-based GUI (which has been designed as a TouchOSC layout) or from any standard MIDI controller/keyboard/input-device, and aims to formulate with the composer a two-way communicative creative process which will result in the generation of ideas in response to those of the users that perhaps they would/could not have envisaged themselves; thus opening up a plethora of new and interesting compositional possibilities.

3. Methodology and Implementation

Following the compilation of the background research presented in the Literature Review it was possible to identify in the wealth of material that had been analysed and evaluated the gaps that existed - whether conceptually, theoretically or in practice - and how it was possible to use the process of designing and developing *ScreenPlay* as a way of filling these gaps in new and unique ways. Most notable was the complete lack of ICMSs adopting the use of more than one of the three overarching approaches to system design (*transformative*, *generative* and *sequenced*) and the negative impact of this restriction on the interactive experience provided by these systems as a result, as well as the equally inadequate application of topic theory outside of Classical and Romantic music; with particular regard to electronic music and the field of HCI in music. The adoption of all three approaches to system design by *ScreenPlay* serves to emphasise the respective strengths of each and at the same time negate their weaknesses, while both topic-theory-inspired transformative and Markovian generative algorithms are employed as a means of breaking routine in collaborative improvisation, generating new musical ideas in composition and providing new and additional dimensions of expressivity.

Also identified as lacking in the field of HCI in music during the process of assembling the Literature Review was the ability of any existing ICMSs to function both as a true multi-user-and-computer collaborative interactive system and a single-user-and-computer interactive compositional tool. *ScreenPlay* is successful not only in addressing this deficiency but is also evolutionary in its implementation of multi-user support with respect to existing examples by affording each individual user with a dedicated GUI and therefore increased influence/control of the musical output of the system.

Subsequent to the determination of these key areas for which to account in *ScreenPlay* it was established that the best way of approaching the design process was to follow the template of an iterative and incremental development cycle, which has been widely practiced in the field of software development since the mid-1950s (Larman & Basili, 2003). Typically, iterative and incremental development is a cyclical, evolutionary process in which multiple iterations of a product are produced over the course of its entire lifecycle, each of which seeks to build upon the successes and improve upon the weakness of the one preceding it and involves numerous different stages including planning (comprised primarily of an analysis of its predecessor and any other contributory research), design/programming, and testing

(Mitchell & Seaman, 2009). It is also not uncommon for this approach to design and development to involve the collection and consideration of feedback from users/focus groups during the testing periods of different iterations (Larman & Basili, 2003), as is exemplified by the work carried out and addressed in the Case Studies chapter of the thesis.

The programming for *ScreenPlay* is broken down into three separate Max for Live MIDI Devices; each of which fulfils a unique purpose. *ScreenPlay-CTRL* is the central juncture through which the TouchOSC GUIs connect to the system, and communicates the incoming control data to both Ableton Live and the other two Max for Live MIDI Devices that make up the system. *ScreenPlay-GEN* houses the Markovian generative algorithm, while *ScreenPlay-TRNS4M* is responsible for conducting the probability-based algorithmic transformation of the "joy-lament" topical opposition and communicating the data relating to the three textural/timbral topical opposition transformations - "stability-destruction", "open-close", and "light-dark" - to *ScreenPlay-FX*, the accompanying Ableton Live Effect Rack. Originally programmed in Max/MSP before being migrated to Max for Live, Max was the logical choice of programming language in which to develop *ScreenPlay*, due to its recognised support of musical applications. Max for Live, in particular, is an ideal platform in terms of maximising the impact of the system not only in the field of HCI in music but also in popular electronic music given the widespread popularity within the electronic music composition/production/performance community and dance music/club culture of Ableton Live. As a result, the intended functionality of the system, which began solely as a collaborative, improvisatory performance installation for multiple users utilising four touchscreen tables to host the TouchOSC GUIs and exemplary of the "Computer Network Music" genre (Early Computer Network Ensembles, n.d.; Brown, n.d.) when programmed as a self-contained Max/MSP patch, has been able to expand and include also the functionality of a single-user studio compositional tool, which implements tablets instead of the touchscreen tables to host the interfaces. The existing and rapidly expanding commercial market for Max for Live devices also represents the best possible opportunity for *ScreenPlay* to have a potential commercial impact.

Similar to the reasons for utilising Max to develop the system programming, *TouchOSC* (Fischer, 2008-present) was, again, the logical choice of software through which to communicate and interact with *ScreenPlay*, as a result of its availability and affordability, universal compatibility with Android and iOS devices, its operational flexibility, sensitivity, and stability, and support of fully customisable interfaces/templates. Before discussing further the technical processes involved in developing the system, the theoretical and conceptual basis of *ScreenPlay* and the considerations in design made as a result of this will be explored.

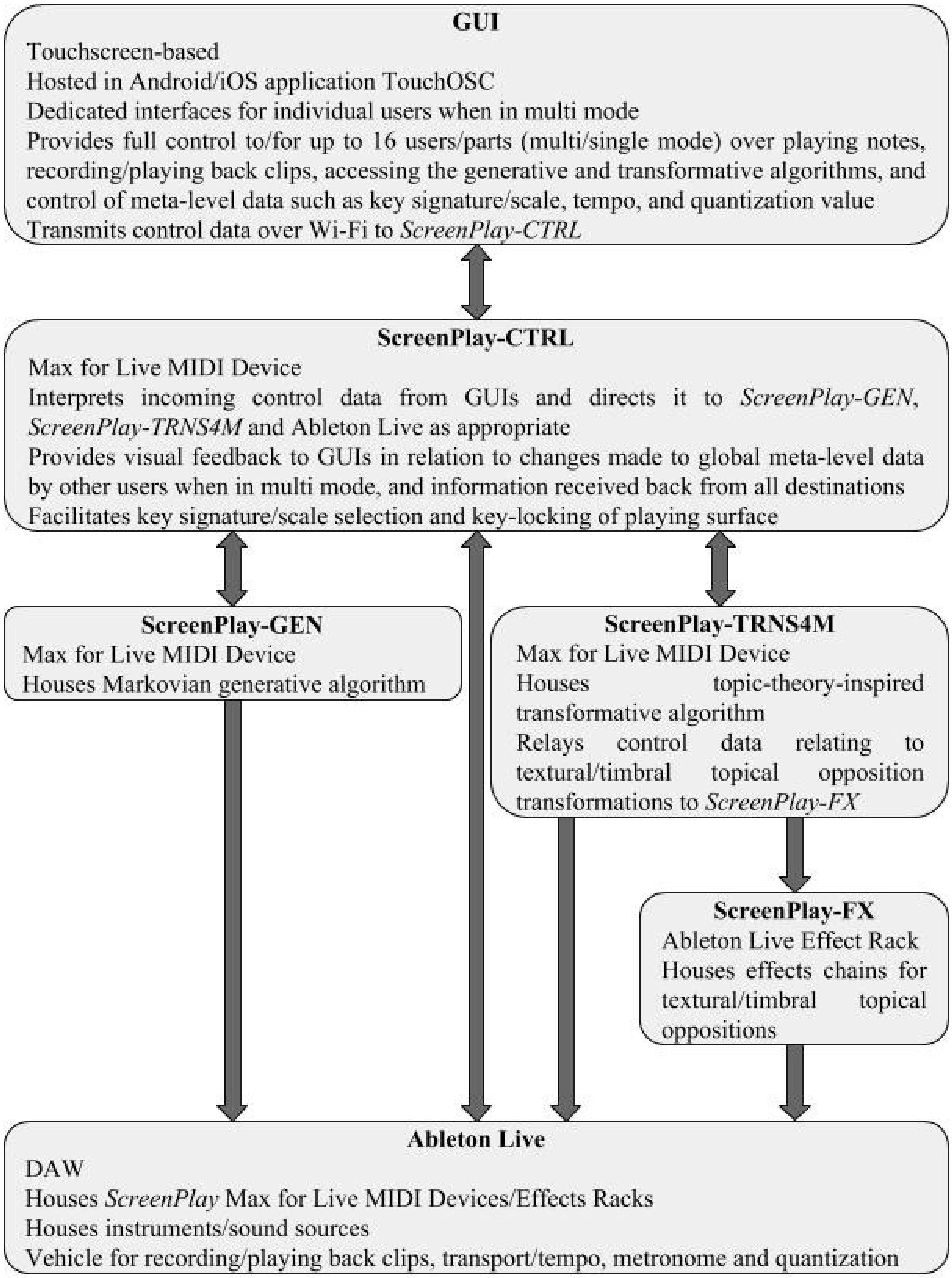

GUI
Touchscreen-based
Hosted in Android/iOS application TouchOSC
Dedicated interfaces for individual users when in multi mode
Provides full control to/for up to 16 users/parts (multi/single mode) over playing notes, recording/playing back clips, accessing the generative and transformative algorithms, and control of meta-level data such as key signature/scale, tempo, and quantization value
Transmits control data over Wi-Fi to ScreenPlay-CTRL
ScreenPlay-CTRL
Max for Live MIDI Device
Interprets incoming control data from GUIs and directs it to ScreenPlay-GEN, ScreenPlay-TRNS4M and Ableton Live as appropriate
Provides visual feedback to GUIs in relation to changes made to global meta-level data by other users when in multi mode, and information received back from all destinations
Facilitates key signature/scale selection and key-locking of playing surface
ScreenPlay-GEN
Max for Live MIDI Device
Houses Markovian generative algorithm
ScreenPlay-TRNS4M
Max for Live MIDI Device
Houses topic-theory-inspired transformative algorithm
Relays control data relating to textural/timbral topical opposition transformations to ScreenPlay-FX
ScreenPlay-FX
Ableton Live Effect Rack
Houses effects chains for textural/timbral topical oppositions
Ableton Live
DAW
Houses ScreenPlay Max for Live MIDI Devices/Effects Racks
Houses instruments/sound sources
Vehicle for recording/playing back clips, transport/tempo, metronome and quantization

Figure 1. System overview diagram.

3.1 The 11 Criteria for Collaborative Musical Interface Design

A large part of the development of *ScreenPlay* involved the incorporation into its design of some key elements collated from a number of well established ICMSs, whilst simultaneously aiming to address the deficiencies exhibited by these systems in relation to the theoretical and conceptual approaches to ICMS design against which they were scrutinized. One concept taken into consideration was Blaine and Fels' 11 criteria for collaborative musical interface design (2003), for which below has been provided a full breakdown and explanation of each:

1. *Focus*: The efforts made to increase the audience's "transparency" (Fels, Gadd & Mulder, 2002) - i.e. the ease with which members of the audience are able to discern the connection between the actions of the user(s)/performer(s) when interacting with the system and the resulting musical output. 'Generally with novice participants, the sound generated within collaborative musical environments is intended for the players' (Blaine & Fels, 2003, p. 414), but Blaine and Fels also note that, if it is the intention that the musical output should be such that it can be appreciated by a wider audience, then this "transparency" becomes essential.
2. *Location*: The impact had by the location of an ICMS's interface(s) on the ability of users/performers to learn from each other how best to interact with the system in terms of the relationships between certain gestures and their corresponding sonic events and sequences - i.e. whether the UIs are 'co-located' (sharing the same performance-space) or randomly distributed 'over a network [at] non-specific locations' (p. 417), as is often the case with "Computer Network Music" systems (Early Computer Network Ensembles, n.d.; Brown, n.d.) that are reliant upon the internet to connect multiple players.
3. *Media*: The use of audiovisual elements within the interactive system 'as a way of enhancing communication and creating more meaningful experiences … by reinforcing the responsiveness of the system to players' actions', as well as the increased potential for '[distracting] players from seeing other players' actions, or from attending to aural elements' (Blaine & Fels, 2003, p. 417), which can ensue as a result of overemphasis on the visual aspect of a system's response to the user(s).
4. *Scalability*: The constraints imposed upon the depth-in-control afforded to the users/performers by the UI over the generated musical output of a system, resulting

from the number of individuals by which the system is intended to be used simultaneously and vice versa. 'An interface built for two people is generally quite different from one built for tens, hundreds or thousands of players.' (p. 417)

5. *Player interaction*: The effects of providing each participant in a collaborative interactive system environment with either the same, similar or differing UIs. An increased presence of identifiable similarities between the interfaces of all performers can 'lead to a more relaxed environment and more spontaneous group behaviours' (p. 417) as a result of an improved collective understanding of what each individual is contributing to the overall auditory experience.
6. *Musical range*: The need to strike the correct balance when limiting the range of freedom afforded to the player(s), in terms of melodic/harmonic chromaticism and the authority of quantizing rhythmic input etc., so as to allow substantial room for improvisatory expression, flair and embellishment while ensuring the resulting musical output generated by the system is coherent, pleasurable, and provides a sense of satisfaction to novice users lacking in musical knowledge and experience.
7. *Physical interface/sensor*: The importance of implementing a suitable means of interfacing with the system, be it tangibly through touch and pressure sensors or gesturally via motion capture and/or body-mounted motion sensors. This is what defines the affordances of a system.
8. *Directed interaction*: The decision whether or not to provide the user(s) with guidance as to more effectively and efficiently interacting with the system and/or each other through it by way of the presence in the performance space of a trained instructor or, alternatively, more rigid performance structures, such as the implementation of a "call and response" scheme with definitive roles for "leaders" and "followers" within the group; as well as consideration of how enforcing such a system would impact upon the overall interactive experience.
9. *Learning curve*: 'an evaluation of the tradeoff between speed of learning and musical constraints' (p. 418). The task of fulfilling the requirements of providing an engaging and satisfying musical experience for novices and first-time users, while leaving scope for improved and more virtuosic interactions with practise in order to captivate the player(s) beyond the point of initial intrigue and exploration of a system's functionality and architecture.

10. *Pathway to expert performance*: The advantages of providing the user(s) with experience-dependent "modes" such as novice, intermediate and expert, which become incrementally less restrictive and thus maximise the possibility for freedom of expression in performance but also for user-input error and mistakes - a system very much like that of selecting the difficulty level of a computer game.
11. *Level of physicality between players*: The effect of requiring users/performers to physically collaborate with each other by working together in order to fully exploit the potentialities of the ICMS, and the impact this can have on the levels of engagement with the system and sonic/performative experience felt by individuals within the group. 'Most often, it is not the interface itself that makes for an engaging, satisfying experience, but the group ambience and development of synergistic relationships between players that leads to positive communal experiences.' (p. 419)

The grid-based design of the playing surface on *ScreenPlay*'s GUI (as seen in Fig. 2) is inspired by that of the Ableton *Push* (2013) and, as such, allows users to "lock" the pitch-intervals between individual "pads" on the button-matrix to those of a specific scale. The notes of the scale are assigned to the "pads" of the input-matrix from the left to right; but also so that, for every 3 notes ascending along the horizontal row, the 3 positioned directly above continue that ascent, thus allowing for any standard triad within the chosen key signature/scale to be formed using the same hand-shape anywhere on the grid. In addition, the "pads" to which the root-notes of the scale are assigned are clearly displayed through the use of a different background colour to that used by the rest of the "pads", providing the player(s) with a frame of reference that enables them to orientate themselves with greater accuracy when interacting with the playing surface (see Fig. 3). This is in coordination with the guidelines for *musical range* as outlined in Blaine and Fels' 11 criteria, by ensuring a pleasurable, satisfying and immersive experience for novice and first-time users lacking in prior musical knowledge; while the ability to enable record quantization at a value of 1/4, 1/8 or 1/16 also serves to fulfil this criterion. Key selection is facilitated by way of a selection matrix (Fig. 4), with the root-note of the key denoted by the position of the selection-marker in relation to the x-axis and the scale/mode denoted by the marker's position on the y-axis - the scales/modes that are available for selection are: Major, Minor, Dorian, Phrygian, Lydian, Mixolydian, Aeolian and Locrian.

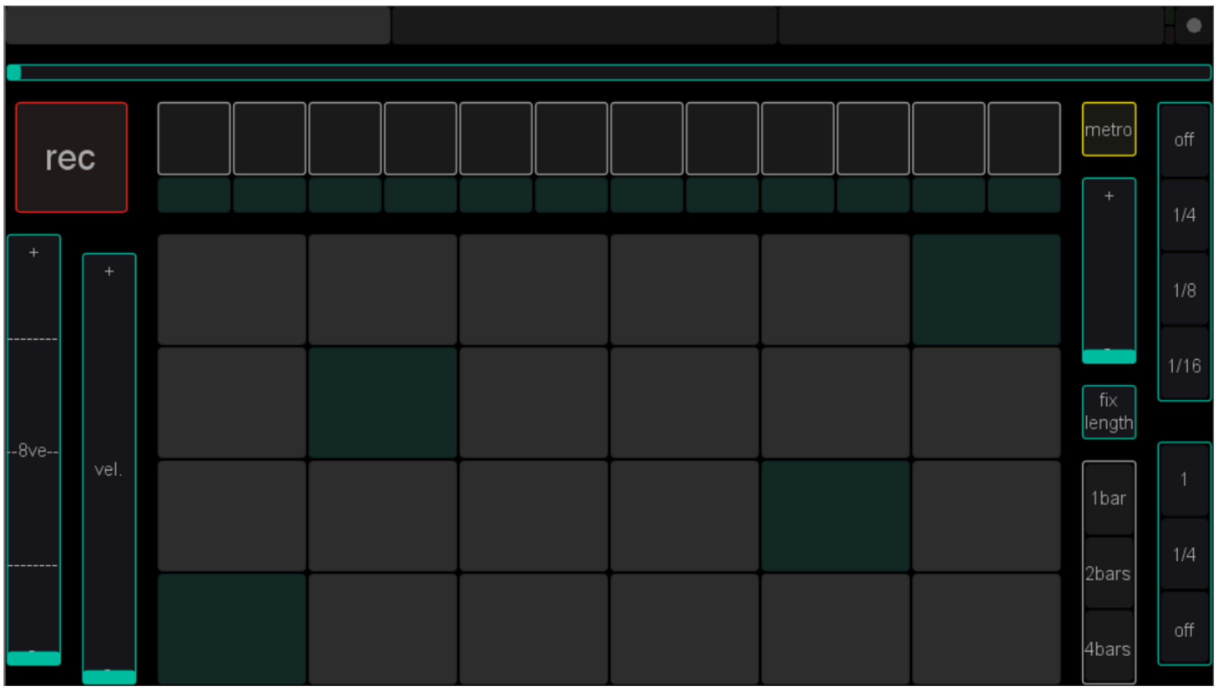


Figure 2. *ScreenPlay* GUI main page, including the grid-based playing surface inspired by that of the Ableton *Push* controller (2013).

B	🟦 **C**	**D**	**E**	**F**
F	🟨 **G**	**A**	**B**	🟦 **C**
🟩 **C**	**D**	🟨 **E**	**F**	**G**

Figure 3. Diagram depicting how a C major chord (highlighted in green/yellow) can be played using *ScreenPlay*'s grid-based playing surface in the key of C Major; the root-notes of which are coloured blue.

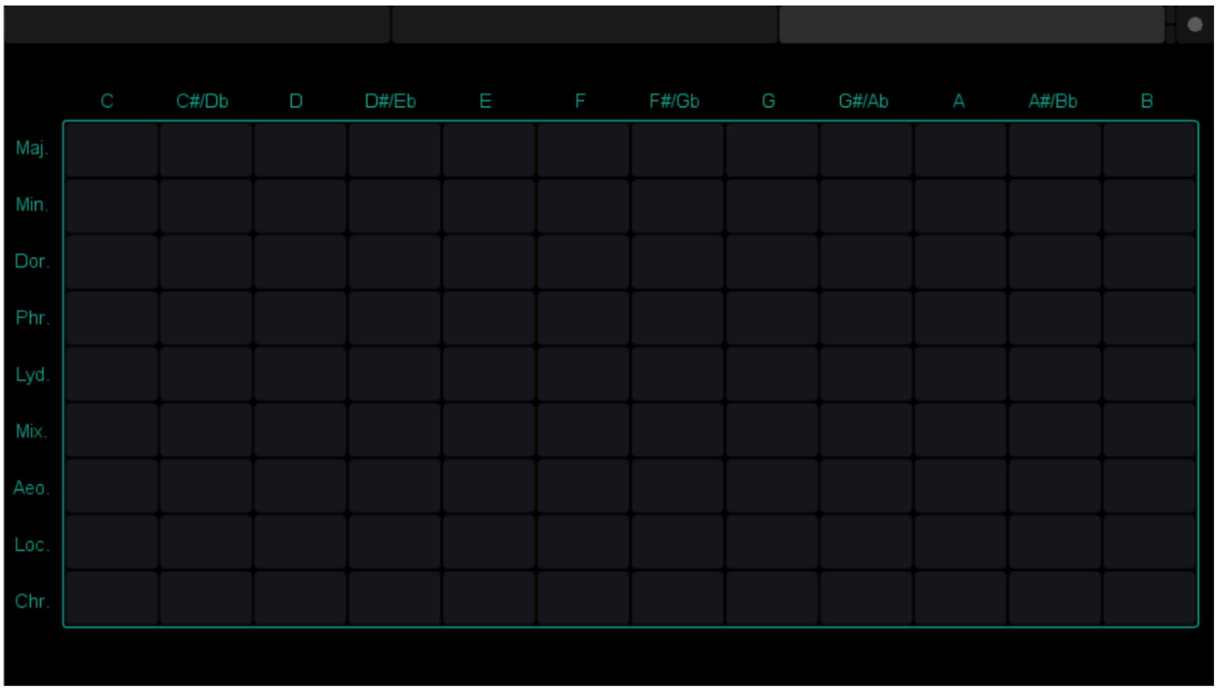


Figure 4. *ScreenPlay* GUI key selection matrix.

By using digital representations of control-objects commonly associated with electronic music production and live performance, such as drum pads, buttons/switches and faders etc., recognising a direct connection between input gestures/commands and the resulting musical output of the ICMS is obvious not only to the player(s) actively engaging and interacting with the system but also to audience members and bystanders not directly involved in the interactive experience. As such, *ScreenPlay* can be considered highly transparent in relation to Blaine and Fels' guidelines on *focus* for the design of collaborative musical interfaces. This level of familiarity - rooted in the learned cultural conventions highlighted by Ferdinand de Saussure in relation to his signifier/signified binary opposition model (1959, pp. 65-70) - extends to the primary mode of interaction with the playing surface, which exhibits many common characteristics and similarities with the functionality and modes of interaction of traditional acoustic instruments like the piano; albeit in a different physical (or, more accurately in the case of *ScreenPlay*, virtual) layout, and thus provides a platform from which to embark for novice-level players interacting with the system for the very first time. Allied with the possibility of exponentially increasing one's proficiency in interacting with the interface and the ability to perform with virtuosity and freedom of expression through practise - in the same manner commonly associated with traditional musicianship via increased dexterity, muscle-memory training and an improved sense of rhythm and timing - the GUI of *ScreenPlay* manages as well to satisfy Blaine and Fels' recommendations with regard to the *learning curve* of a collaborative musical interface. Similarly, learned cultural conventions - this time those associated with the use of touchscreen devices, which have grown out of the increased worldwide prevalence of the technology in recent years - as well as the requirement that users should be afforded a high level of precision in imparting influence and control over the musical output of *ScreenPlay*, were important considerations when making the choice in relation to the *physical interface/sensor* criterion to utilise screen-based, rather than sensory-based, interfacing to control the system.

Because *ScreenPlay* is designed in such a way that, when in multi mode, each individual GUI connected to the central patch via any compatible Android/iOS device running TouchOSC is responsible for controlling one constituent part of the musical output (provided they are all assigned their own MIDI channel), the compromise highlighted by Blaine and Fels in their guidelines for *scalability* - whereby the depth-in-control afforded to individual players within the collective group is directly affected by the total number of players interacting with the system simultaneously - bears little relevance. This is due to the fact that the system is designed to accommodate one individual per interface and more interfaces can be added to the system simply by introducing more virtual instruments/parts or external sound-sources (synthesizers to which MIDI note information generated by the central Max for Live devices can be sent) to the overall arrangement of the final musical output; the total number of which is limited only by the sixteen separate channels supported by MIDI. When set up as a collaborative performance installation the interfaces are also co-located within the same performance space and, as such, the advantages of this mentioned in Blaine and Fels' guidelines for the *location* of collaborative musical interfaces are fully exploited with users able to learn from each other and better communicate whilst interacting with *ScreenPlay*. The encouragement of verbal communication between players during improvisatory performance, in turn, goes some way to addressing the intentions behind the *level of physicality between players* criterion, without actually making requisite the need for players to come into physical contact with one another; while each player is also provided with the exact same GUI as the next, as is suggested by the *player interaction* criterion.

With regard to the remaining criteria that have not yet been addressed in relation to *ScreenPlay*: in line with the *pathway to expert performance* criterion, it is possible not only to disable clip-trigger and record quantization and "key-locking" of the playing surface so as to display chromatic scales, but the system also allows users to bypass the playing surface completely and instead use any MIDI controller in order to play and record notes whilst still having access to the generative and transformative algorithms and all other functionality. The *directed interaction* criterion is satisfied by the inclusion of a pop-up help window in the *ScreenPlay-CTRL* Max For Live device GUI, which addresses how to correctly set up and organise the Ableton Live Set for both single and multiple users, as well as the provision of instructional documentation relating to the TouchOSC GUI. The use of audiovisual elements to improve the collaborative interactive experience, in accordance with the *media* criterion, is, unfortunately, not accounted for in the design of *ScreenPlay* due to the limitations of using TouchOSC to host the GUIs. However, the conceptual design-model of the system - through its support of individual GUIs for each player/user - certainly harbours the potential for future developments to facilitate the transfer of "feedthrough" information (Dix, 1997, pp. 147-148) between all of the interfaces in order to vastly improve the "Computer Network Music" (Early Computer Network Ensembles, n.d.; Brown, n.d.) aspect by allowing for significantly increased lines of communication between players, resulting from the provision of visual feedback relating to how each of the other users is interacting with their particular control surface and, perhaps, enabling crossover control of one another's GUIs/control-parameters where appropriate.

When compared on the strength of adhering to Blaine and Fels' 11 criteria for collaborative musical interface design to one of the main ICMSs from which *ScreenPlay* has drawn influence, *Reactable*/*Reactable Mobile* (Jordà, Kaltenbrunner, Geiger, & Alonso, 2003-present), it can be said that, in some respects, it is a more complete ICMS than is its counterpart. Where *ScreenPlay* maintains almost complete audience transparency, *Reactable* is transparent only in terms of controlling the pitch of oscillators, parameters of effects such as filter cutoff frequencies and distortion intensity etc., and triggering sample playback. Where there is little or no immediate transparency are the protocols for creating loops by inputting tonal and rhythmic information, designing sounds via oscillator wave-shape selection, modification, combination and ADSR/filter envelope shaping etc., assigning/loading samples for playback and, among other things, defining the key signature. Granted, some of these editable parameters/musical characteristics - specifically those concerned with sound-design/synthesis - are not supported with the same level of direct and precise depth-in-control in *ScreenPlay* as they are in *Reactable*, due to the fact that *Reactable* is better categorised as a digital modular synthesizer than it is an ICMS. The ability to sculpt and shape the textural/timbral qualities of the musical output directly from the GUI is supported by *ScreenPlay* but only as part of the topical opposition transformational framework. When configured as a single-user studio compositional tool as opposed to a multi-user collaborative performance system, however, both the sound source and the effects mappings of the topical oppositions can be edited directly by the user via Ableton Live, in addition to the application of any other effects. The lack of immediate transparency with regard to *Reactable*'s implementation of these controls comes as a result of their being accessible only through extensive sub-menus away from the main "table" display of the system interface.

Reactable is also limited when compared to *ScreenPlay* in terms of its s*calability*, as it is not possible to connect multiple systems together and, as such, multiple-performer collaboration is only achievable through a single, shared interface. As a result the total number of users able to interact with each other simultaneously through the system in improvisatory composition/performance is subject to the physical size of the interface, and, as that number increases, the depth-in-control and direct influence attributed to each player dramatically decreases due to the finite size of the "table" display GUI available in which to place "crystals" (the modules/objects used to trigger and control samples/oscillators/effects etc.), as well as the finite amount of "crystals" available for use in the first place. This restriction of space and "crystals" also means that collaborative, multi-user interaction is only really feasible with the table-top hardware instrument incarnation of the *Reactable* system, and not with the much more affordable and readily available *Reactable Mobile*.

In terms of *musical range* and *learning curve*, there are advantages and disadvantages to the GUIs utilised by both *ScreenPlay* and *Reactable*. Like *ScreenPlay*, the *Reactable* system also operates within the confines of a fixed key signature chosen at the discretion of the user(s). However, although the improvisatory performance of sustained-note melodies is made markedly easier for novice musicians and non-expert users with the GUI of *Reactable* than it is with *ScreenPlay* - by way of this being achieved simply through rotating the chosen oscillator "crystal" to move the pitch up and down incrementally through the notes of the selected scale, rather than the reliance of *ScreenPlay* upon traditional musicianship skills and technique - the option to perform and record motifs/lines/loops containing notes with any sort of rhythmic quality or jumps between pitches not adjacent to each other in the chosen scale is far less straightforward. Where in *ScreenPlay* all the control objects required to record and playback loops are present in the main GUI window along with the playing surface, these options, including the playing surface itself, can be accessed in various forms in *Reactable* but only via sub-menus away from the main "table" display. Any material - be it melodic, harmonic or rhythmic - intended by the user/performer to be recorded and triggered/played-back as a loop can only be inputted into the system via one of a few variations on a traditional step-sequencer or a graphical representation of a traditional piano keyboard (which, unlike the playing surface of *ScreenPlay*, supports monophonic input only). Despite Reactable's recent introduction to of the *ROTOR* iOS application (Jordà, Kaltenbrunner, Geiger, & Alonso, 2016), which, among a number of other improvements to *Reactable Mobile*, streamlines some of the menu-based functionality, these protocols are still only accessible away from the main "table" display. Issues exist also with regard to the limited mode of interaction for performing melodies through the main GUI of *Reactable* described above, as there is no relevance or underlying connection to pre-existing modes of interaction with acoustic instruments, which would allow for first-time users to draw upon learned cultural conventions when approaching the interface for the first time. In the case of *ScreenPlay*, as has been mentioned previously, the presence of such recognisable similarities facilitates a connection for first-time users, which they will subconsciously make regardless of any prior musical knowledge or experience.

The aim here is not to compare *Reactable* and *ScreenPlay* with regard to which is "better", but, instead, to highlight the relative strengths and weaknesses of two systems which, despite their similarities, have different aims and approaches to design. As has been said, *Reactable* is much more a digital modular synthesizer than it is an ICMS, due to the lack of two-way collaborative communication with the computer afforded to the user(s) in generating musical output from the system. It could also be argued that *Reactable* is primarily aimed at more experienced users/proficient musicians, especially with regard to taking full advantage of all the system's creative potentialities, whereas *ScreenPlay* aims to be accessible in its entirety to users/musicians of all levels. That being said, the GUI design of *Reactable* could be viewed as being more unique and visually engaging than that of *ScreenPlay* given its combination of tactile, physical control objects and screen-based interfacing, as well as the visual representation via the "table" display of waveforms and the metronome pulse; which is also exemplary of the use of audiovisual elements to improve the interactive experience as expressed by Blaine and Fels' *media* criterion for the design of collaborative interfaces (2003, p. 417). *ScreenPlay*, on the other hand, demonstrates a more unique, innovative and creative approach to the generation, development and evolution of musical ideas in collaboration with both the computer and/or other users in live, improvisatory performance or as part of the compositional process, through its implementation of generative and transformative algorithmic frameworks.

3.2 Affordances

The aforementioned connection that exists between the mode of interaction of the piano (and similar instruments) and that of *ScreenPlay* provides the basis upon which the affordances - mentioned by Blaine and Fels in their *physical interface/sensor* guidelines from the 11 criteria for collaborative musical interfaces - of *ScreenPlay*'s GUI are reliant. As has been covered in the Literature Review chapter of the thesis, Donald Norman (2004) distinguishes between *real affordances* and *perceived affordances*, as well as laying out four principles for screen interfaces in relation to their affordances. Because *ScreenPlay*'s UI is touchscreen-based its *perceived affordances* are of most importance in terms of communicating to the user(s) via the graphical display the actions required to exert influence and control over the system, due to the fact that its *real affordances* are dictated by the tangible interaction with the screen. This is in contrast to motion control gestures which would otherwise be required with a sensory-based system; the affordances of which are *real* and would be far more difficult to convey to the user(s)/audience without the addition of verbal or written instructions.

Below are listed the four principles for the design of screen interfaces outlined by Norman with the aim of clarifying how perceived affordances should be used to greater effect in allowing first-time users with no previous experience interacting with any given GUI to instinctively and intuitively understand its operational methodology, as well as the influence each of the principles has had upon the design of *ScreenPlay*'s GUI:

1. 'Follow conventional usage, both in the choice of images and the allowable interactions' (2004): the advantages of this serve to fulfil the aims outlined by Blaine and Fels with regard to audience transparency in the guidelines for *focus* in the design of collaborative musical interfaces (2003). An example within the context of HCI in music would be the graphical representations of hardware control-objects such as drum-pads, buttons, toggles, faders and rotary encoders etc., which are prevalent in touchscreen MIDI/OSC controller applications *TouchOSC* (Fischer, 2008-present), *Lemur* (Slater et al., 2011-present), *LIVKONTROL* (reference) etc. and also feature heavily in the TouchOSC layout designed as the GUI for *ScreenPlay*.

2. 'Use words to describe the desired action' (2004): where meta-level data is concerned in music-making software, clear and concise language can be used to clarify the roles and parameters of certain control objects where there may otherwise be ambiguity; especially in the case of virtual instruments and DAWs, where there are likely to be an abundance of similar control-objects. In *ScreenPlay*'s GUI certain control objects are labelled to help distinguish them from others with the same or similar appearance but differing functionality; while the provision of instructional documentation relating to both the setup and configuration of the *ScreenPlay* suite of Max for Live MIDI Devices (displayed in a pop-up help window accessible from the *ScreenPlay-CTRL* Max for Live device GUI) and how to correctly impart control over the system via the TouchOSC GUI also fall within the boundaries of this principle.
3. 'Use metaphor': Norman states that '[he] personally believe[s] that metaphors are more harmful than useful' (2004) in the context of screen-based interfaces, although, this being written more than a decade ago and before the cultural dominance of touchscreen-based interfaces in everyday life as we see today, it is somewhat outdated and no longer rings true. Both visual and verbal metaphors can be used to great effect in the UIs of screen-based (*Pixel Tune* (humbleTUNE, 2010-present), *Patatap* (Brandel, 2012-present; Brandel, 2015)) and sensory-based (*La Maison Sensible* (Lasserre, met den Ancxt, Ajima & Nagemi, 2015; Lasserre & met den Ancxt, 2015; Emory, 2015)) ICMSs intended for use by non-experts and novice musicians. In *ScreenPlay* the use of metaphorical language is paramount to accurately conveying the effects upon the musical output of the system of the topical opposition transformations.
4. 'Follow a coherent conceptual model so that once part of the interface is learned, the same principles apply to other parts' (2004): rarely does the UI of any ICMS function in an illogical manner, as such a design would severely detract from the immersiveness of the interactive experience. This is true of *ScreenPlay* and also ties in with the advantages to be gained from providing each individual user in a collaborative ICMS with a UI that is the same or similar to those allocated to the other users (where multiple UIs are present/required), as outlined by Blaine and Fels' guidelines on *player interaction* (2003).

Two design choices made in relation to refining and streamlining the overall affordances and usability of *ScreenPlay* and its GUI, rather than in direct response to Norman's four principles, were to use a horizontal "timebar" to provide a visual representation of tempo, and the omission of a global play/pause button. Tempo was originally intended to be displayed on the GUI via a pulsating LED but, unfortunately, due to fluctuating Wi-Fi signal strength and connection speed, this was not deemed a practical solution due to severe issues with instability and lag in the visual response of the LED to the pulse/tempo of the system's musical output. As a result, the flashing LED was replaced with the "timebar", which is positioned at the top of the main control/playing surface and generative/transformative algorithm control pages of the GUI (see Fig. 2 and 5), and moves from left to right over the course of a single bar. The decision not to include a play/pause button was made in order to avoid issues when configuring *ScreenPlay* as a multi-user collaborative interactive installation with a potentially large number of users each being provided with the controls to start/stop the entire musical output of the system, and the detrimental effect this could have on the immersion and enjoyment of users interacting with the system when not effectively communicating with one another; as might be the case in a public gallery installation of the system.

There is no real negative impact from this design choice on the usability of the system when configured as a single-user studio compositional tool, given that, in this scenario, the user is able to access the controls in Ableton Live for starting/stopping the playback of musical output from the system directly via their computer. When in this configuration the user is also able to switch between up to sixteen individual parts within Ableton Live and the visual feedback displayed on the GUI is updated accordingly to represent the status of the currently selected part at any given time. This includes ensuring that, if the status of a part changes while another is selected, meaning that the visual feedback to be provided to the user via the GUI upon re-selecting the part is required to be different than it was when last that part was selected, the correct changes are applied. Certain controls that are specific to each instrument/sound when in multi mode (such as key signature/scale, octave and velocity of the playing surface etc.) are also applied globally to all parts when in single mode to ensure the best interactive experience possible and that the integration of *ScreenPlay* into the compositional workflow of the user is as seamless as can be.

3.3 High Level Design Strategies and Gesture-types in the *ScreenPlay* GUI

Of Steve Benford's four 'high level design strategies for spectator interfaces' (*magical*, *expressive*, *secretive* and *suspenseful*) (2010, pp. 55-57), the most important in the case of *ScreenPlay*'s GUI design is *expressive* (clearly displaying the effects of specific manipulations). The resemblance between the mode of interaction of *ScreenPlay* and that of traditional acoustic instruments and electronic music production hardware - such as the piano, drum machines, samplers etc. - means that the connection between the input gestures of the user(s) and resulting musical output of this system in response to these actions is easy to recognise; not only for the individual(s) interacting directly with the system but also for those looking on. Despite the fact that it can often be useful in instances such as this to implement *suspenseful* tactics in order to allow audience members to see how the UI of an ICMS operates without having revealed to them the audible results of specific commands before interacting with the system themselves, the prominence of the *expressive* design strategy is simply a by-product of the necessity to provide users of *ScreenPlay* with a precise level of depth-in-control and the ability to gain proficiency and expertise through practise composing/performing with the system and, as such, is not detrimental to the level of intrigue, immersion and enjoyment experienced by players when interacting with the system. To prioritise the *suspenseful* design strategy would be to, on some level at least, risk introducing a level of uncertainty/ambiguity with regard to the audible output generated by the system in response to the control gestures/commands inputted by the user via the GUI and, in turn, make technically expressive and virtuosic performances harder to achieve. In relation to this can be extrapolated Denis Smalley's observations regarding *source-cause interaction* with regard to the spectromorphological analysis of texture/timbre (1997). Devised originally by Smalley in the context of a listener's ability to identify the sound-source and sounding gesture of a sonic artefact based on learned cultural conventions and the association of repetitions within a piece of music; here, blurring the clarity of the *source-cause interaction* would relinquish some of the precise control afforded to users by *ScreenPlay* requisite to the process of increasing technical ability with any instrument.

Closely linked to Benford's four 'high level design strategies' (2010) is his concept of there being three primary gesture types to be considered by ICMS developers with regard to the design of their system's UI; those which are *expected* (by the user), *sensed* (by the system) and *desired* (by the system-designer) (Benford et al., 2005; Benford, 2010). Due to the prominence of the *expressive* high level design strategy in the GUI of *ScreenPlay* and the reasons for this discussed above, the control gestures *expected* by the user(s) to be meaningful upon first seeing the layout of the interface are fairly clear and logical. The motions and gestures able to be detected, or *sensed*, by the interface are limited to tapping, pressing and holding (including vertical/lateral movement when holding in order to reposition sliders etc.) due to the implementation of touchscreen-based interfacing as the mode of interaction. As such, it was necessary to distinguish between the *desired* functionality of specific control-objects by creating the correct *perceived affordances* (Norman, 2004) so as to match as best as possible the outcome of any interaction between the user and GUI with their expectations leading them to perform that particular action in the first place. This is most obvious in relation to the colour coding of the topical opposition transformation sliders and their corresponding activation toggle buttons, as seen in Fig. 5. TouchOSC also offers the potential to implement motion control through the use of the device's internal accelerometer; although this has not been adopted in the design of *ScreenPlay* but does offer the potential for future development in this area. Perhaps as a way of facilitating velocity sensitivity which, at the moment, is controlled via a slider on the main control page of the GUI, due to the limitations imposed by the technical specifications of most touchscreen devices upon the variety and scope of gesture-types that can be *sensed* (including pressure/velocity). The importance to freedom of expression in musical performance and composition of velocity sensitivity/dynamics is paramount and not to provide support for this in any capacity would be a discredit to *ScreenPlay* and the goals it aims to achieve.

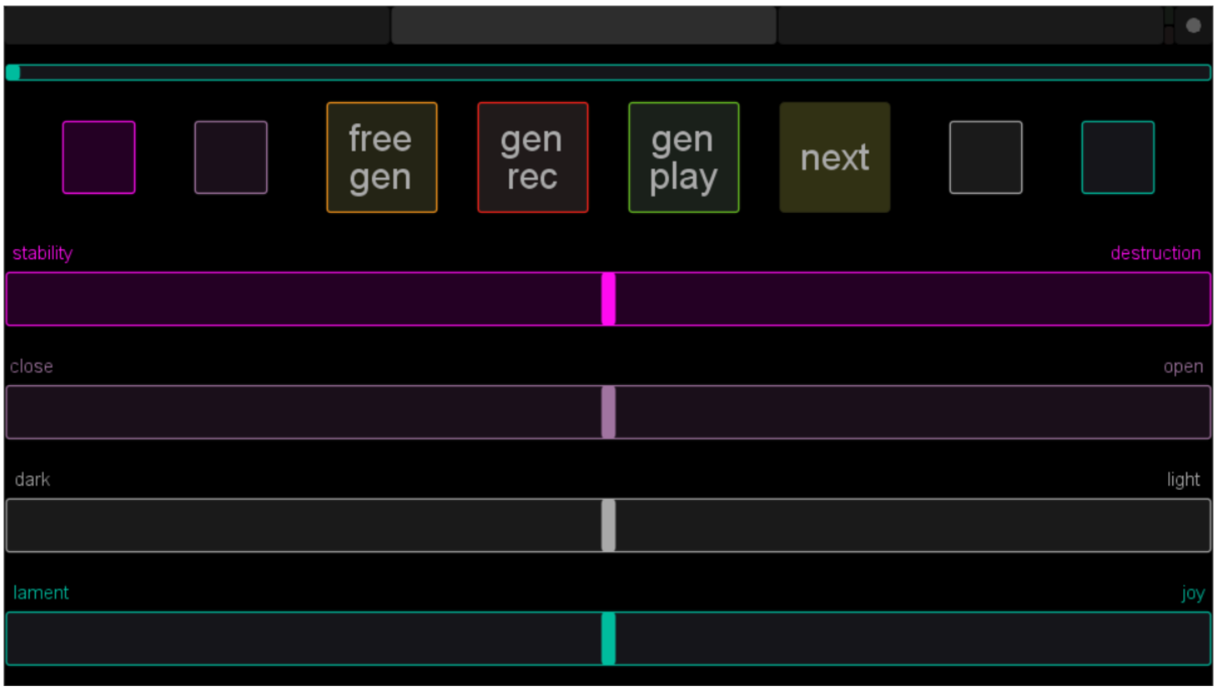
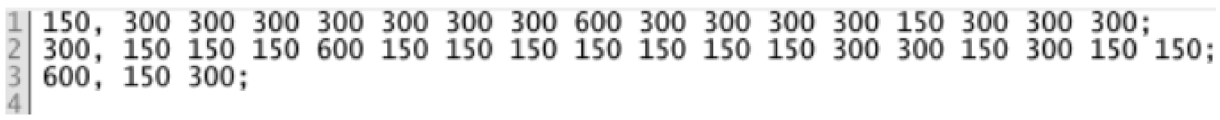


Figure 5. *ScreenPlay* GUI generative/transformative algorithm control page.

3.4 ScreenPlay-GEN

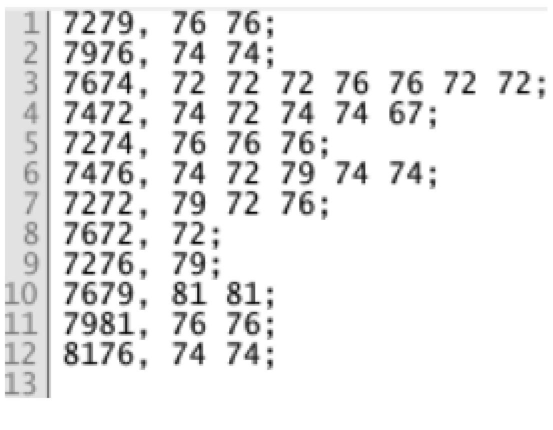
The generative algorithm implemented in *ScreenPlay* is comprised of both first and second order Markov chains. First order Markov chains are used for the generation of velocity, duration and note-on time, while a second order Markov chain is used for the generation of pitch values. This combination provides a good balance between creating variation in the generated musical output of the system but also retaining enough of the original musical character of the source material for the generated results to have a clear connection with it. Values generated by first order Markov chains are dependent only on the value immediately preceding them. Second order Markov chains account for the previous two values when calculating the next. For this reason, first order Markov chains are applied only to the secondary parameters of the source material, due to the fact that the results generated differ greatly from the source material, whereas the results generated by second order Markov chains bear more resemblance to the source material. Because the listener(s)/user(s) will most often associate the generated results with the pitch relationships of the source material it is logical to prioritise consistency and recognisability with regard to this parameter while exploiting the other parameters to create more variation in the results. Third order Markov chains (generated values dependent on three previous values) were also considered for generating pitch values but were ultimately ignored due to the fact that there is too little variation in the results generated by this type of algorithm when compared to the source material.

Before any values can be generated by a Markov chain a transition table must first be compiled containing all the values of the source material. In *ScreenPlay* these transition tables appear as a series of lists, each of which begins with a unique value (first order) or value pairing (second order) that exists in the source material followed by all the values that are preceded by that particular value or value pair. If one of the unique values is followed by the same value more than once all of these repetitions will be listed, thus increasing the probability of that value being generated due to the fact that each individual value listed after a particular unique value/pair has an equal chance of being generated. Seen below in Fig. 6 and 7 are the first and second order transition tables for duration and pitch compiled from the melodic example used in the video demonstration that accompanies this document (2'05") - also seen below in Fig. 8. As can also be seen from the video the Markovian generative algorithm in *ScreenPlay* works with polyphonic material as well as monophonic, but, for simplicity, monophonic material has been used here. (Acuma, 2010a; 2010b)



```
150, 300 300 300 300 300 300 300 600 300 300 300 300 150 300 300 300;
300, 150 150 150 600 150 150 150 150 150 150 150 300 300 150 300 150 150;
600, 150 300;
```

Figure 6. First order transition table for duration. Unique values are followed by a comma. All values that are preceded by each unique value are listed after the comma.



```
7279, 76 76;
7976, 74 74;
7674, 72 72 72 76 76 72 72;
7472, 74 72 74 74 67;
7274, 76 76 76;
7476, 74 72 79 74 74;
7272, 79 72 76;
7672, 72;
7276, 79;
7679, 81 81;
7981, 76 76;
8176, 74 74;
```

Figure 7. Second order transition table for pitch. Unique value pairings are followed by a comma. All values that are preceded by each unique value pair are listed after the comma.

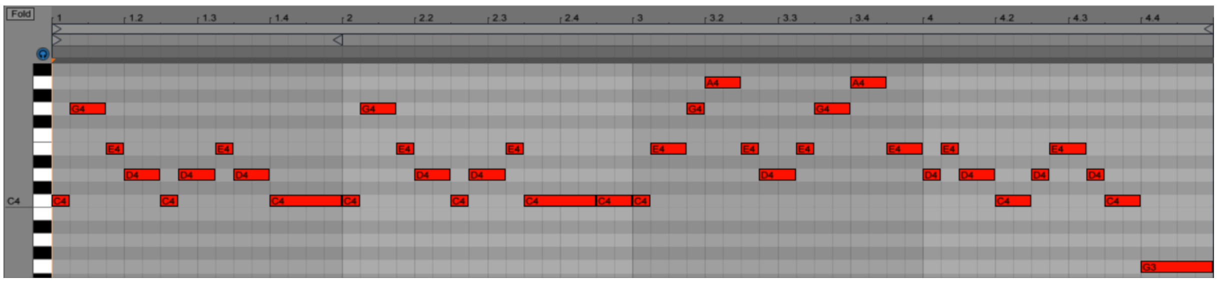


Figure 8. Clip containing original melodic line from which first and second order duration/pitch transition tables in Fig. 6 and 7 are compiled.

Duration values are collected in milliseconds and all notes in the clip are either sixteenth, eighth or quarter notes, which, at 100 bpm, translate to 150, 300 and 600 ms - as such these are the only three unique values. The very first note in the phrase is a sixteenth note and, as can be seen from the transition table, is almost always followed by an eighth note. As a result, the probability of an eighth note being generated in the event of the previous note being a sixteenth note massively outweighs the probability of either a quarter note or another sixteenth note being generated - specifically 7/8 or 87.5% as opposed to 1/16 or 6.25% each. On the contrary, a quarter note is only ever followed by both a sixteenth note and an eighth note once, meaning each has a 50% probability of being generated after a quarter note.

Pitch values are collected and stored as MIDI note numbers. The first two notes in the clip are C4 (72) and G4 (79). This note pairing only appears twice - at the beginning of bars 1 and 2 - and is followed both times by E4 (76). Likewise, the next note pair of 7976 (G4 E4) also only appears twice, due to the fact that it follows on directly from the first note pair, and only precedes the note D4 (74). The third note pair (7674/E4 D4), however, appears a number of times throughout the clip and is followed by two different notes; C4 (72) and E4 (76). C4 is the more likely of the two to be generated with a probability of 3/4 (75%) compared to 1/4 (25%) for E4. As can be seen, despite there being a far greater number of unique pitch values in the clip than are duration values, implementing a second order Markovian analysis greatly constrains the potential pitch variation and pseudo-randomness that would otherwise be introduced to the generated results should a first order Markov chain have been used.

The Markovian algorithm housed within the *ScreenPlay-GEN* Max for Live MIDI Device is controlled by the user(s) via the three toggle buttons and one trigger button positioned centrally at the top of the transformative/generative page of the TouchOSC GUI (Fig. 5), and also appear larger than the four other toggle buttons positioned at either side of them. Activating "gen rec" when a clip is currently active begins the process of collecting all the note information contained within the clip during a single entire playthrough, after which the clip is automatically disabled and the generated musical output of the algorithm begins playing back immediately. The "gen rec" button is turned off and the "gen play" button turned on to indicate as such. Making the transition between playback from the clip and generative playback an automated process helps to improve the usability of the system and ensure a fluid movement in the musical output from one note source to another. Enabling "free gen" before "gen rec" allows for notes to be freely played and recorded into the system to act as the source material for the Markovian generative algorithm, rather than being collected from an active clip. "Gen rec" must then be manually disabled by the user before activating "gen play". Touching the "next" button restarts the generation of musical material in the event that it reaches a point from where it cannot continue, due, for instance, to the pitch pair of the final two notes in the clip not appearing elsewhere in the clip and so not preceding any notes.

3.5 ScreenPlay-TRNS4M/ScreenPlay-FX

The inclusion of the topic-theory-inspired probability-based transformative algorithm in *ScreenPlay* is, in itself, a novel concept within the field of HCI in music. 'topics invoke a well-established oppositional network of meanings' (Hatten, 1994, p. 81) similar to Saussure's signifier/signified binary opposition model (1959, pp. 65-70); while the way in which they are implemented in *ScreenPlay* - as "topical oppositions" made available to the user, the repercussions of which have an inverse effect to one another on the musical output of the system - is, in itself, an example of a binary opposition. Although there is no attempt to use descriptive language as a means of simplifying for non-expert users/musicians the process of selecting key signatures/scales as suggested following the showcase of *ScreenPlay* at Salford Sonic Fusion Festival 2014 (see Case Studies chapter), due to the complexity of facilitating this without vastly reducing the number and variety of available scales/modes, the use of verbal metaphor in order to accurately portray to the user(s) the effects upon the musical output of the system of the topical opposition transformations is paramount to the success of this aspect of the ICMS.

The application of metaphor in this context resonates with two of Donald Norman's four principles for the design of screen interfaces (2004): 'Use words to describe the desired action' and 'Use metaphor'. Metaphor is fundamentally important to both topic theory and

Saussure's signifier/signified theory (1959, pp. 65-70) - with regard to the paradigmatic (substitution)/syntagmatic (combination) binary opposition model (pp. 122-125), which evolved from the theory of signifier/signified in relation to sentence structure and prioritises the appropriation of metaphor and metonymy respectively (Jakobson & Halle, 1956, pp. 76-82). As can be seen in Fig. 5, each of the four topical oppositions is displayed on the *ScreenPlay* TouchOSC GUI at opposite ends of a horizontal slider, the position of which controls the effect the transformation will have upon the musical output once activated/enabled by the corresponding toggle button to each opposition as indicated by its colour. The use of colour coding to differentiate between topical oppositions and their respective on/off switches helps to significantly improve the perceived affordances of the GUI. Whether in single or multi mode, the influence of the topical opposition transformations is part-specific - i.e. will only affect the instrument/part for which the user responsible for triggering/enabling the transformation is in control of, or that is currently selected.

The four topical opposition transformations afforded by *ScreenPlay* are: "joy-lament", "light-dark", "open-close" and "stability-destruction"; each of which affects the musical output of the system in a unique way. The impact of both the "joy-lament" and "light-dark" topical opposition transformations is directly influenced by specific topics, with the basis of "joy" being found in the *fanfare* topic, "lament" in the *pianto*, "light" in the *hunt* and "dark" in *nocturnal*. The impact of the "open-close" transformation has been designed to mimic the effects of increased and decreased proximity to a sound source as well as the size of the space in which it is sounding. A combination of reverb, delay, compression, filtering and equalisation is used to achieve this. The occupancy of sound within space is of great importance in all music and can be described by Denis Smalley's four qualifiers of spectral space: *emptiness-plenitude*, *diffuseness-concentration*, *streams-interstices*, and *overlap-crossover*. Both *emptiness-plentitude* and *diffuseness-concentration* are of particular importance in relation to the "open-close" topical opposition and can respectively be defined as 'whether the space is extensively covered and filled, or whether spectromorphologies occupy smaller areas, creating larger gaps, giving an impression of emptiness and perhaps spectral isolation', and 'whether sound is spread or dispersed throughout spectral space or whether it is concentrated or fused in regions' (1997, p. 121). The "stability-destruction" transformation draws upon my own interpretation and the literal interpretation of a destructive effect upon a sound through the use of real-time granulation to break the sound apart into smaller and smaller grains, and distortion to amplify the effect.

All three of "light-dark", "open-close" and "stability-destruction" impact the texture/timbre of the sound generated by the system. The role of texture and timbre in electronic music is paramount to the extent that, in some genres, it is equally if not more important than melodic/harmonic content. This is in contrast to the view of many topic theorists, with Agawu stating that 'melody, rhythm, and harmony are the primary parameters of tonal music while texture, timbre, and register function in a subsidiary capacity' (1991, p. 39). Although melody and harmony are of huge importance in much of electronic music, and rhythm, in particular, is a fundamental element of the vast majority of popular electronic music, it could be suggested that, when viewed in the context of the application of topic theory in electronic music, this point of view aligns with the idea that topics function best in Classical music. As has been pointed out, however, the value of melody and harmony in a great deal of electronic music is also significant and, as such, cannot be ignored.

3.5.1 Joy-Lament

The "joy-lament" topical opposition transformation is designed to alter the note values recorded into the system by the user in order to evolve and shape the melodic and rhythmic contour of monophonic musical lines and make them sound either "happier" or "sadder". Although the probability-based algorithm is designed primarily to work with monophonic melodic source material, it does also work with polyphonic harmonic source material; although the results are often unpredictable and chaotic. The reason for this is that polyphonic material is treated as separate monophonic lines, beginning with the lowest notes of each chord and moving upwards. The process by which the transformation works is to alter the pitch and duration values of the notes present within the currently/last active clip recorded into *ScreenPlay* by the user, as well as inserting additional passing notes and muting certain notes where applicable. All changes in pitch are done so in terms of an interval increase/decrease in relation to the transformed pitch of the note immediately preceding it, rather than treating each note in isolation. This allows for the melodic contour to be controlled and shaped appropriately in relation to the user-defined position of the "joy-lament" slider on the TouchOSC GUI. In terms of passing notes, their pitch is determined in relation to the transformed pitch of last note that was originally present in the clip at the moment the transformation was enabled, which helps to control the range/spread of passing notes. Likewise, the pitch of the next note originally present in the clip following on from the passing notes is transformed in relation to the transformed pitch of the note preceding the newly generated passing notes, which, again, helps to control the range of notes as well as retain the original character of the clip prior to the transformation. The range of movement in pitch from one note to another is limited to an octave in either direction, unless the interval between the original pitches of the current and previous note is greater than an octave at the moment the transformation is triggered, in which case an octave is added to the resulting interval of the pitch transformation calculation for the current note in order to help retain the original character of the melodic line.

When the transformation is first triggered a calculation occurs to ascertain whether the pitch of the current note will move up or down in relation to the previous note, the pitch of the previous note will be repeated, or the original pitch of the note will remain unchanged. When the "joy-lament" slider is positioned centrally there is an equal probability (25%) of each of the possible outcomes occurring. If the slider is positioned all the way at the "joy" end there is an 85% chance that there will be an upward movement in pitch and a 5% chance each that the pitch will move down, be repeated or remain unchanged. At the "lament" end of the scale there is a 45% chance that the note will move down in pitch, a 45% chance the pitch of the previous note will be repeated, and a 5% chance each that there will either be no change in pitch or the pitch will move up. However, the overall range of the transformation is constrained within the root notes of the chosen key/scale above/below the highest/lowest notes originally present in the clip at the time the transformation is triggered. Like adding an octave to the transformed interval between two notes in the case that the original interval between those notes was greater than an octave, this rule is imposed to help preserve the original musical character of the phrase. If a note is too high or low for the following note to move up or down in pitch respectively from its own, and it is calculated as such, then the following note is forced to move in the opposite direction, or, failing that, replaced with the highest/lowest root note of the key within the defined range. It should also be noted that the manner in which pitch transformations are conducted in relation to the interval between the current note and the one preceding it means it is entirely possible for it to be calculated that a note should move up in pitch from the previous note but in fact move down from its own original pitch and vice versa.

The specific intervals generated between notes during the transformation process are also subject to the position of the "joy-lament" slider on the TouchOSC GUI. Because all the resulting notes of the transformation are generated within the key signature/scale denoted by the key selection matrix on the GUI (and what should also be the key/scale of the melodic line upon which the transformation is occurring, in order to ensure its success), in order to guarantee for non-expert users/novice musicians a pleasurable and satisfying musical experience when interacting with the system, the potential range of generated intervals differs depending on whether a diatonic or chromatic scale is being used. More experimental/adventurous results can be yielded simply by using a chromatic scale regardless of the original key/scale of the melodic line, due to the fact that a chromatic transformation will be able recognise all available pitch values.

If the transformation is triggered using a diatonic scale there is an equal chance when the "joy-lament" slider is positioned centrally of all available intervals within the chosen scale being generated. If the slider is at the "joy" end of the scale there is a 10% chance an octave will be generated and a 45% chance each that either a fourth (perfect/augmented/diminished depending on chosen scale) or perfect fifth will be generated. If the slider is positioned at "lament" there is a 10% chance that a seventh will be generated (major/minor depending on the chosen scale) and a 45% chance each for both a second or third (again, major/minor depending on chosen scale).

If the transformation is triggered using a chromatic scale there is an equal chance of all available intervals being generated when the slider is positioned centrally. This time, if the slider is positioned at "joy", there is 5% chance each of both an octave or major seventh being generated, and a 22.5% chance each that either a minor sixth, perfect fifth, diminished fifth/augmented fourth or perfect fourth will be generated. If the slider is at "lament" there is a 5% chance each that the result of the calculation will be a major sixth or minor seventh, and a 22.5% chance each that it will be a minor second, major second, minor third or major third. The probabilities of the four outcomes with regard to both the calculation of whether the pitch moves up, down, is repeated or remains unchanged, and those of the specific intervals to be generated between notes, gradually shift as the slider is moved from one end of the scale to the other.

The "joy-lament" topical opposition transformation is directly influenced by the defining musical characteristics of specific topics, as is the "light-dark" transformation. "Lament" is based upon the topic of the *pianto*, which 'signifies distress, sorrow and lament' (Monelle, 2000, p. 11). 'the motive of a falling minor second' (Monelle, 2000, p. 17), the *pianto* '[overarches] our entire history; from the sixteenth to the twenty-first centuries' (Monelle, 2006, p. 8), is equally applicable in both vocal and instrumental music (Monelle, 2000, p. 17) and is commonplace in popular music (Monelle, 2006, p. 4) - all of which makes it ideally suited to being applied within the context of *ScreenPlay*. When first established, the *pianto* signified the action of weeping and was always accompanied by the textual representation of this. The signification later shifted (around the time of the eighteenth century) to represent sighing as opposed to weeping, before eventually coming to represent merely the emotions of 'grief, pain, regret [and] loss' associated with such actions (Monelle, 2006, p. 17). In other words, 'the sighing appoggiatura no longer means "sigh" … it has become a "system-bound expression", a lexical unit, its signification limited to the indexicality or *associations* of the sigh' (Monelle, 2006, p. 17). Monelle goes on to say:

> It is very doubtful that modern listeners recall the association of the *pianto* with actual weeping; indeed, the later assumption that this figure signified sighing, not weeping, suggests that its origin was forgotten. It is now heard with all the force of an arbitrary symbol, which in culture is the greatest force of all. (2006, p. 73)

Along with everything else that has thus far been mentioned regarding the *pianto*, this observation encapsulates its suitability for use as the basis of the "lament" transformation in *ScreenPlay*.

As has been previously explained, the "lament" transformation does not use exclusively minor second intervals but also allows for the generation of major seconds, minor/major thirds, major sixths and minor/major sevenths (some of which are available exclusively to diatonic or chromatic scale transformations). This is necessary for two reasons, the first of which being that, in order for the pitch values generated by the "lament" transformation to remain in key when using a diatonic scale, it is not always possible for a minor second to be used instead of a major second. Monelle highlights some potential issues with regard to the use of a descending major second in place of a minor second to convey lament/grief/sadness, suggesting that 'Some caution must be observed … in connecting the *pinato* with other motives in which the interval is a major second, or in which the initial minor second is part of a diatonic figure' due to the fact that 'the topical evocation is always different.' He even goes as far as to suggest that 'It is unlikely … that a descending major second would be associated with the *pianto*' (2000, p. 72). It is important to remember here that *ScreenPlay*'s transformative algorithmic framework is *inspired* by topic theory, and there is a balance to be struck between serving the needs of the system in supporting a wide range of different levels of user experience and musical proficiency in terms of providing a pleasurable, satisfying and coherent interactive musical experience at the same time as remaining true to the musical characteristics traditionally associated with the topics that have influenced its design and development. It should also be mentioned that the reason Monelle highlights the issues he perceives when using a descending major second in place of its minor counterpart with regard to the associative significance of the topic is that, as the *pianto* evolved over time, it was not uncommon for descending major seconds to be used in place of minor seconds in Classical music - or even ascending figures - when evoking pain/misery/grief/sadness.

The second reason for using a broader range of potential intervals - including thirds, sixths and sevenths - is to add variation to the results of transformations. The theoretical justification for this can, again, be found in the evolution of the *pianto***.** As time progressed there emerged a new topic based upon the *pianto* known as *Empfindsamkeit* (Monelle, 2000, p. 69). As well as the occasional inversion of the descending minor second and the use of a major second in its place, as already mentioned, was introduced the use of a descending chromatic fourth (pp. 68-69); the inclusion of all of which when expressing sorrow/grief is described by Monelle as 'the true style of *Empfindsamkeit*' (p. 70).

The reason for using thirds (both minor/major) instead of the chromatic fourth associated with *Empfindsamkeit* is twofold. First, as has already been discussed in relation to the use of major seconds, all notes generated by the "joy-lament" transformation when a diatonic scale is being used must remain inside the chosen key signature/scale, thus a chromatic fourth would not be possible regardless of the rules imposed in relation to the specific intervals available to the "lament" transformation, unless conducting a chromatic transformation. Second, even though it is specifically the descending minor second associated with the *pianto* and the use of chromatic fourths, major and minor seconds associated with *Empfindsamkeit*, generally speaking, the broad evocation of the emotions associated with these two topics (pain/sadness/grief etc.) is achieved primarily through the use of smaller intervals in general and reduced pitch variation, while larger intervals and increased pitch variation are associated with joy and happiness (Moore, 2012). The use of thirds, which are obviously smaller than fourths, helps to achieve this and also increases the contrast with the impact upon the pitch values of the musical output had by the "joy" transformation. Although large intervals, the inclusion of sixths and sevenths in the "lament" transformation is justifiable due to the small probability of the intervals being generated and the fact that, without their occasional presence, there would more often than not be very little variation in the results yielded from the transformation. The characteristic reduction of pitch variation associated with conveying sadness in music is also the reason why the "lament" transformation has a far higher probability that the transformed pitch of the previous note will be repeated than does the "joy" transformation - 45% as opposed to 5%.

Equally as likely as the previous pitch being repeated is the pitch of the current note moving down from that of the note preceding it, which is in line with the descending melodic contour of the *pianto*. The 5% chance each that there will be no change in pitch or that it will move up from that of the previous note is, again, in order to ensure there is sufficient room for variation in reaction to the "lament" transformation; although theoretical justification for the low-probability of upward movement in pitch can be found in the occasional inversion of the descending minor second of the *pianto* during the course of its evolution, as mentioned above. Regardless of the creative license taken with regard to the implementation of the *pianto* topic in the "joy-lament" topical opposition in *ScreenPlay*, its suitability as the basis for the musical impact of the "lament" transformation is summarised perfectly by Monelle in stating that 'The *pianto* … [is] seemingly so thoroughly appropriate to its evocation - somehow, the moan of the dissonant falling second expresses perfectly the idea of lament.' (2000, p. 72)

The basis of the "joy" transformation can be found in the *fanfare* topic, which is a rising triadic figure (Monelle, 2000, p. 35) most commonly used in the eighteenth century - but also the nineteenth and twentieth centuries (p. 30) - to represent 'traditional heroism, the medieval association of warfare, with a slightly theatrical and unreal flavor [sic] proper to the age' (p. 19). The evocations of heroism, triumph and joyfulness are compounded by the fact that, at the time, 'Literary warfare was dominated by idealistic heroism' (p. 35). Monelle also cites examples of the *fanfare* being used in a different context to that of the overarching *military* topic - which also encompassed the *march* - specifically in softer/gentler pieces including Mozart's Piano Concerto in B♭, K. 595 (1788-1791) and "Non più andrai" from *Figaro* (1786) (pp. 35-36), which 'helps to explain the diminutiveness, the toy-like quality, of many manifestations of fanfarism' (p. 38). This is supported by Ernst Toch who distinguishes between two different kinds of melody associated with the *fanfare*: the "masculine type" and the "feminine type" (1948, pp. 106-107).

It is this cross-compatibility of the *fanfare* topic that makes it suitable for use within the context of the modern electronic style of *ScreenPlay*. Perhaps even more important for this reason is Monelle's assertion that, very much like the topic of the *pianto*:

> the military fanfare may function associatively for a modern audience, who are sensitive [to] the slightly strutting pomp of the figure's character without realizing that it is conveyed by its origin as a military trumpet call. In this case, the topic is functioning, in the first place, through the indexicality of its original signification; the latter has been forgotten, and the signification has become *arbitrary* (2000, p. 66)

As has already been discussed, the intervals which can be generated by the "joy" transformation are primarily fourths (perfect, diminished and augmented), fifths (perfect and diminished) and minor sixths, as well as less-likely major sevenths and octaves; some of which are only possible either with diatonic or chromatic transformations. Likewise with the "lament" transformation, the decision to utilise fourths and fifths as opposed to thirds and fifths as suggested by the triadic nature of the *fanfare* topic is to increase the overall size of the intervals present in a melody given that, again, the general emotions of joy, happiness and triumph associated with the idea of heroism signified by the *fanfare* are chiefly evoked by larger intervals and more variation in pitch (Gabrielsson & Lindström, 2010, pp. 240-241). This is also why the chance of the transformed pitch of the previous note being repeated is significantly smaller for "joy" than it is for "lament" - 5% as opposed to 45%. Again, as with "lament", the decision to include octaves and major sevenths is to increase the variation in results of the transformation and can be justified due to the small probability of these intervals being generated. The overall increased interval size of the "joy" transformation helps to maximise the contrast between the two polar effects of the topical opposition. Finally, it is the aforementioned rising nature of the *fanfare* that informs the predominantly upward movement applied to intervals in the "joy" transformation (85% probability), although a downward movement in pitch, repeat of the previous transformed pitch and leaving the original pitch of a note unchanged are each allowed a probability of 5% to help increase variation in the results.

The implementation of musical characteristics associated with topics used to signify happiness/sadness is paramount to successfully conveying these emotions through the musical output of *ScreenPlay* in response to the "joy-lament" topical opposition transformation. This is primarily due to the fact that the traditional correlation between the minor/major key/scale opposition and the cultural opposition of tragic/non-tragic (Hatten, 1994, pp. 11-12) cannot be relied upon given that the selection/application of key signature/scale to the musical output of the system is done so at the discretion of the user(s). The application of the *fanfare*, in particular, in a minor key could traditionally be regarded as being in stark opposition to the character of the topic, thus nullifying its credentials as an example of *fanfare* and, therefore, its significant impact. Indeed, Agawu describes the *fanfare* as 'archetypically major-mode invariant' and even goes as far as to describe the term "minor-mode-fanfare" as 'heretical' (1991, p. 48). However, this, again, simply serves to highlight the distinction made with regard to the transformational algorithmic framework in *ScreenPlay* being *topic-theory-inspired*, and that the overall results of the "joy-lament" oppositional transformation in successfully conveying the respective emotional mood/tone of happiness/sadness when applied to the musical output of the system is justification for the creative license taken with regard to the topics of the *pianto* and *fanfare* that serve as the foundation from which the two oppositional effects have developed.

It is not only intervals between notes and overall variation in pitch that are useful when evoking happiness and sadness through music. Also of importance is the duration of individual notes as well as the speed of movement between notes, due to the fact that 'The feeling that music is progressing or moving forward in time is doubtless one of the most fundamental characteristics of musical experience' (Lippman, 1984, p. 121). When discussing characteristic traits of Romantic music in relation to the representation of lament/tragedy at the time, in reference to Poulet, Monelle states:

> Time-in-a-moment and progressive time respectively evoke lostness and struggle; the extended present of lyric time becomes a space where the remembered and imagined past is reflected, while the mobility of progressive time is a forum for individual choice and action that is ultimately doomed.
>
> Lyric time is the present, a present that is always in the present. And for the Romantic, the present is a void. "To feel that one's existence is an abyss is to feel the infinite deficiency of the present moment" [(Poulet 1949/1956 [English translation], p. 26, cited in Monelle, 2000, p. 115)]; in the present people felt a sense of lack tinged with "desire and regret". (2000, p. 115)

With progressive time being the overall sense of forward motion in music and lyric time being the internal temporality of the melody, Monelle is suggesting that the extension of lyric time - i.e. increased duration of notes in the melody and reduction of speed of movement between them - is inherent in signifying to the listener a sense of doom and regret. He later builds on this, again citing Poulet, to reiterate the sense of lament invoked through the extension of lyric time:

> The future, too, is separated, marked by doubt and clouding. Jouffroy speaks of "the torments of the human mind as it confronts the question of its destiny", and Quintet describes "the pain of the future, sleepless, piercing pain … What kills us … is having to support the weight of the future in the void of the present" [(Poulet 1949/1956 [English translation], p. 28, cited in Monelle, 2000, p. 115)]. Poulet adds: "It is as if duration had been broken in the middle and man felt his life torn from him, ahead and behind." (2000, p. 116)

Extended melodic durations are applied to the "lament" transformation to increase the sense of sorrow/sadness translated to the user(s) via the musical output of the system, first by increasing the likelihood that the duration of a note will be increased (90% chance and 10% chance the original duration of the note will remain unchanged) and then by multiplying the duration of the current note either by a factor of three (90% chance) or two (10% chance). The note following on from one whose duration has been increased will also be subject to being muted so long as the note off time of the previous note surpasses its note on time by at least a quarter note. At the "lament" end of the scale there is a 90% chance that the next note will be muted and a 10% chance that it will remain unmuted, while for "joy" these probabilities are inverted. As usual there is an equal chance of the two possible outcomes when the "joy-lament" slider is positioned centrally. The high probability for the "lament" transformation, first, of a note being increased in duration to a considerable degree and, second, of the following note being muted should the two notes overlap, effectively reduces the total number of notes in the melodic line following the application of the transformation, which helps to increase the sense of sadness conveyed through the resulting musical output of the system by reducing the speed of movement between notes in the figure.

When the "joy-lament" slider is positioned centrally there is a 33.3% chance each that the duration of a note will be increased, decreased or remain unchanged, and the same range of probabilities is applied to the multiplication/division factors of the note of 1.5, 2 or 3. The "joy" transformation inverts the probabilities expressed by the "lament" transformation, resulting in a 90% chance that there will be a reduction in note duration and a 10% chance that it will remain unchanged. This time there is 90% chance that the duration will be divided by three and a 10% chance it will be divided by two. As the slider is moved between the two poles the probabilities of all calculations gradually shift from one extreme to the other. The probability of passing notes being added between two notes as well as the number of passing notes to be added (which is also subject to the size of the gap between notes) is also increased at the "joy" end of the scale. So long as the gap between notes is at least a quarter note there is a 90% chance that at least one passing note will be added and a 10% chance that no passing notes will be added. For "lament" the probabilities are reversed and there is an equal chance of both with the slider positioned centrally. A gap less than a half note only ever allows for a single passing note to be added, while a gap of greater than six beats can accommodate up to eight passing notes. The maximum number of passing notes available gradually increases with the size of gap, with the minimum remaining always as one passing note. With the slider positioned at "joy" it is highly likely that a larger number of passing notes will be added and at "lament" a lower number of passing notes is the likelier outcome. There are only ever, at most, two available durations for each passing note generated, which are subject both to the size of gap and number of passing notes being inserted. At "joy" it is highly likely that the shorter of the two durations will be applied and at "lament" the longer. Passing notes inserted into a gap of more than six beats are quantized to eighth notes, otherwise they are quantized to sixteenth notes. The decision to increase the likelihood of shortening note durations and inserting passing notes for the "joy" transformation has not been done only in opposition to the characteristics with regard to conveying sadness/lament through music. The increased pitch variation and speed of movement between notes that come as a result of the shortening of notes and addition of passing notes also links with the fact that the 'fanfare is essentially a rhythmicized arpeggio' (Agawu, 1991, p. 48). An additional advantage of this in the context of *ScreenPlay* is that the rhythmic characteristics of the *fanfare* help to increase its mobility in terms of style and genre.

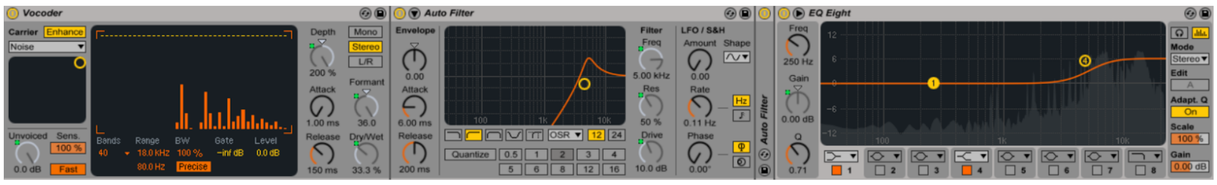
3.5.2 Light-Dark

Closely related to the topic of the *fanfare* is the topic of the *hunt*, which serves as the inspiration for the "light" transformation in the "light-dark" topical opposition. Having signified the morning time during the Classical era and beyond, due to the fact that 'the courtly hunt of the period took place during the morning, a fact we can ascertain from social history' (Monelle, 2006, p. 3), the *hunt* was the logical candidate from which to draw inspiration for the "light" transformation. Itself a type of fanfare - hence the topical association - predominantly in a 6/8 meter (p. 82), it is not the melodic/rhythmic contour or organisation of notes inherent to the topic that are of interest in the context of *ScreenPlay*. The subject of the "light-dark" topical opposition is the texture/timbre of the musical output from *ScreenPlay* and so it is these characteristics of the *hunt* that have been emulated in the effect of the "light" transformation. The French and German hunting horns - the traditional hunting calls and fanfares of which shaped the topic of the *hunt* (Monelle, 2006) - were usually of different sizes, with the French hunting horn ordinarily being considerably larger than the German - although there are examples of different sizes/variations of both. Because of this the range/register of the two instruments is markedly different; the French being predominantly lower and the German being higher. However, the texture/timbre of the instruments themselves and the calls/fanfares played on both, as a result of their dynamics, were very similar. Both produce a very bright, loud and forthright sound with a large amount of presence throughout the midrange of the frequency spectrum, as well as the ability to reach more towards the high end of the spectrum when played at a higher register that is typical of many brass instruments. Examples of both are easy to find online (Lepoultier, 2014; Lodge, 2012).

The way this has been translated to the textural/timbral impact of the "light" transformation in *ScreenPlay* is through the use of a high-pass filter, EQ and a vocoder to mix a small amount of noise with the dry signal in order to accentuate the high-end of the sound. All of the effects used in *ScreenPlay-FX*, not only for the "light-dark" topical opposition transformation but also "stability-destruction" and "open-close", are included as standard with Ableton Live. The reason for this is to adhere with one of the core design principles of *ScreenPlay*, to utilise only existing technology so as to maximise the potential impact of the system both within the fields of human-computer interaction in music and electronic music production. Using third-party plug-ins would not have been in opposition to this ethos but would have had a seriously harmful effect on the potential impact of the system in the area of electronic music production by vastly reducing the number of potential users who would be able to meet the software requirements of the system in order to run it as intended. However, the choice to use an Ableton Live Audio Effect Rack to house all of the plug-in devices, with *ScreenPlay-TRNS4M* only having to translate the incoming control data received from each of the topical opposition sliders on the TouchOSC GUI to a single macro control from where all effects parameters are mapped using the Audio Effect Rack internal mapping protocol, gives users the opportunity to easily add/remove any effects they like as well as alter to specific parameter mappings within the default effects assigned to each of the topical oppositions.

When the "light-dark" slider is positioned all the way at "light" the cutoff frequency of the high-pass filter rises to 5 kHz, its resonance to 50% and drive to 10%. The high-pass filter is followed by a high shelving boost in the EQ of 6 dB at 5 kHz. These settings combine to remove the majority of the low-end frequency content from the incoming signal at the same time as boosting the resonant frequencies in the upper midrange and highs. The high-pass filter employs a gradual 12 dB per octave curve to ensure the lower midrange and bottom end of the frequency spectrum is rolled off as smoothly as possible so as to make for a more natural-sounding result to the transformation. Likewise, the resonance of the high-shelving boost in the EQ remains at its default value of 0.71 to result in a similar characteristic quality to the sound in the upper midrange and high-end. The gentle slope of the shelving boost means that the 6 dB boost applied at 5 kHz does not fully impact the sound until around 10 kHz and, as such, primarily affects only the high-end of the frequency spectrum. The increase to 50% applied to the resonance of the high-pass filter is utilised to compensate for this by providing a fairly substantial bump in the upper midrange of the frequency spectrum from around 5/6 - 7/8 kHz. Drive is a parameter available in certain analogue emulation configurations of Ableton Live's Auto Filter device. A 10 dB boost to drive has been included in the "light" transformation due to its ability to add light distortion - and therefore an increase in high-end frequency content - to the signal outputted from the filter by increasing the input gain of the dry signal as it first enters the filter. Located between the high-pass filter and EQ can also be found a low-pass filter that is used for the "dark" transformation. For the "light" transformation, however, the cutoff frequency is set to the maximum level of 19.9 kHz with a resonance level of 0% and a drive level of 0 dB so as to ensure the filter has no audible effect on musical output of the system.

At the very beginning of the effects chain utilised for the "light" transformation is a vocoder, which is used to mix some white noise in with the dry input signal in order to further increase the high-end presence of the outputted sound. The position of the vocoder at the beginning of the effects chain was chosen so as to allow the noise signal to be subject to both high and low-pass filters when the "light-dark" topical opposition slider is placed in various positions along the scale. For the "light" transformation the dry/wet level of the vocoder is set to 33%, meaning that a substantial but not overpowering level of noise is mixed with the input signal. In addition to the increase in the dry/wet signal the values for depth and formant are both increased to their maximum levels of 200% and 36.0 respectively. Increasing the depth level causes the noise carrier signal to respond only to high peaks in the modulator signal's amplitude and thus tones down the effect of the vocoder when the "light" transformation is being applied to quieter/softer sounds, without having an adverse effect on the results when the transformation is applied to louder, more up-front sounds. By increasing the formant level the frequencies of the bandpass filters used to analyse the incoming modulator signal are increased and detection of higher frequency content is prioritised over the bottom end, thus further shifting the balance of the resulting musical output from the system towards the midrange/high-end of the frequency spectrum (DeSantis et al., 2016, p. 388). Finally, the unvoiced noise generator is increased to its maximum level of 0.0 dB. Ordinarily 'used to resynthesize portions of the modulator signal that are pitchless, such as "f" and "s" sounds' (p. 387), the inclusion in the "light" transformation of the unvoiced noise generator helps give the noise level more presence in the wet signal due to the doubling up effect of the voicing. This approach is more effective in the context of the "light-dark" transformation on the whole than simply increasing the gain level of the main carrier signal as the "light-dark" slider on the GUI is moved more towards "light" (and decreasing it as the slider moves more towards "dark"), due to the fact that, by introducing an additional dimension of control over the noise level in the musical output through the balancing of two separate noise generators, a more consistent noise level can be achieved throughout the entire range of the transformational effect of the topical opposition whilst still allowing for the presence of noise in the output signal to be enhanced at the "light" end of the spectrum. This is important because, as will be explained below, some noise is applied also in the "dark" transformation and increasing/decreasing both the dry/wet balance and the gain level of the primary carrier signal would result in little audible noise in the musical output of the system when the "light-dark" slider is positioned more towards the "dark" end of the spectrum.



Carrier: Noise, Enhance
Unvoiced: 0.0 dB
Sens.: 100 %, Fast
Bands: 40
Range: 18.0 kHz, 80.0 Hz
BW: 100 %, Precise
Gate: −inf dB
Level: 0.0 dB
Depth: 200 %
Stereo
Attack: 1.00 ms
Release: 150 ms
Formant: 36.0
Dry/Wet: 33.3 %

Envelope: 0.00
Attack: 6.00 ms
Release: 200 ms
OSR, 12
Filter Freq: 5.00 kHz
Res: 50 %
Drive: 10.0 dB
LFO / S&H Amount: 0.00
Rate: 0.11 Hz
Phase: 0.00°

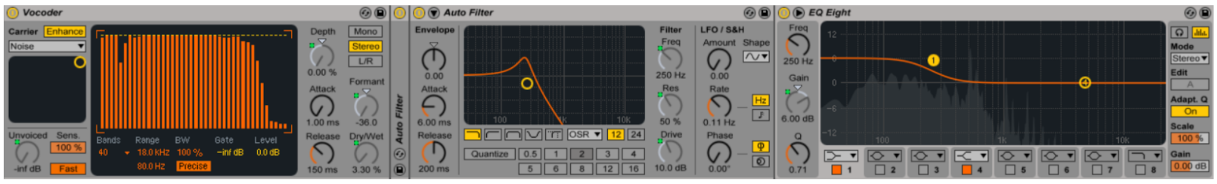
Freq: 250 Hz
Gain: 0.00 dB
Q: 0.71
Mode: Stereo
Edit: A
Adapt. Q: On
Scale: 100 %
Gain: 0.00 dB

Figure 9. "Light" transformation effects chain: Vocoder, high-pass filter, EQ.

The textural/timbral impact of the "dark" transformation is founded in those qualities of *nocturnal* music, which originated in the establishment of the *horn of nocturnal mystery* topic. '[taking] on some of the associations of the hunt, especially the mysterious depth of the woodland, but [abandoning] others' (Monelle, 2006, p. 91), the *horn of nocturnal mystery* was a 'solo horn in the nineteenth century, playing a fragrant cantilena with a soft accompaniment' (p. 106). Monelle goes on to reaffirm the link between the *horn of nocturnal mystery* and nature/woodland when he states that 'the frequent manifestation of the mysterious, poetic horn in Romantic music has much to do with its evocations of magic, darkness, and the forest' (p. 107). Examples of the link between *nocturnal* music and the woodland can still be found today; even in popular electronic music, as seen in the song *Night Falls* by Booka Shade (2006), which serves to highlight the suitability of *nocturnal* characteristics in the context of *ScreenPlay*.

Despite the significations of nocturnal woodland in *Night Falls* this does not figure in impact of "dark". Instead, like with "light", the focus is specifically on the textural/timbral characteristics of *nocturnal* music and the *horn of nocturnal mystery* topic. The majority of sounds used throughout *Night Falls* are lacking in high frequency content and, predominantly, play notes, lines and progressions that are low in register. There are occasional interjections of higher register notes/lines and sounds with more high-end frequency content that draw focus but, for the most part, even the "lead" sounds, such as the "bubbly" arpeggiated synthesizer in the introduction (0'22") and breakdown (2'55"), and the arpeggiated bell pad that first enters during the breakdown at 3'18" before continuing on into the second verse to hold the melodic focus of the piece prior to the re-entry of the guitar-like arpeggiated lead at 3'49" (first heard 1'33") that holds the melodic focus for much of the piece, are subject to the application of a high-pass filter and lack in any significant top-end frequency content. The guitar-like arpeggiated lead itself, despite having more high-end presence than the previous two sounds, is still substantially lacking in top-end, and even high frequency percussion sounds like the high-hats have a minimal, unintrusive quality to their tone and are positioned relatively low in the overall mix. In fact, during the first verse section of the piece, the melodic focus is shared by two distinct bass sounds which interweave and, at times, play simultaneously. A fairly unique compositional choice within any style/genre of music, this continues up until the entry of the higher-register guitar-like arpeggiated lead at 1'33", after which the arpeggiated bass sound drops out (for the most part).

The same textural/timbral and registral characteristics are inherent to *nocturnal* music and the *horn of nocturnal mystery* topic. The impact upon the tone of the instrument of playing the horn softly is almost like applying a low-pass filter to the sound; effectively rolling off/smoothing out a large amount of the higher frequency content that ordinarily creates the harsh, bright, buzzing textural/timbral characteristics in the sound produced by the instrument. Likewise, the accompaniment to the solo horn was not only texturally soft but also generally low in register; again similar to the overarching instrumentation and compositional style of *Night Falls*. These primary characteristics are translated to the "dark" transformation by way of low-pass filtering, in order to remove high frequency content and smooth out the sound, and enhancement of the low frequency content/low resonant frequencies of the signal. The cutoff frequency of the low-pass filter is reduced to 250 Hz and, as with the high-pass filter used for the "light" transformation, a gradual curve of 12 dB per octave is used to give the resulting musical output as natural a feeling as possible. Again, as with the high-pass filter used for the "light" transformation, the resonance is boosted to 50% in order to generate a bump in the frequency spectrum from around 150-300 Hz, and compensate for the lack of gain applied to those frequencies by the EQ low-shelving boost of 6 dB at 250 Hz using the default resonance setting of 0.71. Boosting this area of the frequency range not only helps to shift the overall balance of the sound outputted by the system to a lower register but also reduces the clarity of the signal by increasing the "muddiness" of the sound - an idea that resonates with that of one's vision being impaired when in the dark. This blurring of clarity is increased further still by retaining a small amount of noise generated by the vocoder in the musical output of the system. The level of the unvoiced noise generator is reduced to -inf dB to ensure there is no doubling up of noise in the final signal, while the formant level is also reduced to its minimum value of -36.0 in order to shift the frequency range of the bandpass filters used to detect the incoming modulator signal as low as possible and emphasise the bottom end of the frequency range. The dry/wet balance is brought down to 3.30% but, to compensate for the low level in the mix of the noise carrier signal, the depth level is also reduced to 0%; meaning that the amplitude envelope of the incoming modulator signal is entirely ignored by the device and the noise generated is done so at a consistent level, regardless of the musical character/dynamics of the source material/sound upon which the transformation is being carried out. As before, with the settings of the low-pass filter in the "light" transformation, the high-pass filter used in the "light" transformation is effectively bypassed by reducing the cutoff frequency, resonance and drive to their minimum values of 26 Hz, 0% and 0.00 dB. Moving the "light-dark" slider on the TouchOSC GUI between the two topical oppositions causes the values of all parameters to shift from one extreme to the other, with a central position denoting the median value for each of the mapped parameters. The same is true of both the "open-close" and "stability-destruction" topical opposition transformations. It should also be noted that the effectiveness of the "light-dark" oppositional transformation is, of course, subject to sound selection and the musical source material upon which the transformation is applied and how this relates to the position of the "light-dark" slider on the TouchOSC GUI. Lower sounds and notes naturally contain more low-end frequency content, while the opposite is true of higher sounds and notes.


Vocoder
Carrier Enhance
Noise
Unvoiced -inf dB
Sens. 100 %
Fast
Bands 40
Range 18.0 kHz 80.0 Hz
BW 100 %
Precise
Gate –inf dB
Level 0.0 dB
Depth 0.00 %
Attack 1.00 ms
Release 150 ms
Mono
Stereo
L/R
Formant -36.0
Dry/Wet 3.30 %

Auto Filter
Envelope 0.00
Attack 6.00 ms
Release 200 ms
OSR
12
24
Quantize 0.5 1 2 3 4 5 6 8 12 16
Filter
Freq 250 Hz
Res 50 %
Drive 10.0 dB
LFO / S&H
Amount 0.00
Shape
Rate 0.11 Hz
Hz
Phase 0.00°

EQ Eight
Freq 250 Hz
Gain 6.00 dB
Q 0.71
1 2 3 4 5 6 7 8
Mode Stereo
Edit A
Adapt. Q On
Scale 100 %
Gain 0.00 dB

Figure 10. "Dark" transformation effects chain: Vocoder, low-pass filter, EQ.

3.5.3 Open-Close

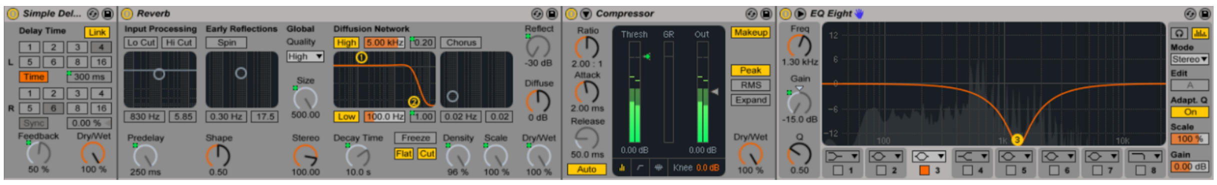
As has previously been mentioned the "open-close" and "stability-destruction" topical opposition transformations are not rooted in the musical and textural/timbral characteristics of specific topics but, instead, are both an interpretive expression of and intended to mimic the textural/timbral qualities inherent to their respective significations. The "open-close" oppositional transformation is based upon the idea of sound in space, with "open" representing large, wide-open spaces and "close" representing confined, enclosed spaces. As such, the effects chain used for these transformations consists of a delay, reverb, compressor and EQ in that order. The mapping inside the delay effect is fairly simple with only two parameters accounted for; feedback, and time. Used in place of the dry/wet balance - which remains static at 100% - due to the fact that very little of the effect can be heard in the wet signal until the dry/wet balance reaches the upper end of its range (especially with quieter/softer/slow attack sounds), the feedback setting, which is responsible for feeding the output of the delay back into its input in order to layer the delay tails on top of one another and "thicken" the effect, is increased to 50% for "open" and reduced to 0% for "close". When paired with an extremely short delay time, which is reduced to its minimum value of 1 ms for "close", the effect is all but inaudible and, in practice, rarely heard. For "open" the delay time is increased to its maximum value of 300 ms.

The mapping inside the reverb effect is much more complex. First, the predelay (time after which the effect responds upon receiving an input signal) is increased to its maximum value of 250 ms for "open" - giving the impression the walls of the space in which the sound is being heard are very far from the source - and decreased to its minimum value of 0.5 ms for "close"; thus having the opposite effect. Next, the reverb size is increased to the maximum setting of 500 for "open" and reduced to the minimum of 0.22 for "close"; effectively altering the overall dimensions in all directions of the space within which the instrument is sounding. Following this the decay time (time it takes for the reverb tails to die away after first responding to the input signal) is increased to 10 seconds for "open" and reduced to 1 second for "close" - again, the longer the decay time, the larger the space and vice versa. The reason for only reducing the decay time to as much as 1 second for "close" is that, even in an extremely small space, there is always some reverberation in response to a sound source and this translates that effect to the results of the transformation.

High and low shelving filters are then applied to the diffusion network of the reverb tail at 5 kHz and 100 Hz respectively to further sculpt the sound. For "open", the high shelving filter is reduced to its minimum value of 0.2 and the low shelving filter is increased to its maximum value of 100. Removing/attenuating the high frequency content of the reverb tail whilst simultaneously boosting the low frequency content reduces the definition and clarity of the sound, as is the case with reverberations from a sound in a very large space. For "close", the inverse settings are applied to attain the opposite effect. Next, the density of the diffusion network is increased to the maximum and minimum settings for "open" and "close" of 96% and 0.1% respectively. A high density value results in a thickening of the reverb effect and therefore both makes the effect more audible and obscures the clarity/definition of the sound source in the musical output of the system. Likewise, the diffusion scale is also increased to the maximum value of 100% for "open" and the minimum of 5% for "close". Responsible for controlling the coarseness of the diffusion network, a higher value serves to reduce the brightness of the sound and vice versa. The reflect parameter controls the gain level of early reflections in the diffusion network. As such, this is reduced to its minimum value of -30 dB for "open" and increased to its maximum of 2 dB for "close" in order to reflect the respective masking/prominence of early reflections from a sound when heard in a large/small space. Finally, the dry/wet balance of the reverb is increased to 100% for "open" and reduced to 10% for close. As with decay time, the reason for not reducing the value completely to 0% for "close" is due to the fact that, even when heard in an extremely small space, the reflections from a sound source will still produce a minute amount of audible reverberation. Retaining a small amount of reverb in the output signal for "close" results in a more natural sound than would be the case if it were to be completely dry. In the real world such a sound could only come close to being replicated in an acoustically treated environment such as an anechoic chamber.

The final two effects in the "open-close" topical opposition transformation chain are a compressor and an EQ, which work together to convey not the size of the space in which the instrument is sounding but the proximity of the listener to the sound source. In order to increase the proximity for "close" the threshold of the compressor is reduced to -20 dB, thus "squashing" the sound and increasing the volume of quieter transients in the input signal in relation to the louder ones. For "open", the threshold is increased to 0 dB; thus nullifying the effect and removing the need to alter the dry/wet balance, which remains static at 100%. The attack, release and ratio settings - as well as the threshold being reduced only as far as -20 dB for "close" - all remain static and retain their default, relatively sensible levels of 2:1, 2 ms and 50 ms. This allows for the transformative effect to accommodate a large range of potential sounds and musical material in terms of texture/timbre, note density/frequency, register and amplitude whilst also helping to retain transients in the input signal when present by not compressing the sound too much. The openness of using Ableton Live's Audio Effect Rack to house all of the devices for the topical opposition transformations does, however, invite users to create extra mappings within any of the effects or alter the default settings of existing parameter mappings as they wish in order to better suit the musical context in which the transformations are being applied.

Following the compressor is an EQ, which implements a bell curve filter with a modest resonance level of 0.5 at 1.3 kHz. For "open", the gain level of the curve is reduced to the minimum value of -15 dB in order to attenuate the midrange frequencies in the signal and reduce clarity, definition and harshness in the sound; thus giving the effect of the sound source being farther from the listener. In contrast, the midrange frequencies are boosted to the maximum level of 15 dB for "close" in order to convey the opposite effect in the musical output of the system. The reason the compressor is positioned prior to the EQ in the effects chain is so that the increased level of the midrange frequencies for the "close" transformation are not reduced/"squashed" by the action of the compressor or, for "open", the attenuated midrange frequencies are not brought back up in level to compensate for the their comparatively low gain in relation to the rest of the frequency range - the impact of which would be to nullify the equalisation effect across the range of the "open-close" topical opposition transformation.



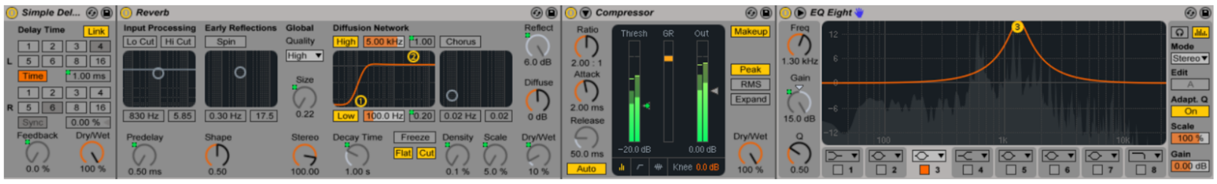
Simple Del... — Delay Time: Link; L: Time 300 ms; R: Sync 0.00 %; Feedback 50 %; Dry/Wet 100 %

Reverb — Input Processing: Lo Cut, Hi Cut, 830 Hz, 5.85; Early Reflections: Spin, 0.30 Hz, 17.5; Global Quality: High; Size 500.00; Diffusion Network: High 5.00 kHz 0.20, Low 100.0 Hz 1.00; Chorus 0.02 Hz 0.02; Reflect -30 dB; Diffuse 0 dB; Predelay 250 ms; Shape 0.50; Stereo 100.00; Decay Time 10.0 s; Freeze; Flat; Cut; Density 96 %; Scale 100 %; Dry/Wet 100 %

Compressor — Ratio 2.00 : 1; Attack 2.00 ms; Release 50.0 ms; Auto; Thresh 0.00 dB; GR; Out 0.00 dB; Knee 0.0 dB; Makeup; Peak; RMS; Expand; Dry/Wet 100 %

EQ Eight — Freq 1.30 kHz; Gain -15.0 dB; Q 0.50; Mode Stereo; Edit A; Adapt. Q On; Scale 100 %; Gain 0.00 dB

Figure 11. "Open" transformation effects chain: delay, reverb, compressor, EQ.



Simple Del... — Delay Time: Link; L: Time 1.00 ms; R: Sync 0.00 %; Feedback 0.0 %; Dry/Wet 100 %

Reverb — Input Processing: Lo Cut, Hi Cut, 830 Hz, 5.85; Early Reflections: Spin, 0.30 Hz, 17.5; Global Quality: High; Size 0.22; Diffusion Network: High 5.00 kHz 1.00, Low 100.0 Hz 0.20; Chorus 0.02 Hz 0.02; Reflect 6.0 dB; Diffuse 0 dB; Predelay 0.50 ms; Shape 0.50; Stereo 100.00; Decay Time 1.00 s; Freeze; Flat; Cut; Density 0.1 %; Scale 5.0 %; Dry/Wet 10 %

Compressor — Ratio 2.00 : 1; Attack 2.00 ms; Release 50.0 ms; Auto; Thresh −20.0 dB; GR; Out 0.00 dB; Knee 0.0 dB; Makeup; Peak; RMS; Expand; Dry/Wet 100 %

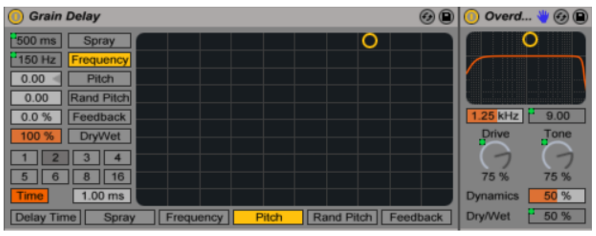
EQ Eight — Freq 1.30 kHz; Gain 15.0 dB; Q 0.50; Mode Stereo; Edit A; Adapt. Q On; Scale 100 %; Gain 0.00 dB

Figure 12. "Close" transformation effects chain: delay, reverb, compressor, EQ.

3.5.4 Stability-Destruction

The "stability-destruction" oppositional transformation, again, has a very literal impact upon the musical output of the system by "destroying" the sound through the use of granulation and distortion; the destructive impact of which increases as the "stability-destruction" slider on the TouchOSC GUI is moved away from "stability" and towards "destruction". Unlike traditional granulation of a predetermined slice of audio associated with granular synthesis, the granulation of the musical output of *ScreenPlay* happens in real time and so is carried out by Ableton Live's Grain Delay effect. As with the delay effect used in the "open-close" transformation, rather than adjusting the dry/wet balance of Grain Delay to increase the destructive impact of the effect, the spray of the delay is instead used with dry/wet remaining static at 100%. This is due to the fact that the wet signal can sometimes be hard to detect in a sound right up until the dry/wet balance reaches the higher end of its range. With the "stability-destruction" slider positioned at "stability" there is little to no audible effect of the granulation in the musical output of the system. As such, the spray - which is responsible for generating random alterations to the delay time within the range denoted by its value - is set to the minimum of 0 ms, meaning there can be no changes in delay time. Because, with the dry/wet balance remaining static at 100%, only the processed signal and none of the dry, unprocessed signal is outputted from the effect, a spray setting of 0 ms is inaudible and heard simply as the dry signal, except for when the device is activated/deactivated and a slight momentary shift in pitch can sometimes be heard. At "destruction" the spray level is set to its maximum setting of 500 ms, thus breaking apart the sound into many individual grains delayed from the original, unprocessed input signal by anything from 0-500 ms. The frequency value alters the size and duration of individual grains, meaning that smaller values result in larger grains and more coherence in the musical output of the system and larger values result in smaller grains and more chaotic results. The frequency is set to the minimum value of 1 Hz for "stability" and the maximum of 150 Hz for "destruction".

Following the Grain Delay is Ableton Live's Overdrive distortion effect, which helps to further emphasise the destructive effect of the granulation carried out by the Grain Delay. For "stability", the drive, tone, dry/wet balance and bandpass filter bandwidth are all reduced to their minimum settings of 0% for the first three and 0.5 for the latter. The drive - amount of distortion applied to the signal - and tone - boosts the high-end of the frequency spectrum following the application of distortion - are both increased to 75% for "destruction". The dry/wet balance is increased to 50%. The bandwidth of the bandpass filter, which shapes the incoming signal prior to the application of distortion, is increased to its maximum value of 9. The frequency of the bandpass filter remains at 1.25 kHz regardless of the position of the "stability-destruction" slider. At the very end of the combined effects chain for all three textural/timbral topical opposition transformations is a limiter, which is always active and helps to control the overall level of the the musical output from the system. This is particularly important when applying the "destruction" transformation to loud source material.


Grain Delay
500 ms Spray
150 Hz Frequency
0.00 Pitch
0.00 Rand Pitch
0.0 % Feedback
100 % DryWet
1 2 3 4
5 6 8 16
Time 1.00 ms
Delay Time | Spray | Frequency | Pitch | Rand Pitch | Feedback

Overd...
1.25 kHz 9.00
Drive 75 %
Tone 75 %
Dynamics 50 %
Dry/Wet 50 %

Figure 13. "Destruction" transformation effects chain: Grain Delay, Overdrive.

All of the devices used for the three textural/timbral topical opposition transformations are toggled on/off when the corresponding toggle buttons on TouchOSC GUI to each of the three transformations are activated/deactivated. Moving the topical opposition sliders on the GUI while a transformation is currently active will alter the musical output of the system in real time. All parameter mapping values for specific transformations have been decided upon after extensive experimentation in relation to their audible effects upon source material occupying different registers across the frequency spectrum and with a wide range of sounds with differing textures/timbres in order to provide the best possible results. However, as has been previously mentioned, any of the parameter mappings can be freely edited by the user and new mappings or even new effects can be added in order to tailor the response of the system to the three textural/timbral topical opposition transformations.

The overall range of transformations included in *ScreenPlay*, as well as the specific topical oppositions used, were also subject to much consideration in order to ensure the transformative aspect of the system was capable of producing a wide variation of musical results both tonally and texturally/timbrally, without becoming overly complicated so as to remove the intuitiveness of the system for users of different experience levels. The support for both textural/timbral and tonal transformations is paramount due to the fact that the former is of such consequence in electronic music, while the latter is often thought of as the 'most important parameter' of music (Agawu, 1991, p. 37). In reiteration of a point made earlier in this chapter, the effectiveness of utilising aspects of topic theory in the implementation of the transformative algorithmic framework in *ScreenPlay* is largely down to the fact that the specific signification of topics 'is very much a contextual matter.' (Agawu, 1991, p. 50)

4. Case Studies

Throughout the course of *ScreenPlay*'s development lifecycle there have been a number of public and private focus-group-esque showcases of the system in order to gather feedback from a wide range of users in terms of demographic, musical proficiency and experience with interactive music technology. The collection of this information has been vital in helping to achieve the fundamental design-goals of *ScreenPlay* - in particular to create a unique and innovative, all-encompassing ICMS that is accessible, immersive and captivating for both novice and experienced users/musicians alike - by focussing the attention of the development process on evolving and refining specific aspects of the system's internal programming and GUI design highlighted by the users as having potential for improvement. Ethical approval was granted on the grounds that the details of why the information was being collected and how it would be used in the context of the research project were disclosed to participants, and that their written consent was collected and stored in a secure location accessible only to the researcher before, ultimately, being destroyed upon completion of the PhD. In accordance with this the consent forms were stored in a personal locker/filing cabinet accessible only by the researcher located in the postgraduate research study room in Adelphi Building at the University of Salford, which is accessible only to postgraduate researchers in possession of the required key code to gain access to the entrance of the room.

4.1 Salford Sonic Fusion Festival 2014

The first public showcase of *ScreenPlay* came in April 2014 at Salford Sonic Fusion Festival. At this time the central patch of the system was programmed in Max/MSP and had not yet been migrated to Max for Live, while the TouchOSC GUIs were designed to be hosted via four separate touchscreen tables; the specifications of which consisted of a Mac Mini, running Windows 7, connected to a very large touch-sensitive flat-screen television monitor (the exact technical specifications of which are unknown) positioned horizontally within the bodywork/casing of the table. Despite being well received and fairly successful in its operation/functionality for a first exhibition within six months of the research project commencing, as well as providing a vital opportunity to gather the insight and opinions of first-time users so early on in the research timeline, running the system in this technical configuration led to a number of issues that severely impacted upon the overall experience of interacting with it and, ultimately, led to the migration of the central patch to Max for Live and the employment of tablets to host the TouchOSC GUIs.

Of the various problems encountered there were two major complications, the first of which was the result of *ScreenPlay*'s design as a "Computer Network Music" system (Early Computer Network Ensembles, n.d.; Brown, n.d.) employing Wi-Fi to form the connection between the GUIs and central Max patch, and having to run TouchOSC - a native Android/iOS application - via the Windows 7 operating system. This meant having to host TouchOSC within the third-party Android emulator *BlueStacks* (Sharma, 2009-present). Despite marginally slower operating speeds than would have been the case running TouchOSC from a native Android/iOS device, the addition of BlueStacks to the system specification caused a breakdown in the two-way communication between the Max patch and TouchOSC GUIs; allowing only for the transmission of data out of TouchOSC and into Max/MSP, not vice versa. This meant the GUIs were not able to actively display the resulting changes to global parameters of the system made by other users via separate interfaces, as well as having to implement a makeshift audible metronome instead of the visual metronome originally programmed into the system, which, again, was not ideal and took away from the fluidity of the experience of interacting with the system and other users.

The reasoning behind this issue can be explained by the requirement of TouchOSC to utilise a Wi-Fi connection in order to transmit and receive MIDI/OSC data, and therefore the individual IP addresses of each of the devices connected to the network between which the information is transferred/shared. Because MIDI/OSC data being sent out of TouchOSC was being received by a [udpreceive] object within the Max patch hosted by the central computer system within the shared network, all that was required to make the connection between each of the "touchtables" hosting the GUIs and the Max patch was the IP address of the central computer and to match the UDP port number assigned to [udpreceive] with that designated in TouchOSC through which to transmit outgoing OSC data. Because the transmission of MIDI data was unidirectional (from the TouchOSC GUIs to the Max patch) there was no apparent issue, while the outgoing OSC data-stream also reached its destination unhindered. However, the problems encountered when transmitting OSC data back out of the Max patch and into TouchOSC came as a result of having to assign to the [udpsend] object not only the corresponding UDP port set to receive incoming OSC data in TouchOSC, but also the IP address of the target-device hosting the TouchOSC application. BlueStacks works by generating an emulated Android device that, in turn, generates its own individual IP address, which is separate from that of the computer on which BlueStacks is running. This abbreviated, "virtual"/secondary IP address is contained within the "real"/primary IP address of the computer hosting BlueStacks, and is inaccessible without first passing through the primary IP address. This made it impossible to establish a direct connection from the Max/MSP patch to the TouchOSC application, due to an inability to assign multiple IP addresses to the [udpsend] object.

The second major issue involved a high level of inaccuracy in the record quantization process, resulting from combining the use of [pipe] and [seq~] objects in Max/MSP to carry out this task. By default, [pipe] functions as a way of delaying any messages arriving at its input by a period of time denoted as milliseconds in its arguments, but is also used in Max as the standard way of facilitating quantization by assigning to it the "@quantize" argument. This enables [pipe] to delay incoming messages by a factor ranging from 1/128-4 beats in relation to the global transport of the patcher. The problem herein lies that, by only allowing for incoming messages to be delayed to the next "step" in time as dictated by the quantization value and not pushed back in time to the previous "step", the result is not that of a true system of quantization. If, for instance, the user sets the quantization value to 1/4 and aims to record a single note on the first beat of the bar but triggers the note-on message a fraction of a second after beat number 1, rather than being moved back in time and stored in the position originally intended by the user, as would be the case with a true system of quantization, the note is instead "locked" to beat number 2; thus giving an inaccurate representation upon playback of the musical phrase inputted by the user. As such, the requirement that notes be played on or before the quantization "step" for which they are intended when implementing [pipe] to fulfil the role of record quantization makes for a highly unergonomic and inefficient process that seriously detracts from the level of user enjoyment, immersion and fluidity experienced when interacting with the system while record quantization is enabled.

In response to this problem it was important to develop in Max a system for proper, real-time record quantization that provided support for the polyphonic input of MIDI notes. There are numerous examples of varying ways to achieve this with monophonic input, which have been developed and shared by members of the Cycling '74 online community, but those which incorporate support for polyphony are few and far between. One of the most common monophonic methods, and that which was utilised in developing the polyphonic record quantization system for *ScreenPlay*, is to ascertain whether or not a note-on message triggered by the user is received in the first or second half of the "step" in time of the assigned quantization value by employing the fundamental rules of Max time value syntax, which divides the duration of each beat into 480 "ticks" - independent of tempo. For instance, taking again the quantize value of 1/4 as with the previous example: if the user is aiming to record a single note on the first beat of the bar in 4/4 and triggers a note-on message at any time during the latter half of beat number 4 (i.e. with a "tick" value > 240 with respect to beat 4) or the first half of beat 1 ("tick" value < 240 with respect to beat 1), then the note will be stored and played-back on the first beat of the bar; if the user is late in their execution and triggers the note-on message during the latter half of beat 1 ("tick" value > 240 with respect to beat 1) then the note will instead fall on beat number 2 in the bar when recorded and played-back.

In addition to those discussed above, there were a number of other minor issues encountered as a result of the system configuration used during the showcase at Salford Sonic Fusion Festival in 2014; one of which was the inclusion in the hardware specifications of the "touchtables" of an infrared layer positioned a millimetre or so above the touchscreen surface. The source of this was embedded in the table body/casing that housed the screen/computer and, due to being unable to deactivate it, often caused note-on messages to be triggered by users accidentally prior to actually making contact with the playing surface. Also, the use of [seq~] - a signal-driven MIDI sequencer used as standard in Max to record, store and playback musical loops and phrases of any length - was the source of problems with playback related to altering the duration of loops and musical phrases after they had been recorded. Because [seq~] has no fixed internal reference of time and/or duration - and instead contains an arbitrary unit of time which can manifest as any duration - once the incoming list of MIDI messages arrives at its input each of its individual constituent elements is fixed in the position in time upon which it enters the object in relation to the total duration of that particular sequence. This means that, if the length of the sequence is increased from 1 to 2 bars or decreased from 2 to 1 bars, then, rather than maintaining the timing and tempo/playback speed of the first original bar and adding a second empty bar to the end of the first, or removing the second bar and, again, maintaining the timing/tempo of the first, the entire sequence is played back and either doubled or halved in tempo. This was another reason that prompted the development of a bespoke system to quantize, record and play back the musical lines/phrases inputted by the user(s).

In order to facilitate real-time record quantization of polyphonic source-material in the new system, the [live.step] object (the primary function of which is to serve as a front-end step-sequencer UI object to be interfaced with directly by the user) was used in place of [seq~] as the vehicle into which musical phrases/loops were recorded, stored, and played back from after being quantized. As has been touched upon already, the importance of this within the context of the Cycling '74 community was significant, due to a lack of native support for the process in Max as well as alternative solutions proposed by users from the online community. After devoting a considerable amount of time and effort to developing and fine tuning the system in order to reach the stage at which it was fully and reliably functional, it came to light during tests that the [live.step] object does not support arbitrary durations. Instead, notes inputted into [live.step] can have any duration that is a multiple of a 128th note, up to the length of 2 beats; effectively ruling out not only arbitrary durations but, even more critically, those which are not multiples of a 128th note and/or have a duration greater than 2 beats, such as 1.5 beats, 3 beats, 4 beats, 5 beats etc. This problem is compounded within the context of *ScreenPlay* due to the fact that, much in the same way as [pipe] handles the quantization of messages arriving at its input by delaying them and placing them at the next "step" in time as dictated by the value of quantization, [live.step] also assigns durations to incoming MIDI notes by stretching them out and positioning the note-off message at the end of the next available duration. This is not overly problematic and, for the most part, works fine when used with a percussive or staccato sound-source - which, coincidentally, was the nature of the sound-source used throughout the process of developing the system and the reason why the problem was overlooked - but becomes a major issue when attempting to record and play back sustained-note harmonic/melodic progressions and phrases. If the user wants to record a note or chord of 1 beat in length and releases said note/chord fractionally after the duration for which they were aiming has already passed, then the resultant recording will reveal upon playback a duration of 2 beats and thus a huge disparity between that which was initially inputted by the user.

Ultimately, within the context of *ScreenPlay*, the only feasible option going forward was for the quantization system to be scrapped and replaced with an alternative, more suitable means. In hindsight it may seem a glaring oversight to have dedicated so much time to developing the system without noticing such a fatal flaw in the functionality of the fundamental object upon which the entire design depended, but the effort put into the process was not completely in vain. As has already been mentioned, the quantization system is still perfectly workable when dealing with percussive or staccato sound-sources and is relatively unique in its support of polyphonic MIDI input. Additionally, the Cycling '74 team was notified about the discovery made with regard to the [live.step] object's lack of support for arbitrary durations and, even though this was something of which they were already aware, it was communicated in their response that facilitating support for arbitrary durations would be put under consideration, which means that, if this comes to pass, there is potential for the range of possible applications of the quantization system to expand and for it to fulfil entirely its originally intended capabilities/feature-set.

After some careful consideration it was decided that the most appropriate way to move forward with the development of *ScreenPlay* was to migrate the Max patch from Max/MSP to Max for Live, which allows for the process of record quantization to be taken care of by the protocols already established within Ableton Live 9. The migration process in itself was not a simple one and the fundamentals of the patch required a great deal of evolution and adaptation to work in the new environment. There are both advantages and disadvantages of moving the central patch of the system from Max/MSP to Max for Live, but it would seem that, upon completing the transitional process, the positive implications for the functionality of *ScreenPlay* far outweighed the negative.

Principally, the system was given the additional dimension of functioning not only as a multi-user "Computer Network Music" system (Early Computer Network Ensembles, n.d.; Brown, n.d.) designed specifically for use in collaborative, improvisatory performance, but also as a single-user studio compositional tool. This is due to the ability to record both MIDI and audio generated as a result of the interactions of the user(s) with the system directly into Ableton Live - which was not possible with the Max/MSP iteration of the patch - as well as to have full control over up to sixteen different instruments/parts directly from a single touchscreen device hosting the TouchOSC GUI, and apply the generative/transformative algorithms to each of these individual parts within the overall musical arrangement when composing as a single-user.

The move also effectively solved an issue with the Max/MSP version of the system relating to the best way in which to incorporate elements of control over sound-design, as well as a wider variety of preset sounds to begin with. For the Salford Sonic Fusion Festival exhibition all MIDI note information generated from the users' interactions with the system was sent back out of the Max/MSP patch and into an Access Virus synthesizer. This worked well within the context of the first ever public showcase of the system, but did not allow for any alteration to/selections of the sounds generated without doing so via the synthesizer itself. Programming these functions into the system would have been a complex and time-consuming process, but not impossible; while incorporating all sound-generating elements of the system into the patch itself was another mooted possibility, which would have provided greater flexibility but, again, would have been a drain on valuable time better spent developing the patch elsewhere as well as being extremely heavy on the CPU had it come to pass.

Using Max for Live provides users with the entire Ableton library of sounds and instruments, as well as any other third-party VST plug-ins they may have at their disposal; although selecting and editing them (beyond the possibilities afforded by the textural/timbral topical opposition transformations) is still a process that needs to be carried out in Live rather than via the TouchOSC GUI. Likewise, there exists the possibility to allow for the grid-based playing surface on the GUI to be bypassed and instead enable interaction with the system to be carried out via any MIDI-compatible control-surface that did not exist with the previous iteration of the system. Additionally, the trend in recent years of artists such as Mad Zach releasing their music as Ableton Live sound Packs containing the constituent parts of a composition to allow for fans of their music to playback and interpret the piece in their own way as opposed to "listening" in the traditional sense of the word - and in doing so addressing the issues highlighted by Roland Barthes with regard to the 'two musics' and the "death" of the amateur musician in *Musica Practica* (1977, p. 149) - demonstrates the potential for harnessing the creative possibilities of *ScreenPlay* in a similar manner that would not have been possible when running the central patch of the system through Max/MSP. Other related examples, such as *Reactable Gui Boratto* (Boratto, Jordà, Kaltenbrunner, Geiger & Alonso, 2012) and *Reactable Oliver Huntemann* (Huntemann, Jordà, Kaltenbrunner, Geiger & Alonso, 2012), as well as the recent release by Native Instruments of the *Stems* audio format (Ramley, 2015), show that it is imperative the design of a forward-thinking, all-encompassing ICMS such as *ScreenPlay* should possess this potential; a point summarised best by Barthes, who states:

> What is the use of composing if it is to confine the product within the precinct of the concert or the solitude of listening to the radio? To compose, at least by propensity, is *to give to do*, not to give to hear but to give to write. (1977, p. 153)

Allying the potential support for this application of *ScreenPlay* with its built-in generative and transformative algorithms would provide users with a level of uniqueness to the experience of "playing" their favourite music as opposed to listening to it in comparison to that which is provided by the other examples mentioned.

The large online presence of communities surrounding Max for Live and TouchOSC - and the existing commercial market for Max for Live devices - also means that the trio of Max for Live MIDI Devices and the Ableton Live Effect Rack that now make up the central programming of the system and the accompanying GUI layout can be easily distributed to, shared between and modified by a multitude of potential users. Although one advantage of utilising Max/MSP was that anybody could use the system for free simply by downloading the Runtime version of Max, which enables the running but not editing of patches, as opposed to requiring a paid license for both Ableton Live and Max for Live. This potentially widened the scope of *ScreenPlay*'s target demographic to include novice- and non-musicians, although realistically, it is much more likely that such individuals would interact with the system as a gallery installation at a public exhibition than download the required software to run it themselves at home, regardless of the cost involved in doing so. A further, albeit minor disadvantage is that, where before the record quantization and playback launch quantization values could be changed at the discretion of each individual user with regard to the specific instrument/sound within the musical output of the system for which they were responsible in controlling, utilising the architecture of Ableton Live to carry out these functions means that the effects are global rather than localised, regardless of whether the system is configured to run in single or multi mode.

Finally, the exhibition of *ScreenPlay* at Salford Sonic Fusion Festival offered up a valuable opportunity to gain feedback from many first-time users of the system at a very early stage in its development cycle. In general the system was very well received, although the majority of the issues discussed thus far, having already been identified during setup and tests prior to the actual exhibition, were compounded by a number of the comments and suggestions received; such as the need for an improved quantization system, the desire for increased sonic possibilities etc. Of greater interest were the suggestions that the use of descriptive language would be a worthwhile consideration in order to make the selection of key signature/scale and alteration of sounds etc. a more accommodating experience for first-time users and novice-level musicians (the implementation of which is discussed in the Methodology and Implementation chapter), and that perhaps there was a danger of watering or "dumbing" down the traditional learning process associated with attaining a level of expertise in the areas of music theory and musicianship - upon which the replacement of actual key signature/scale names with descriptive/emotive language could also have a negative impact.

Granted, it could be argued that the inclusion in *ScreenPlay*'s design of a "key-locking" system potentially makes it more difficult to learn the basics of music theory through interacting with the system than would following the traditional route associated with learning, for example, the piano. However, *ScreenPlay* is not designed as an educational tool and this is a necessary compromise to make in order to maximise the effectiveness of implementing the *learning curve* guidelines provided by Blaine and Fels in their 11 criteria for collaborative musical interface design (2003). Also, the ability when playing with *ScreenPlay* to increase proficiency and musicality through practise as discussed in relation to Blaine and Fels' *pathway to expert performance* criterion (see Methodology and Implementation chapter) enables users to improve technical skills associated with the playing styles of many more traditional/acoustic instruments, such as dexterity, muscle memory and timing etc., as well as gain an understanding of music theory "by ear". This process of learning is one which has been followed by many a guitarist who, while without learning to read music, still manage to gain a solid and instinctive understanding of key signatures, harmony and chord progressions/cadences etc. Additionally, users are provided the option not only of disabling the "key-locking" system and instead displaying on the TouchOSC grid-based playing surface a chromatic scale, but also the ability to bypass the playing surface displayed on the GUI altogether and instead use any other MIDI controller they desire to play and record notes into the system.. As a result, a fairly even balance has been struck between providing a pleasurable interactive experience for non-expert users and novice musicians with the potential to maintain the interest of users beyond the point of initial intrigue as their affinity with the system increases, regardless of their level of experience and/or musical knowledge; which is one of the primary goals of the system.

4.2 8th MMU Postgraduate Research Conference "Innovation"

On the 5th November 2015 a late iteration of *ScreenPlay*, in which the Markovian generative algorithm and "stability-destruction", "open-close" and "light-dark" textural/timbral topical opposition transformations were fully functional, was showcased at the 8th MMU Postgraduate Research Conference "Innovation". The opportunity was taken to distribute questionnaires to users with a view to discovering how easy and intuitive it was for individuals from a range of demographics to interact with the system so as to gauge the success of its design and highlight any areas in need of further development and improvement over the course of the final year of research.

The questionnaire (which can be found in Appendix 4) was answered by a total of twenty people - a small number of which were collected at a later date - spanning a broad range of demographics in terms of age, musical background/proficiency, experience with ICMSs and relationship with touchscreen technology. The total number of respondents to the questionnaire was lower than had been hoped, with certain demographics being unaccounted for, such as app developers and individuals above the age of sixty. In spite of this the exercise did prove to be worthwhile in revealing trends in the results that gave a good enough estimation of how successful *ScreenPlay* was in terms of fulfilling its principal design aim of being a system that provides novice users and musicians with an intuitive and satisfying interactive musical experience whilst maintaining a similar level of engagement and intrigue for more advanced users and musicians. A final showcase was also carried out in April 2016 exclusively with academic staff and postgraduate research students from the University of Salford - although this time without the distribution of questionnaires - once the probability-based transformative algorithm for the "joy-lament" topical opposition transformation was functional, in order to gain an insight into any final developments/improvements that could be made to the system and GUI over the last months of the research project.

Overall the vast majority of respondents to the questionnaire found the experience of interacting with *ScreenPlay* straightforward, intuitive and enjoyable, regardless of their age, musical background/proficiency, previous experience with ICMSs and relationship with touchscreen technology. Most valuable moving forward with the development of *ScreenPlay* over the course of the final year of research, however, were the comments left by users in response to the final question asking them whether they felt that the addition/availability of an instructions page would be beneficial to *ScreenPlay*, as well as to leave any further comments with regard to how the design of the system could be improved upon. All but one of the respondents stated that an instructions page would be beneficial, while comments in relation to further development/improvements included a number of suggestions, such as: developing *ScreenPlay* into a standalone application for Android/iOS; colour coding related control objects on the TouchOSC GUI; and adding pop-up instruction/tip windows when first interacting with control objects on the GUI.

The design of *ScreenPlay* means the system would be very well suited to function as a standalone application, and this is something that had been considered a potential future development even before the suggestion made by one of the users. That this has been recognised by a user reiterates the suitability of the system to exist in this format. However, to have developed *ScreenPlay* as a standalone application would not have been in line with one of the core design principles laid out before beginning the development process, which was to use only existing technology. It is possible to save Max/MSP patches as standalone applications for Windows and OS X but, due to the reliance of *ScreenPlay* upon Ableton Live/Max for Live and external TouchOSC GUIs, to do so is not feasible.

The suggestion to use colour coding for control objects with related functionality on the GUI is something that was incorporated into the design of the controls for the topical opposition transformations. Prior to collecting the user feedback questionnaires all of the transformation controls were coloured blue in line with the overall visual design/theme of the interface, and were distinguished from one another using text. Text is still used to signify to users the effect on the musical output of the system had by each transformation and how the position of the corresponding slider affects this. However, each slider is now coloured differently and the on/off toggle buttons correspond with this as opposed to being labelled in the same way as the sliders. The result is a vast improvement in both the appearance of the transformative/generative control page on the GUI - by vastly reducing the amount of text used - and the perceived affordances of the system/GUI in terms of the intuitive understanding users have regarding the functionality of these controls. It may have also been possible to implement colour coding on the main control page of the GUI but, in this case, the aesthetics of the system were prioritised, which, with a modern and innovative ICMSs such as *ScreenPlay*, are also an important factor for consideration.

Finally, although the inclusion of temporary tips/instructions that appear when touching a control object/parameter for the first time is difficult to achieve using TouchOSC, the idea of a walkthrough - suggested by one of the users who also made the suggestion of pop-up instruction tips - was considered and intended to be included in the design. The working concept was to use a dedicated page of the GUI on which, after being started by the user, a slideshow-type walkthrough would be displayed that introduced each element of the interface one at a time along with accompanying directions relating to functionality. The user would be in control of when to progress to the next stage of the walkthrough and introduce a new element of the interface until it was complete. This is a much easier instructional model to implement in TouchOSC than pop-up tips that appear on the existing interface, due to the fact that a slideshow-type walkthrough can be reset/started again by the user if necessary without having to restart TouchOSC and/or the *ScreenPlay-CTRL* Max for Live device. Unfortunately, due to the extensive amount of time dedicated to developing and refining the probability-based transformative algorithm for the "joy-lament" topical opposition transformation as a result of its uniqueness and complexity, the implementation of the concept for a walkthrough tutorial page included in the GUI of the finalised version of the system was not feasible; although it is still a worthwhile consideration for future development. In place of this, a simple and brief accompanying document outlining the functionality of the GUI has been drawn up, which can be found in Appendix 2.

4.3 April 2016 Showcase of ScreenPlay

The information obtained from the user feedback questionnaires collected at the 8th MMU Postgraduate Research Conference "Innovation" was crucial in giving confirmation to the design path down which *ScreenPlay* had been taken up until that stage, as well as in providing information and ideas to further develop the system. However, it was also felt necessary to showcase the system for a final time once the improvements made to the system following the showcase in November 2015 at Manchester Metropolitan University were complete, and the probability-based transformative algorithm for the "joy-lament" topical opposition transformation had been developed. The showcase, which was carried out privately over the course of three days, was open to academic staff and postgraduate research students at the University of Salford and provided a final opportunity to obtain an outside perspective on the overall features, performance and usability of the system at a late stage in its development cycle.

Overall the feedback received was extremely positive and reaffirmed the confidence in *ScreenPlay*'s design taken from the response of users at the 8th MMU Postgraduate Research Conference "Innovation" showcase. The rigorous testing to which the system was exposed through an intensive exploration of its entire feature set did, however, highlight a number of minor issues in the programming, which were later resolved. The most significant of these related to an issue with pitch values being generated by the "joy-lament" transformation that were not part of the chosen key signature/scale and, depending on the position of the "joy-lament" slider when the transformation was triggered, could result in multiple dropped notes not being inserted into the clip or even halting the entire transformation process. The reason for this was that, if a pitch value was generated either side of the MIDI note number range of 0-127, the generated value would be replaced with either a 0 or 127 depending on whether it was above or below the limits of the MIDI range. This method of ensuring all generated pitch values stayed within the range of 0-127 did not, however, account for the fact that there are multiple key signatures/scales in which the MIDI note numbers 0 and/or 127 do not appear.

The problem was rectified by dynamically changing the boundaries of the range within which notes can be generated, depending on the key signature/scale chosen by the user, so that the upper and lower limits of the range and the pitches used to replace those generated outside the range correspond with the root note of the key. The opportunity was also taken whilst doing this to greatly reduce the overall range within which it is possible to generate pitch transformations, from the upper and lower limits of the entire MIDI note number range to the root notes of the chosen key signature/scale above and below the highest and lowest notes originally present in the clip at the time the transformation is first triggered. This development proved to be a huge improvement in terms of the overall musical results of the transformation process by helping to ensure that the original musical character of the loop/phrase is retained following the transformation. The process of rectifying this issue also provided the opportunity to considerably speed up the time taken for the transformation process to be completed after being triggered. The programming and GUI of *ScreenPlay* were finalised upon completion of all the necessary tweaks following this private showcase.

5. Conclusion

Upon completion of the research project it can be said that the process of designing and developing *ScreenPlay* has led to the establishment within the field of HCI in music of a new and unique approach to ICMS design that is all-encompassing of the *transformative*, *generative* and *sequenced* design models, while remaining accessible, engaging and entertaining to both novice and experienced users/musicians alike. The ability to record and play back up to twelve individual clips in *ScreenPlay* is an evolution of the typical approach to *sequenced* ICMS design; examples of which generally afford the user the ability to arrange in real time the constituent parts of a pre-existing composition, thus resulting in an exclusive interpretation of the piece. The implementation of both first and second order Markov chains in *ScreenPlay*'s generative algorithm can also be seen to be an advancement in relation to the prevailing approach to designing *generative* systems; most of which focus on generating simplistic ambient electronic music. By supporting up to sixteen separate instruments/sounds, each of which is subject to its own generative algorithm and can be controlled by individual users via dedicated GUIs when the system is configured in multi mode as a collaborative interactive installation, or can all be controlled from a single GUI when configured in single mode and used as a studio compositional tool, as well as providing the ability to create with the system any genre of electronic music, *ScreenPlay* demonstrates a level of depth and complexity beyond that of any current examples of *generative* ICMSs. The introduction of topic theory into the realms of HCI in music and electronic music composition/production/performance through its inclusion in the form of a topic-theory-inspired transformative algorithm in *ScreenPlay* is a novel endeavour that has proven very successful. The probability-based algorithm itself has been designed exclusively for *ScreenPlay* and, although probability-based algorithms have been used in music for many years in a multitude of different contexts, it can be said with a great deal of certainty that the appropriation of the concept in the manner inherent to *ScreenPlay*'s "joy-lament" topical opposition transformation is one of a kind. Applying the *transformative* approach to ICMS design in the context of modern electronic music composition/production/performance in *ScreenPlay* is also something rarely seen, given that current examples of *transformative* systems are few and far between, with this approach to ICMS design being more commonly associated with early "score-follower" systems designed specifically to be integrated into the live performance of particular compositions using traditional instrumentation and electronics.

The aforementioned multifunctionality of *ScreenPlay* with regard to its capacity to be used both as a collaborative, improvisatory multi-user interactive composition/performance installation and as a single-user studio compositional/production tool is also unique, excepting only the *Reactable* (Jordà, Kaltenbrunner, Geiger, & Alonso, 2003-present). Although the lack of two-way communication between user and computer afforded by *Reactable* in terms of the computer contributing in conjunction with the user to the generation of musical material could question the legitimacy of its categorisation as an ICMS and, instead, suggest it more accurately be defined as a digital modular synthesizer; despite its support of multi-user collaboration in composition/performance. Regardless, the multifaceted nature of *ScreenPlay*'s design opens up the possibility of the system having a significant impact not only in the field of HCI in music but also electronic music composition/production/performance; something in which *Reactable* has already paved the way with substantial success. The implementation of *ScreenPlay*'s generative and transformative algorithms allows for new and unique ways of breaking routine in collaborative composition/performance and generating new musical ideas when composing in the studio. In a similar way to that in which *Reactable Mobile* supports musical releases from the likes of Gui Boratto and Oliver Huntemann - enabling users to interactively play/perform the included compositions using their constituent parts and adding in new instruments/effects etc. - the tight integration between Ableton Live and *ScreenPlay*, in tandem with the generative and transformative algorithms, presents a plethora of new possibilities in terms of interacting with musical releases in the form of Ableton Live Packs. The framework for such releases is already established, as exemplified by those from artists such as Mad Zach, but in the same way as does *Reactable Mobile,* currently only incorporates a *sequenced* approach to interaction.

Reactable also serves to demonstrate a number of ways in which *ScreenPlay* could further improve and develop going forward, the first of which being the design of a bespoke GUI. The engaging, entertaining, visually stunning and unique GUI is one of *Reactable*'s greatest strengths, and, in the same way as has *Reactable*'s, developing a similarly exciting and seductive GUI for *ScreenPlay* would almost certainly increase its appeal to potential users, and thus its impact. To do so would also make more feasible the provision to users of "feedthrough" information (Dix, 1997, pp. 147-148), allowing them to see and, potentially, influence the actions of other users in real time directly via the GUI. This was a planned inclusion in *ScreenPlay*'s design from the beginning of the development process, due primarily to the possibilities it would open up with regard to widening the scope further still for collaborative engagement with other users/players in improvisatory composition/performance. Unfortunately, however, the creative appropriation of "feedthrough" information was abandoned as a result of TouchOSC - the Android/iOS application used to design and host the GUI - not being optimized to support such a task without significant and time-consuming programming workarounds; although arbitrary, meta-level information such as when a user activates the global metronome or changes the global clip trigger and record quantization values is still shared between interfaces.

The creation of a custom GUI for *ScreenPlay* would naturally give way to further expansion of the system involving the design of a dedicated application for Android/iOS devices. The conceptual multifunctionality of *ScreenPlay* is well suited to this and could enable users to engage in interactive composition/performance individually with the system, or connect with other users' devices via WiFi or Bluetooth. The affordance of dedicated interfaces to each user in a collaborative ICMS mobile application would be unique to *ScreenPlay*, and may also increase the maximum number of users able to interact with one another in collaborative, improvisatory composition/performance simultaneously. The success of *Reactable*'s transition from a tabletop hardware instrument into a mobile application in the form of *Reactable Mobile* and, more recently, *ROTOR* (Jordà, Kaltenbrunner, Geiger, & Alonso, 2016) serves to demonstrate the potential for *ScreenPlay* to have a significant commercial impact in this incarnation. Such a development would also help to increase the impact of the system in electronic music composition/production/performance, due to being compatible with any music-making software/DAW and not limited to working only within Ableton Live via Max for Live. Of greater importance would be the increased impact of the system overall, and in particular in helping to further advance the establishment of HCI in music in popular culture as a result of not requiring any existing music technology software/hardware whatsoever.

The use of existing technology was, however, a fundamental design principle of *ScreenPlay* from the outset, and comes with its own benefits. The potential for commercial impact is still palpable, given the already established and ever-growing market for Max for Live devices. The thriving communities surrounding Ableton Live, Max/MSP, Max for Live and TouchOSC also present the possibility of maximising the impact of the system within electronic music production. Furthermore, the current incarnation of *ScreenPlay* enables users/programmers to refine and develop the system to suit their own particular creative needs and musical tastes. While the system would still allow for the development of a custom GUI in its current incarnation, there is also the possibility of designing a number of variations on the current GUI; whether simplified and more complex versions for smaller and larger mobile touchscreen devices respectively, or in order to provide support for other, similar applications to TouchOSC, such as Lemur (Slater et al., 2011-present). Alternatively, the addition of controls for the generative and transformative algorithms directly on the *ScreenPlay-GEN* and *ScreenPlay-TRNS4M* Max for Live MIDI Devices would allow users to bypass the GUI entirely, expanding upon the current functionality of bypassing the playing surface in order to use any other MIDI controller when incorporating the system into the compositional/production process.

The topic-theory-inspired transformative algorithm itself, and the appropriation of topic theory as a whole in electronic music, also has vast potential for further development. This is particularly apparent in the context of manipulating texture and timbre, given the importance of these sonic characteristics in electronic music and their inherent ability to evoke other-worldly imagery beyond the realms of reality - as evidenced by Smalley's spectromorphological concept of *technological listening* (the perception by the listener of the technology or technique behind a sound/piece of music as opposed to the music itself (1997, p. 109)) - which is highlighted by Monelle as prevailing in the signifieds of musical topics throughout the history of topic theory (2006, p. 13). For *ScreenPlay* to bring about a rejuvenation of topic theory in the modern age through its application in this manner within electronic music would, arguably, be one of its more significant achievements.

6. Reference List

1620 Data Processing System. (n.d.). Retrieved 26th May, 2015, from https://www-03.ibm.com/ibm/history/exhibits/mainframe/mainframe_PP1620.html.

7090 Data Processing System. (n.d.). Retrieved 18th March, 2017, from https://www-03.ibm.com/ibm/history/exhibits/mainframe/mainframe_PP7090.html.

Ableton. (2001-present). *Live*. [Music production software]. Berlin, Germany: Ableton.

Ableton & Akai Professional. (2013). *Push*. [Dedicated MIDI controller for Ableton Live]. Berlin, Germany: Ableton/Cumberland, RI, USA: Akai Professional.

Acuma. (2010a). Algorithmic Composition: Markov Chains in PureData. Retrieved 11th October, 2016, from http://www.algorithmiccomposer.com/2010/05/algorithmic-composition-tutorial-markov.html.

Acuma. (2010b). Algorithmic Composition: Markov Chains in Max MSP. Retrieved 11th October, 2016, from http://www.algorithmiccomposer.com/2010/05/algorithmic-composition-markov-chains.html.

Agawu, K. (1991). *Playing With Signs*. Princeton, NJ, USA: Princeton University Press.

Aisher, B. (2014). Korg Gadget for iOS. Retrieved 29th May, 2015, from http://www.musicradar.com/reviews/tech/korg-gadget-for-ios-598763.

Akai Professional. (1988-present). *MPC series*. [Hardware sampler/groovebox]. Cumberland, RI, USA: Akai Professional.

Akai Professional. (2009). *APC40*. [Dedicated MIDI controller for Ableton Live]. Cumberland, RI, USA: Akai Professional.

Akai Professional. (2012-present). *iMPC*. [Software emulation of MPC sampler for iOS]. Cumberland, RI, USA: Akai Professional.

Alsop, R. (1999). Exploring the Self Through Algorithmic Composition. *Leonardo Music Journal*, *9*, pp. 89-94. DOI: 10.1162/096112199750316866.

Appignanesi, R. & Garratt, C. (2013). *Introducing Postmodernism A Graphic Guide* (2nd ed.) London, UK: Icon Books Ltd.

Ariza, C. (2011). Two Pioneering Projects from the Early History of Computer-Aided Algorithmic Composition. *Computer Music Journal*, *35* (3), pp. 40-56. DOI: 10.1162/COMJ_a_00068.

Arturia. (2014). *iProphet*. [Virtual-analogue synthesizer for iPad]. Grenoble, France: Arturia.

Arturia iProphet. (2014). *Computer Music Magazine*, *211*, p. 106.

Babbitt, M. (1955). Some Aspects of Twelve-Tone Composition. *The Score and I.M.A Magazine*, *12*, pp. 53-61.

Babbitt, M. (1962). Twelve-Tone Rhythmic Structure and the Electronic Medium. *Perspectives of New Music*, *1* (1), pp. 49-79. DOI: 10.2307/832179.

Babbitt, M. (2011). *The Collected Essays of Milton Babbitt*. Princeton, New Jersey, USA: Princeton University Press.

Barthes, R. (1977). *Image Music Text*. London, UK: Fontana Press.

Barthes, R. (1983). *Elements of Semiology* (8th ed.). New York City, New York, USA: Hill and Wang.

Barthes, R. (1957/1972/1991). *Mythologies* (1st ed./English translation/25th ed.). Paris, France: Editions du Seuil/London, UK: Jonathan Cape Ltd./New York City, New York, USA: The Noonday Press.

Batchelor, P. (2014). *EtherSurface*. [Synthesizer for Android]. Boston, MA, USA: Batchelorsounds.

Benford, S. (2010). Performing Musical Interaction: Lessons from the Study of Extended Theatrical Performances. *Computer Music Journal*, *34* (4), pp. 49-61. DOI: 10.1162/COMJ_a_00025.

Benford, S., Crabtree, A., Reeves, S., Flintham, M., Drozd, A., Sheridan, J. & Dix, A. (2006). The Frame of the Game: Blurring the Boundary between Fiction and Reality in Mobile Experiences. In *Proceedings of the 2006 SIGCHI Conference on Human Factors in Computing Systems: 22-27 April 2006, Palais des Congrès, Montreal* (pp. 427-436). Montreal, Canada.

Benford, S., Giannachi, G., Boriana, K. & Rodden, T. (2009). From Interaction to Trajectories: Designing Coherent Journeys Through User Experiences. In *Proceedings of the 2009 SIGCHI Conference on Human Factors in Computing Systems: 4-9 April, 2009, Hynes Convention Centre, Boston* (pp. 709-718). Boston, MA, USA.

Benford, S., Schnädelbach, H., Koleva, B., Anastasi, R., Greenhalgh, C., Rodden, T., … Steed, A. (2005). Expected, Sensed, and Desired: A Framework for Designing Sensing-Based Interaction. *ACM Transactions on Computer-Human Interaction*, *12* (1), pp. 3-30.

Bischoff, J., Perkins, T., Brown, C., Gresham-Lancaster, S., Trayle, M. & Stone, P. (1987-present). *The Hub*. [Interactive computer network music ensemble].

Björk. (2011). *Biophilia*. [Interactive album application]. London, UK: Second Wind Ltd., One Little Indian Ltd. & Well Hart Ltd.

Blaine, T. & Fels, S. (2003). Collaborative Musical Experiences for Novices. *Journal of New Music Research*, *32* (4), pp. 411-428.

Blomert, C., Garcia, S., Keppmann, F., Blomert, P. & Kapp, P. (2010-present). *touchAble*. [Touchscreen-based button-matrix MIDI controller for iOS]. Berlin, Germany: Zerodebug.

Boer, R. (2011). The Robbery of Language? On Roland Barthes and Myth. *Culture, Theory and Critique*, *52* (2-3), pp. 213-231. DOI: 10.1080/14735784.2011.630890.

Booka Shade. (2006). Night Falls. [CD/studio album]. *Movements*. Berlin, Germany: Get Physical Music.

Boratto, G., Jordà, S., Kaltenbrunner, M., Geiger, G., & Alonso, M. (2012). *Reactable Gui Boratto*. [Interactive album release for Reactable Mobile touchscreen application]. Barcelona, Spain: Reactable.
Brandel, J. (2012-present). *Patatap*. [Interactive music game]. San Francisco, CA, USA: jonobr1.

Brandel, J. (2015). Patatap. Retrieved 28th May, 2015, from http://works.jonobr1.com/Patatap.

Bredin, H. (1984). Sign and Value in Saussure. *Philosophy*, *59* (227), pp. 67-77.

Brooks, S. & Ross, B. J. (1996). Automated Composition from Computer Models of Biological Behavior. *Leonardo Music Journal*, *6*, pp. 27-31. DOI: 10.2307/1513301.

Brown, C. (n.d.). Collaborations. Retrieved 12th Jan, 2014, from http://www.cbmuse.com/collaborations.

Burton, S. (2003). HYPERSENSE COMPLEX. Retrieved 21st Jan, 2014, from http://arrowtheory.com/hypersense/index.html.

C4Cat Entertainment. (2014-present). *Dynamix*. [Interactive music game]. Hong Kong: C4Cat Entertainment.

Cavia Incorporated, Sano, N. & Mitsuda, Y. (2008). *KORG DS-10*. [Virtual-analogue synthesizer for Nintendo 3DS]. Tokyo, Japan: AQ Interactive/Torrance, CA, USA: XSEED Games/Scoresby, Australia: Nintendo Australia.

Chow, O. (2013). Artist Manipulates Water with the Power of her Mind. Retrieved 27th May, 2015, from http://thecreatorsproject.vice.com/blog/eunoia-seeking-enlightenment-by-tracking-brainwaves

Council of Scientific and Industrial Research. (1949-64). *CSIRAC*. [Computer system]. Sydney, Australia.

da Cruz, F. (2009). The IBM 1620 Data Processing System. Retrieved 26th May, 2015, from http://www.columbia.edu/cu/computinghistory/1620.html.

CSIRAC. (n.d.). Retrieved 26th May, 2015, from http://museumvictoria.com.au/discoverycentre/infosheets/csirac/.

Curtius, E. R. (1948/1953). *European Literature and the Latin Middle Ages* (1st ed./English translation). Bern, Switzerland: A. Francke Verlag/New York City, New York, USA: Pantheon Books, Inc.

Daft Punk. (2001). Harder, Better, Faster, Stronger. [Single]. *Discovery*. London, UK/Los Angeles, CA, USA: Virgin Records.

Daft Punk. (2005). Technologic. [Single]. *Human After All*. London, UK/Los Angeles, CA, USA: Virgin Records.

DDR. (n.d.). Retrieved 27th May, 2015, from https://www.konami.com/ddr/.

Deleuze, G. & Guattari, F. (1980). *A Thousand Plateaus*. Minnesota, USA: University of Minnesota Press.

Dellidj, M. (n.d.). *DaftPunKonsole*. [Web-browser-based ICMS]. Lille, France: Dathink.

DeSantis, D. (2015). *Making Music: 74 Creative Strategies for Electronic Music Producers*. Berlin, Germany: Ableton.

DeSantis, D., Gallagher, I., Haywood, K., Knudsen, R., Behles, G., Rang, J., … Torsten, S. (2016). *Ableton Reference Manual Version 9 for Windows and Mac OS*. Berlin, Germany: Ableton.

Desprez, J. M. (n.d.-present). *FM Synthesizer/SynprezFM II*. [Virtual-analogue FM synthesizer for Android]. Paris, France: Jean-Marc Desprez.

Di Nunzio, A. (2014). UPIC. Retrieved 28th October, 2016, from http://www.musicainformatica.org/topics/upic.php.

Dika, N. (2014-present). *Photophore*. [Flock-modelling synthesizer for iPad]. Warwickshire, UK: Taika System Limited.

Dix, A. (1997). Challenges for Cooperative Work on the Web: An Analytical Approach. *Computer Supported Cooperative Work: The Journal of Collaborative Computing*, *6* (2), pp. 135-156. DOI: 10.1023/A:1008635907287.

Division Paris, Skrillex & Creators Project. (2014). *http://doompypoomp.skrillex.com/#/*. [Interactive audiovisual website]. Los Angeles, CA, USA: OWSLA.

DJ Pangburn. (2014). Interact with the Virtual Worlds of Teengirl Fantasy's New EP. Retrieved 29th May, 2015, from http://thecreatorsproject.vice.com/blog/interact-with-the-virtual-worlds-of-teengirl-fantasys-new-ep.

DJ TechTools. (2009-present). *Midi Fighter*. [MIDI controller]. San Francisco, CA, USA: DJ TechTools.

Dodge, C. & Jerse, T. A. (1997). Composition with Computers. In *Computer Music: Synthesis, Composition and Performance* (2nd ed.) (pp. 341-402). New York City, NY, USA: G Schirmer Inc.

Drummond, J. (2009). Understanding Interactive Systems. *Organised Sound*, *14* (2), pp. 124-133. DOI: 10.1017/S1355771809000235.

Early Computer Network Ensembles (n.d.). Retrieved 12th Jan, 2014, from http://music.columbia.edu/cmc/musicandcomputers/popups/chapter4/xbit_4_3.php.

Eco, U. (1976). *A Theory of Semiotics*. Bloomington, Indiana, USA: Indiana University Press.

Eco, U. (1989). The Poetics of the Open Work. In A. Cancogni (Ed.), *The Open Work*, 1st ed. (pp. 1-23). Cambridge, Massachusetts: Harvard University Press.

Edwards, M. (2011). Algorithmic Composition: Computational Thinking in Music. *Communications of the ACM*, *54* (7), pp. 58-67. DOI: 10.1145/1965724.1965742.

Emory, S. (2015). The Walls Have Feelings in Majestic Interactive Room. Retrieved 27th May, 2015, from http://thecreatorsproject.vice.com/blog/walls-have-feelings-in-majestic-interactive-room?utm_source=tcpfbus.

Eno, B. & Chilvers, P. (2008). *Bloom*. [Interactive music application]. Essex, UK: Opal Limited.

Eno, B. & Schmidt, P. (1975). *Oblique Strategies*. [Idea cards to promote creativity and productivity in music-making]. Essex, UK: Opal Limited.

Fels, S., Gadd, A. & Mulder, A. (2002). Mapping Transparency through Metaphor: Towards more Expressive Musical Instruments. *Organised Sound*, *7* (2), pp. 109-126. DOI: 10.1017/S1355771802002042.

Ferranti Ltd. (1951). *Ferranti Mark I*. [Computer system]. Hollinwood, UK.

Ferranti Ltd. (1951-57). *Ferranti Mark I**. [Computer system]. Hollinwood, UK.

Ferranti Ltd. (1957). *Ferranti Mercury*. [Computer system]. Hollinwood, UK.

Fikentscher, K. (2000). *"You Better Work!" Underground Dance Music in New York City*. Hanover, New Hampshire, USA: Wesleyan University Press.

Fischer, R. (2008-present). *TouchOSC*. [customizable touchscreen-based MIDI controller application]. Berlin, Germany/Sofia, Bulgaria: Hexler.

Flanagan, R. E., Nguyen, Q. & Boom, H. (2011-present). *FRACT OSC*. [Interactive musical exploration game/virtual reality environment]. Montreal, Canada: Phosfiend Systems.

Focusrite Audio. (2013). *Novation Launchpad*. [Digital groovebox adaptation/emulation of Launchpad MIDI controller for iOS]. High Wycombe, UK: Novation.

FreeStyleGames & Exient Entertainment. (2009). *DJ Hero*. [Interactive music game]. Santa Monica, CA, USA: Activision.

Funk, B. & Stearns, L. (2012). *Quantize Mapper*. [Max for Live device]. New York, NY, USA: AfroDJMac and Loudon Stearns.

Gabrielsson, A. & Lindström, E. (2010). The Role of Structure in the Musical Expression of Emotions. In P. N. Justin & J. A. Sloboda (Eds.), *Handbook of Music and Emotion: Theory, Research, Applications*, 1st ed. (pp. 367-400). Oxford, UK: Oxford University Press.

Garnett, G. E. (2001). The Aesthetics of Interactive Computer Music. *Computer Music Journal*, *25* (1), pp. 21-33. DOI: 10.1162/014892601300126089.

Gibson, J. J. (1979/1986). The Theory of Affordances. In *The Ecological Approach to Visual Perception*, 1st ed. (pp. 127-143). Boston, Massachusetts: Houghton Mifflin.

Gibson, R. & Richards, K. (n.d.). Bystander. Retrieved 12th Jan, 2014, from www.lifeafterwartime.com/media/pdfs/bystander.pdf.

Gibson, R. & Richards, K. (2004-2006). *Bystander*. [Interactive audiovisual gallery installation]. Sydney, Australia: Ross Gibson & Kate Richards.

Griffiths, D. (2014). Native Instruments Komplete Kontrol S Series. Retrieved 29th May, 2015, from http://www.musicradar.com/reviews/tech/native-instruments-komplete-kontrol-s-series-609155.

Guedes, C. (1996). *Pierre Schaeffer, Musique Concrète, and the Influences in the Compositional Practice of the Twentieth Century*. (Unpublished PhD thesis). New York University, New York City, NY, USA.

Guido. (c. 1026). *Micrologus*. [Score]. Arezzo, Italy.

Handicrafter. (n.d.-present). *Full of Music*. [Interactive music game].

Harley, J. (2002). The Electroacoustic Music of Iannis Xenakis. *Computer Music Journal*, *26* (1), pp. 33-57.

Harley, M. A. (1998). Music of Sound and Light: Xenakis's [sic] Polytopes. *Leonardo*, *31* (1), pp. 55-65.

Harmonix & MTV Games. (2007-present). *Rock Band*. [Interactive music game]. Redwood City, CA, USA: Electronic Arts.

Harmonix, Neversoft, Budcat Creations, Vicarious Visions & FreeStyleGames. (2005-present). *Guitar Hero*. [Interactive music game]. Santa Monica, CA, USA: Red Octane/Activision.

Hatten, R. S. (1994). *Musical Meaning in Beethoven: Markedness, Correlation, and Interpretation*. Bloomington, Indiana, USA: Indiana University Press.

Hiller, L. (1967). Programming a Computer for Music Composition. In G. Lefkoff (Ed.), *Computer Applications in Music* (pp. 65-88). Morgantown, West Virginia, USA: West Virginia University Library.

Hiller, L. (1969). Some Compositional Techniques Involving the use of Computers. In H. von Foerster & J. W. Beauchamp (Eds.), *Music by Computers* (pp. 71-83). Hoboken, New Jersey, USA: John Wiley & Sons, Inc.

Hiller, L. & Isaacson, L. (1956). *Illiac Suite*. [Score]. Champaign-Urbana, IL, USA.

Hiller, L. & Isaacson, L. (1958). Music Composition with a High-Speed Digital Computer. *Journal of the Audio Engineering Society*, *6* (3), pp. 154-160.

Hiller, L. & Isaacson, L. (1959). *Experimental Music: Composition with an Electronic Computer*. New York City, New York, USA: McGraw-Hill Book Company, Inc.

Holliday, W. T. (2011-present). *Audulus*. [Modular synthesis application]. San Francisco, CA, USA: Subatomic Software.

humbleTUNE. (2006-present). *Kinetic*. [Interactive generative music application]. Norrköping, Sweden/Manchester, UK: humbleTUNE.

humbleTUNE. (2010-present). *Pixel Tune*. [Interactive generative music application]. Norrköping, Sweden/Manchester, UK: humbleTUNE.

humbleTUNE. (2013-present). *Svärm*. [Interactive generative music application]. Norrköping, Sweden/Manchester, UK: humbleTUNE.

Huntemann, O., Jordà, S., Kaltenbrunner, M., Geiger, G., & Alonso, M. (2012). *Reactable Oliver Huntemann*. [Interactive album release for Reactable Mobile touchscreen application]. Barcelona, Spain: Reactable.

Hyperinstruments (n.d.). Retrieved 12th Jan 2014, from http://opera.media.mit.edu/projects/hyperinstruments.html.

Iazzetta, F. & Kon, F. (1995). *MaxAnnealing: A Tool for Algorithmic Composition based on Simulated Annealing*. Paper presented at the 2nd Brazilian Symposium on Computer Music (SBC '95), Canela, RS, Brazil. Retrieved from choices.cs.uiuc.edu/~f-kon/cm/maxanneal.ps.gz.

Image-Line. (2011-present). *FL Studio Mobile*. [Mobile DAW for iOS and Android]. Ghent, Belgium: Image-Line.

Imaginado. (2011-present). *LIVKONTROL*. [Touchscreen-based button-matrix MIDI controller for iOS and Android]. Braga, Portugal, Imaginado.

Inkle Studios Limited. (2014). *80 Days*. [Interactive novel game]. Cambridge, UK: Inkle Studios Limited.

International Business Machines Corporation. (1959). *IBM 7090*. [Computer system]. New York, USA.

International Business Machines Corporation. (1959-70). *IBM 1620*. [Computer system]. New York, USA.

International Business Machines Corporation. (1960). *IBM 7072*. [Computer system]. New York, USA.

Jakobson, R. & Halle, M. (1956). *Fundamentals of Language*. The Hague, Netherlands: Mouton & Co.

Jordà, S., Kaltenbrunner, M., Geiger, G., & Alonso, M. (2003-present). *Reactable/Reactable Mobile/Reactable Experience*. [Touchscreen-based modular synthesizer hardware instrument/iOS and Android application]. Barcelona, Spain: Reactable.

Jordà, S., Kaltenbrunner, M., Geiger, G., & Alonso, M. (2016). *ROTOR*. [Touchscreen-based digital modular synthesizer iOS application]. Barcelona, Spain: Reactable.

Kaganskiy, J. (2013). 1000 Hands. Retrieved 27th May, 2015, from http://www.universaleverything.com/projects/1000-hands/.

Kershaw, S., Kelledy, R., Meikle, G., O'neill, S. & Reiser, J. (2011-2012). Ways of Seeing: Interwoven Lives. [Audiovisual sound-art installation]. Burnley, UK.

Kestrel Games Studio. (2013-present). *Beat Beat Volcaloid*. [Interactive music game].

Koenig, G. M. (1970). Project One. In *Electronic Music Report, volume 2* (pp. 32-46). Utrecht, Netherlands: Institute of Sonology, Utrecht University.

Konami. (1998-present). *Dance Dance Revolution*. [Interactive music arcade/video game]. Tokyo, Japan: Konami.

Korg iElectribe. (2010). Retrieved 28th May, 2015, from http://www.musicradar.com/reviews/tech/korg-ielectribe-260400.

KORG Incorporated. (2010). *iElectribe*. [Virtual emulation of Electribe groovebox for iPad]. Tokyo, Japan: KORG Incorporated.

KORG Incorporated. (2010-present). *iMS-20*. [Virtual-analogue synthesizer for iPad]. Tokyo, Japan: KORG Incorporated.

KORG Incorporated. (2013). *M01D*. [Virtual-analogue synthesizer for Nintendo 3DS]. Tokyo, Japan: KORG Incorporated.

KORG Incorporated. (2014). *DSN-12*. [Virtual-analogue synthesizer for Nintendo 3DS]. Tokyo, Japan: KORG Incorporated.

KORG Incorporated. (2014-present). *Gadget*. [Mobile DAW for iPad]. Tokyo, Japan: KORG Incorporated.

KORG Incorporated. (2015). *iM1*. [Virtual-digital synthesizer for iPad]. Tokyo, Japan: KORG Incorporated.

Kutuzov, D. (2014). *Navichord*. [Interactive educational instrument for iPad].

Lady Gaga. (2013). *ARTPOP*. [Interactive album application]. Santa Monica, CA, USA: Interscope Records.

Langley, S. (n.d.). HyperSense Complex. Retrieved 21st Jan, 2014, from http://www.criticalsenses.com/hypersense.

Largillier, G., Joguet, P. & Olivier, G. (2007). *JazzMutant Lemur*. [Touchscreen-based MIDI controller]. Bordeaux, France: JazzMutant.

Larman, C. & Basili, V. R. (2003). Iterative and Incremental Development: A Brief History. *Computer*, *36* (6), pp. 47-56. DOI: 10.1109/MC.2003.1204375.

Lasserre, G. & met den Ancxt, A. (2015). La Maison Sensible. Retrieved 27th May, 2015, from http://www.scenocosme.com/maison_sensible_e.htm.

Lasserre, G., met den Ancxt, A., Ajima, L. & Nagemi, Y. (2015). *La Maison Sensible*. [Interactive audiovisual gallery installation]. Carrières-sous-Poissy, France.

Lepoultier, J. (2014). *La St. Hubert French Hunting Horn fanfare*. [YouTube video]. Retrieved from https://www.youtube.com/watch?v=UwfcNMJBMig.

Lewis, G. E. (1993). *Voyager*. [ICMS/compositions for trombone, saxophone and electronics]. Berkeley, CA, USA/Tokyo, Japan: Avant Records.

Lewis, G. E. (2000). Too Many Notes: Computer Complexity and Culture in Voyager. *Leonardo Music Journal*, *10*, pp. 33-39. DOI: 10.1162/096112100570585.

Lindetorp, H. (n.d.). *Symphone*. [Web-based ICMS]. Stockholm, Sweden.

Lippe, C. (1992). *Music for Clarinet and ISPW*. [ICMS/composition for clarinet and computer]. Kunitachi College of Music, Tokyo, Japan: Centre for Computer Music & Music Technology.

Lippe, C. (1993). A Composition for Clarinet and Real-Time Signal Processing: Using Max on the IRCAM Signal Processing System. In *Proceedings of the 1993 10th Italian Colloquium on Computer Music: Milan, Italy* (pp. 428-432). Milan, Italy.

Lippman, E. A. (1984). Progressive Temporality in Music. *The Journal of Musicology*, *3* (2), pp. 121-141. DOI: 10.2307/763540.

Lodge, I. L. (2012). German Hunting Horn. [YouTube video]. Retrieved from https://www.youtube.com/watch?v=kWC6I0hGMw4.

LongestJourney. (2011). *Anomaly Performance 0.126*. [Max for Live device].

Machover, T. (n.d.a). *Drum-Boy*. [ICMS for non-expert musicians]. Boston, MA, USA: MIT.

Machover, T. (n.d.b). *Joystick Music*. [ICMS for non-expert musicians]. Boston, MA, USA: MIT.

Machover, T. (1986-present). *Hyperinstruments*. [ICMS design project]. Boston, MA, USA: MIT.

Machover, T. & Chung, J. (1989). *Hyperinstruments: Musically Intelligent and Interactive Performance and Creative Systems*. Paper presented at the 1989 International Computer Music Conference (ICMC89), San Francisco, California, USA. Retrieved from http://quod.lib.umich.edu/i/icmc/bbp2372.1989?rgn=full+text.

Manoury, P. (1988). *Pluton*. [ICMS/composition for piano and electronics]. Paris, France: IRCAM.

Marino, G., Serra, M. H. & Raczinski, J. M. (1993). The UPIC System: Origins and Innovations. *Perspectives of New Music*, *31* (1), pp. 258-269. DOI: 10.2307/833053.

Massumi, B. (2005). Translator's Foreword: Pleasures of Philosophy. In Deleuze, G. & Guattari, F. *A Thousand Plateaus*, 11th ed. (pp. ix-xv). Minnesota, USA: University of Minnesota Press.

Matossian, N. (1986). *Xenakis*. New York City, NY, USA: Taplinger Publishing Co. Inc.

Mattheson, J. & Lenneberg, H. (1958). Johann Mattheson on Affect and Rhetoric in Music (II). *Journal of Music Theory*, *2* (2), pp. 193-236.

Meikle, G. (2016). Examining the Effects of Experimental/Academic Electroacoustic and Popular Electronic Musics on the Evolution and Development of Human–Computer Interaction in Music. *Contemporary Music Review, 35* (2), pp. 224-241. DOI: 10.1080/07494467.2016.1221634.

Miranda, E. R. (n.d.a). *Inter-Harmonium*. [Interactive BCI music system].

Miranda, E. R. (n.d.b). *BCMI-Piano*. [Interactive BCI music system].

Miranda, E. R. (2003). On the Music of Emergent Behaviour: What can Evolutionary Computation Bring to the Musician? *Leonardo*, *36* (1), pp. 55-59. DOI: 10.1162/002409403321152329.

Miranda, E. R. & Brouse, A. (2005a). Interfacing the Brain Directly with Musical Systems: On Developing Systems for Making Music with Brain Signals. *Leonardo*, *38* (4), pp. 331-336. DOI: 10.1162/0024094054762133.

Miranda, E. R. & Brouse, A. (2005b). *Toward Direct Brain-Computer Musical Interfaces*. Paper presented at the 5th International Conference on New Interfaces for Musical Expression (NIME '05), Vancouver, BC, Canada. Retrieved from http://cmr.soc.plymouth.ac.uk/publications/NIME05-BrouseMiranda.pdf.

Mitchell, S. M. & Seaman, C. B. (2009). *A Comparison of Software Cost, Duration, and Quality for Waterfall vs. Iterative and Incremental Development: A Systematic Review*. Paper presented at the 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM '09), Lake Buena Vista, Florida, USA. DOI: 10.1109/ESEM.2009.5314228.

Monelle, R. (2000). *The Sense of Music*. Princeton, New Jersey: Princeton University Press.

Monelle, R. (2006). *The Musical Topic: Hunt, Military and Pastoral*. Bloomington, Indiana: Indiana University Press.

Moore, S. (2012). Interval Size and Affect: An Ethnomusicological Perspective. *Empirical Musicology Review*, *7* (3-4), pp. 138-143. Retrieved from http://emusicology.org/article/view/3747/3397.

Moorefield, V. (2005). *The Producer as Composer: Shaping the Sounds of Popular Music*. Cambridge, MA: MIT Press.

Mozart, W. A. (1786). Non più andrai. [Score]. *The Marriage of Figaro, K. 492*.

Mozart, W. A. (1788-1791). *Piano Concerto No. 27 in B ♭ Major, K. 595*. [Score].

Nagle, P. (2014). Korg Gadget for iPad. Retrieved 29th May, 2015, from http://www.soundonsound.com/sos/mar14/articles/app-works-0314.htm.

Najle, M. (2010-present). *iDaft*. [Web-browser/iOS/Windows-based ICMS]. Buenos Aires, Argentina: Najle.

Native Instruments. (2000-present). *Traktor*. [DJ software]. Berlin, Germany: Native Instruments.

Native Instruments. (2009-present). *Maschine*. [MIDI controller/groovebox]. Berlin, Germany: Native Instruments.

Native Instruments. (2011-present). *iMaschine*. [Software emulation of NI Maschine groovebox for iOS]. Berlin, Germany: Native Instruments.

Native Instruments. (2014). *Komplete Kontrol S-Series*. [MIDI controller]. Berlin, Germany: Native Instruments.

Norman, D. (1994). *Defending Human Attributes in the Age of the Machine*. [CD-ROM]. Retrieved 17th March, 2014, from https://www.youtube.com/watch?v=NK1Zb_5VxuM.

Norman, D. (2002). *The Design of Everyday Things* (2002 ed.). New York City, NY, USA: Basic Books.

Norman, D. (2004). Affordances and Design. Retrieved 17th Mar, 2014, from http://www.jnd.org/dn.mss/affordances_and.html.
Novation. (2009). *Launchpad*. [Dedicated MIDI controller for Ableton Live]. High Wycombe, UK: Novation.

Novation. (2015). *Launchpad Pro*. [Dedicated MIDI controller for Ableton Live]. High Wycombe, UK: Novation.

Nowak, A. (2015). *Launch Buttons*. [Touchscreen-based button-matrix MIDI controller for Android]. Potsdam, Germany: BassApps.

Oblique Strategies. (n.d.). Retrieved 29th May, 2015, from http://www.enoshop.co.uk/product/oblique-strategies?filter=Oblique%20Strategies.

Oswalt, P. (2002). Iannis Xenakis' Polytopes. *Contemporary Music Review*, *21* (2-3), pp. 35-44. DOI: 10.1080/07494460216658.

Padberg, H. (1964). *Computer-Composed Canon and Free-Fugue*. (Unpublished PhD Thesis), St. Louis University, St. Louis, MO, USA.

Paine, G. (1999-2001). *Gestation*. [ICMS].

Paine, G. (2013). Gestation. Retrieved 12th Jan, 2014, from http://www.activatedspace.com/Installation_Works/Gestation/Gestation.html / http://www.garthpaine.com/Installation_Works/Gestation/Gestation.html.

Park, L. (n.d.). Eunoia. Retrieved 27th May, 2015, from http://www.thelisapark.com/#/eunoia.

Park, L. (2013). *Eunoia*. [Interactive BCI music system].

Peirce, C. S. (1940). *The Philosophy of Peirce: Selected Writings*. London, UK: Kegan Paul, Trench, Trubner & Co. Ltd.

Planet-H. (2013-present). *G-Stomper Studio*. [Mobile DAW for Android]. Pfäffikon, Switzerland: Planet-H.

Play this Hybrid of an RPG and a Synthesizer in your Browser. (2015). Retrieved 28th May, 2015, from http://www.factmag.com/2015/03/25/synthesizer-roguelike-rpg-browser-game/.

Plohman, A. (2000). Atau Tanaka Global String. Retrieved 13th Jan, 2014, from http://www.fondation-langlois.org/html/e/page.php?NumPage=284.

Prinz, D. (n.d.). *Introduction to Programming on the Manchester Electronic Digital Computer Made by Ferranti Ltd.* Manchester, UK: Ferranti Ltd.

Puckette, M. (1988-present). *Max/MSP*. [Modular visual programming environment software]. Paris, France: IRCAM/San Francisco, CA, USA: Cycling '74.

Puckette, M. (1996-present). *Pure Data*. [Modular visual programming environment software].

Puckette, M. & Lippe, C. (1992). Score Following in Practice. In *Proceedings of the 1992 International Computer Music Conference (ICMC92): International Computer Music Association, San Francisco, CA, USA* (pp. 182-185). San Francisco, CA, USA.

Pyke, M., Tucker, M., Muller, A., Pyke, S., Kerr, A., Shepherd, M., … Elmsly, K. (2013). *1000 Hands*. [Interactive audiovisual gallery installation]. Sheffield, UK: Universal Everything.

Ramley, Z. (2015). Native Instruments Debuts Stems, a New Artist-Oriented Audio Format. Retrieved 29th May, 2015, from http://thump.vice.com/en_uk/article/native-instruments-debuts-stems-a-new-artist-oriented-audio-format?utm_source=thumpfbuk.

Randon, A. (2014-present). *Arpio*. [Arpeggio instrument for Android]. San Francisco, CA, USA: Alexandernaut.

Ratner, L. G. (1991). Topical Content in Mozart's Keyboard Sonatas. *Early Music*, *19* (4), pp. 615-619.

Rayark Incorporated. (2012-present). *Cytus*. [Interactive music game]. Taipei, Taiwan: Rayark Incorporated.

Rayark Incorporated. (2013-present). *Deemo*. [Interactive music game]. Taipei, Taiwan: Rayark Incorporated.

Riddell, A. (n.d.). *Alistair Riddell*. Retrieved 21st Jan, 2014, from http://www.alistairriddell.com.

Riddell, A. (2005). *HyperSense Complex: An Interactive Ensemble*. Paper presented at the 5th Australasian Computer Music Conference (ACMC '05 Generate + Test), Queensland University of Technology, Brisbane, Australia. Retrieved 21st Jan, 2014, from http://www.alistairriddell.com/publications/HC_ACMC05.pdf.

Riddell, A., Langley, S. & Burton, S. (2002-2005). *HyperSense Complex*. [ICMS].

Rogerson, B. (2015). Korg iM1 app brings the M1 workstation to the iPad. Retrieved 28th May, 2015, from http://www.musicradar.com/news/tech/korg-im1-app-brings-the-m1-workstation-to-the-ipad-621901.

ROTOR (n.d.). Retrieved 28th October, 2016, from http://reactable.com/rotor.

Rowe, R. (1992). *Maritime*. [ICMS/composition for violin and electronics].

Rowe, R. (1994). 1 - Interactive Music Systems. In *Interactive Music Systems: Machine Listening and Composing*, 2nd ed. (pp. 1-8). Cambridge, Massachusetts: MIT Press.

Rowe, R. (1999). The Aesthetics of Interactive Music Systems. *Contemporary Music Review*, *18* (3), pp. 83-87. DOI: 10.1080/07494469900640361.

Sandler, S., Windle, J. & Muller, L. (2011-present). *NodeBeat*. [Interactive generative music application]. San Diego, CA, USA: AffinityBlue.

Saussure, F. de. (1916/1959). *Course in General Linguistics* (1st ed./3rd ed., English translation). New York City, New York, USA: The Philosophical Library, Inc.

Schaeffer, P. (1952/2012) *In Search of a Concrete Music* (1st ed./English translation). Paris, France: Editions du Seuil/Oakland, California, USA: University of California Press.

Schneider, N. (2013-present). *Heat Synthesizer*. [Virtual-analogue subtractive synthesizer for Android]. Neuss, Germany: Nils Schneider.

Serra, M. H. (1993). Stochastic Composition and Stochastic Timbre: GENDY3 by Iannis Xenakis. *Perspectives of New Music*, *31* (1), pp. 236-257.

Sharma, R. (2008-present). *BlueStacks*. [Android emulator]. San Francisco, CA, USA: BlueStacks.

Single Cell Software. (2013-present). *Caustic 3*. [Mobile DAW for iOS and Android].

Skrillex. (2014). Doompy Poomp. [Single]. *Recess*. Los Angeles, CA, USA: OWSLA.

Slater, B., Miller, D., Noreau-Hébert, E., Williams, G., Acquaviva, J., Quail, M., … Vezon, G. (2011-present). *Lemur*. [Software emulation of JazzMutant Lemur for iOS and Android]. Schoonhoven, Netherlands: Liine.

Slowik, S. (n.d.). *ring_keyPress*. [Max/MSP external object].

Smalley, D. (1997 [2001 online]). Spectromorphology: explaining sound shapes. *Organised Sound*, *2* (2), pp. 107-126. DOI: 10.1017/S1355771897009059.

Smith, A. (2011). *Ethereal Dialpad*. [Synthesizer for Android]. Santa Cruz, CA, USA: Adam Smith.

Smith-Brindle, R. (1966). *Serial Composition*. Oxford, UK: Oxford University Press.

So Far So Good. (2011-present). *Incredibox versions 1, 2, 3 and 4*. [Interactive music game]. France: So Far So Good.

Solomos, M. (2002). Xenakis' Early Works: From "Bartókian Project" to "Abstraction". *Contemporary Music Review*, *21* (2/3), pp. 21-34. DOI: 10.1080/07494460216657.

Sonami, L. (n.d.). The Lady's Glove, a brief history. Retrieved 13th Jan, 2014, from http://www.sonami.net/works/ladys-glove/.

Sonami, L., DeMarinis, P. & Bongers, B. (1991-2001). *Lady's Glove*. [ICMS]. The Hague/Amsterdam, The Netherlands.

Souster, T. (1968). Xenakis's [sic] 'Nuits'. *Tempo*, (85), pp. 5-18.

STEPSEQUENCER. (n.d.). Retrieved 27th May, 2015, from http://www.schnellebuntebilder.de/#/fourxfour/stepsequencer/.

Sterken, S. (2001). Towards a Space-Time Art: Iannis Xenakis's [sic] Polytopes. *Perspectives of New Music*, *39* (2), pp. 262-273.

Storey, J. (1997). *An Introduction to Cultural Theory and Popular Culture* (2nd ed.). Hemel Hempstead, UK: Prentice Hall/Harvester Wheatsheaf.

Stravinsky, I. (1947). *Poetics of Music in the Form of Six Lessons*. Cambridge, MA, USA: Harvard University Press.

Subatomic Software Audulus. (2014). *Computer Music Magazine*, *201*. p. 102.

Superbrothers, Guthrie, J. & Capybara Games. (2011-present). *Superbrothers Sword & Sworcery*. [Interactive musical exploration game/virtual reality environment]. Toronto, Ontario, Canada: Capybara Games.

Superbrothers Sword & Sworcery. (n.d.). Retrieved 29th May, 2015, from https://play.google.com/store/apps/details?id=com.capybaragames.sworcery&hl=en_GB.

synthhead. (2014a). The Literal Step Sequencer. Retrieved 27th May, 2015, from http://www.synthtopia.com/content/2014/11/01/the-literal-step-sequencer/.

synthhead. (2014b). The Brain-Controlled Art of Lisa Park. Retrieved 27th May, 2015, from http://www.synthtopia.com/content/2014/11/22/the-brain-controlled-art-of-lisa-park/.

synthhead. (2014c). Korg DSN-12 Synth + Sequencer for Nintendo 3DS Now Available. Retrieved 28th May, 2015, from http://www.synthtopia.com/content/2014/09/24/korg-dsn-12-synth-sequencer-for-nintendo-3ds-now-available/.

synthhead. (2014d). Arturia iProphet brings Sequential Circuits Prophet VS Sound to iPad. Retrieved 28th May, 2015, from http://www.synthtopia.com/content/2014/09/30/arturia-iprophet-brings-sequential-circuits-prophet-vs-sound-to-ipad/.

synthhead. (2014e). Photophore brings 'Flock Synthesis' to the iPad. Retrieved 28th May, 2015, from http://www.synthtopia.com/content/2014/12/10/photophore-brings-flock-synthesis-to-the-ipad

synthhead. (2014f). Navichord - A new iPad Instrument for Exploring Chord Progressions. Retrieved 28th May, 2015, from http://www.synthtopia.com/content/2014/09/17/navichord-a-new-ipad-instrument-for-exploring-chord-progressions/.

synthhead. (2014g). LIVKONTROL Turns Your Android Device into an Ableton Live Controller. Retrieved 29th May, 2015, from http://www.synthtopia.com/content/2014/02/12/livkontrol-turns-your-android-device-into-an-ableton-live-controller/.

synthhead. (2015). Korg brings Classic M1 Workstation to iPad. Retrieved 28th May, 2015, from http://www.synthtopia.com/content/2015/05/25/korg-im1-brings-classic-m1-workstation-to-ipad/.

Tanaka, A. (n.d.). Global String. Retrieved 13th Jan 2014, from http://www.ataut.net/site/Global-String.

Tanaka, A. & Toeplitz, K. (1998-2001). *The Global String*. [Interactive computer network music/gallery installation]. Linz, Austria: Ars Electronica Centre.

Taruskin, R. (1982). On Letting the Music Speak for Itself: Some Reflections on Musicology and Performance. *The Journal of Musicology*, *1* (3). pp. 338-349. DOI: 10.2307/763881.

Teengirl Fantasy. (2014). *THERMAL*. [EP]. Carrboro/Durham, NC, USA: Break World Records.

Teengirl Fantasy & 4real. (2014). *http://tgf-thermal.herokuapp.com/.* [Interactive audiovisual website]. Carrboro/Durham, NC, USA: Break World Records/New York City, NY, USA: 4real.

TheBroomInstitute. (2015). *Synthesizer 7DRL*. [Interactive "roguelike" synthesizer/RPG hybrid instrument/game].

The New Reactable Experience (n.d.). Retrieved 17th Mar, 2014, from http://reactable.com/products/experience.

This is the Remix. (2010). *Computer Music Magazine*, *157*, pp. 24-37.

Timpernagel, J., Heinsch, I., Huber, S., Pohle, R., Haberkorn, M., Bernstein, J., … Buether, A. (2013-2014). *STEPSEQUENCER*. [Interactive computer music/gallery installation]. Berlin, Germany: Schnellebuntebilder.

Toch, E. (1948). *The Shaping Forces in Music: An Enquiry into the Nature of Harmony, Melody, Counterpoint, Form* (2nd ed.). New York City, New York, USA: Criterion Music Corporation.

Tucker, M. (2014). 1000 Hands at the Media Space in London. Retrieved 27th May, 2015, from https://www.youtube.com/watch?v=gXOJoXI6EB8.

Waisvisz, M. (2006). The Hands. Retrieved 13th Jan, 2014, from http://www.crackle.org/TheHands.htm.

Waisvisz, M., den Biggelaar, J., Rijnsburger, W., Venmans, H., Cost, P., Demeijer, T., Bongers, B. & Balde, F. (1984-2006). *The Hands*. [ICMS].

Wheatley, J. (2007). The Sound of Architecture. *Tempo*, *61* (242), pp. 11-19.

Wittchow, O. (1998-present). *Nanoloop series*. [Interactive music game for original Game Boy, Game Boy Advance and Android/iOS]. Hamburg, Germany: Oliver Wittchow.

Wolfe, C. (2008-present). *Jasuto*. [Modular synthesis application]. Los Angeles, CA, USA: Chris Wolfe.

Wuorinen, C. (1979). *Simple Composition*. New York City, New York: Longman, Inc.

Xenakis, I. (1953-54). *Metastaseis*. [Score]. London, UK: Boosey & Hawkes.

Xenakis, I. (1955-56). *Pithoprakta*. [Score]. London, UK: Boosey & Hawkes.

Xenakis, I. (1956-57). *Achorripsis*. [Score]. Berlin, Germany: Bote und Bock.

Xenakis, I. (1956-62). *Atrées*. [Score]. Paris, France: Salabert.

Xenakis, I. (1956-62). *Morsima/Amorsima*. [Score]. Paris, France.

Xenakis, I. (1956-62). *ST/10*. [Score]. London, UK: Boosey & Hawkes.

Xenakis, I. (1956-62). *ST/4*. [Score]. London, UK: Boosey & Hawkes.

Xenakis, I. (1956-62). *ST/48-1.240162*. [Score]. London, UK: Boosey & Hawkes.

Xenakis, I. (1956-62). *Strategie*. [Score]. London, UK: Boosey & Hawkes.

Xenakis, I. (1959). *Duel*. [Score]. Paris, France: Salabert.

Xenakis, I. (1960-61). *Herma*. [Score]. London, UK: Boosey & Hawkes.

Xenakis, I. (1963-64). *Eonta*. [Score]. London, UK: Boosey & Hawkes.

Xenakis, I. (1965-66). *Terretektorh*. [Score]. Paris, France.

Xenakis, I. (1966). *Nomos Alpha*. [Score]. Paris, France.

Xenakis, I. (1966). The Origins of Stochastic Music. *Tempo*, (78), pp. 9-12.

Xenakis, I. (1967). *Nuits*. [Score]. Paris, France: Salabert.

Xenakis, I. (1967). *Polytope of Montreal*. [Score]. London, UK: Boosey & Hawkes.

Xenakis, I. (1967-68). *Nomos Gamma*. [Score]. Paris, France: Salabert.

Xenakis, I. (1969). *Persephassa*. [Score]. Paris, France: Salabert.

Xenakis, I. (1969). *Synaphai*. [Score]. Paris, France: Salabert.

Xenakis, I. (1969-70). *Hibiki-Hana-Ma*. [Score]. Paris, France: Salabert.

Xenakis, I. (1970). Towards a Metamusic. *Tempo*, (93), pp. 2-19.

Xenakis, I. (1963/1971/1992). *Formalized Music: Thought and Mathematics in Composition* (1st ed./English translation/revised ed.). Paris, France: Editions Richard-Masse/Bloomington, Indiana, USA: Indiana University Press/Hillsdale, New York: Pendragon Press.

Xenakis, I. (1971). *Persepolis*. [Score]. Paris, France: Salabert.

Xenakis, I. (1972). *Polytope of Cluny*. [Score]. Paris, France.

Xenakis, I. (1974). *Erikthon*. [Score]. Paris, France: Salabert.

Xenakis, I. (1974). *Gmeeorh*. [Score]. Paris, France: Salabert.

Xenakis, I. (1974). *Noomena*. [Score]. Paris, France: Salabert.

Xenakis, I. (1975). *N'Shima*. [Score]. Paris, France.

Xenakis, I. (1976). *Khoai*. [Score]. Paris, France: Salabert.

Xenakis, I. (1977). *The Legend of Er*. [Score]. Paris, France: Salabert.

Xenakis, I. (1978). *Mycenae Alpha*. [Composition/score using UPIC]. Mycenae, Greece: Polytope of Mycenae.

Xenakis, I. & Saint-Jean, P. (1977). *UPIC*. [ICMS]. Paris, France: *Centre d'Etudes de Mathématique et Automatique Musicales* (CEMAMu).

Yorke, T., Godrich, N. & Donwood, S. (2014). *PolyFauna*. [Interactive musical exploration game/virtual reality environment]. Sheffield, UK: Universal Everything & Ticker Tape Ltd.

7. Appendix 1: ScreenPlay Setup/Configuration Instructions

7.1 Software/Technical Requirements

In order to run *ScreenPlay* it is necessary to have both Ableton Live Suite and Max/MSP/Max for Live installed. A 30 day free trial of both can be downloaded from https://www.ableton.com/en/live/max-for-live/. Once the two applications have been installed it will be necessary in Ableton Live to navigate to **Options → Preferences → File Folder → Max Application** in order to define the location of Max/MSP on the computer's hard drive. The installation of TouchOSC on the touchscreen device(s) to be used to interact with the system is also a requirement, which can be found in both the Google Play Store and App Store for Android and iOS respectively.

The accompanying TouchOSC Bridge application should be installed on the central computer system used to run Ableton Live and the *ScreenPlay* Max for Live MIDI Devices (available for Windows and macOS from http://hexler.net/), and initiated prior to opening Ableton Live. Once activated the name of the computer hosting the TouchOSC Bridge software will appear in the TouchOSC application under **Settings → MIDI Bridge**. After selecting the MIDI Bridge host, the same IP address that populates the "Host" text box at the top of the page should be copied into the "Host" text box in **Settings → OSC**. To configure TouchOSC Bridge in Ableton Live's MIDI settings so as to allow *ScreenPlay* to function correctly, navigate to **Options → Preferences → Link MIDI** and enable "Track" for the TouchOSC Bridge MIDI input port, and both "Track" and "Remote" for the TouchOSC Bridge MIDI output port.

The *ScreenPlay* GUI TouchOSC layout files included on the accompanying data CD to the thesis can be loaded onto the touchscreen device(s) used to host the TouchOSC GUIs either wirelessly via the TouchOSC Editor software (also available for Windows/macOS from http://hexler.net/) or by connecting the device(s) to a computer via USB and copying the files over. To open the layouts in TouchOSC navigate to **Layout → Add from Editor/Add from File**.

It should be noted that a strong, reliable and fast Wi-Fi connection is required to run *ScreenPlay* efficiently and ensure the best possible interactive experience. A weak signal may result in a failure to accurately communicate user-input commands and/or represent the current status of the system via the GUI(s), due to a delay/failure in the transmission of

MIDI/OSC messages between TouchOSC and *ScreenPlay-CTRL*. Connecting only the central computer system running Ableton Live and the touchscreen device(s) used to host the TouchOSC GUI(s) to the network can help to minimise the possibility of any such problems occurring; while disconnecting the router from the internet can also be beneficial. Another source of these issues could be the computer's firewall, and it may be necessary to add TouchOSC Bridge as an exception in the firewall's settings in order to avoid this. The video demonstration provided on the accompanying data CD accurately represents the level of performance that can reasonably be expected when interacting with *ScreenPlay* using a standard home wireless router.

7.2 Directions for Setup and Configuration

The setup process for *ScreenPlay* differs depending on whether the system is configured to run in either single or multi mode. However, the very first step applies to both and is imperative to the intended behaviour of the system/GUI.

After initiating TouchOSC Bridge and opening Ableton Live in that order, and before doing anything else, navigate to **Options → Preferences → Record Warp Launch → Launch** and set Default Launch Mode to "Toggle". This ensures that triggering a clip while it is active causes it to stop playing at the next clip trigger quantization step rather than restarting playback from the beginning, which is the default behaviour of all clips in Ableton Live. Note that the change will not apply to any clips already generated with another default launch mode, such as "Trigger", and will need to be made instead via the clip's own Launch Box in the Clip View menu.

7.2.1 Single Mode

1. In single mode two MIDI tracks are required per instrument/part to be controlled via *ScreenPlay*'s GUI. Press ctrl/cmd + shift + T to insert a new MIDI track in session view of the Ableton Live Set.
2. In the first of the two MIDI tracks insert the *ScreenPlay-CTRL* Max for Live MIDI Device. The MIDI From and MIDI To settings for this track should be left as "All Ins"/"All Channels" and "No Output" respectively. Set Monitor to "In".

3. In the second MIDI track insert the *ScreenPlay-GEN* and *ScreenPlay-TRNS4M* Max for Live devices in that order. Following on from *ScreenPlay-TRNS4M* should be the virtual instrument chosen for that particular part, which should be followed, finally, by the *ScreenPlay-FX* Max for Live Audio Effect Rack. The final sequence of the device chain should be as follows: *ScreenPlay-GEN* → *ScreenPlay-TRNS4M* → virtual instrument → *ScreenPlay-FX*.
4. Set the MIDI From and Audio To settings of the second MIDI track to "name of MIDI track containing *ScreenPlay-CTRL*"/"Post Mixer" and "Master" respectively. Set Monitor to "Auto".
5. Ensure that all three Max for Live devices share the same part number and are set to "Single" mode. Although it should not be necessary you may wish to press the "Reset Track ID" button on each of the devices.
6. On the *ScreenPlay-TRNS4M* Max for Live device set the position of the *ScreenPlay-FX* Audio Effect Rack by typing the number of its position in the device chain, starting from 1 for the very first device, into the box provided and press enter - i.e. following the setup instructions exactly would require the number 4.
7. Returning to *ScreenPlay-CTRL*, enable the "IP" button in the top left hand corner of the device and input into the adjacent text box the IP address of the touchscreen device hosting the TouchOSC GUI, which can be found under "Local IP Address" in **TouchOSC** → **Settings** → **OSC**. Then input the outgoing OSC port as defined in **TouchOSC** → **Settings** → **OSC** to the "Port"/"In" text box, and the incoming OSC port as defined in TouchOSC to the "Port"/"Out" text box.
8. Press "Commit". The TouchOSC GUI should update to display the current status of the corresponding instrument/part in the Ableton Live Set.
9. Arm the second of the two MIDI tracks to begin playing notes via the GUI playing surface; or any other MIDI controller so long as the "TouchOSC/Other" button on *ScreenPlay-CTRL* is set to "Other".
10. To add more instruments/parts either follow the steps exactly from the very beginning - remembering to use a different part number for each new instrument/part - or duplicate the existing two MIDI tracks using ctrl/cmd + D and begin again at stage 5 of the setup procedure. Upon completing the setup and configuration process click with the mouse on any clip slot in the Ableton Live Set to avoid any issues with generating "fix length" clips via the GUI. If switching between instruments/parts with

the mouse when in single mode it will be necessary to do so again after selecting the chosen track. No such issue exists with using a MIDI controller to switch between tracks.

7.2.2 Multi Mode

1. In multi mode there are three separate MIDI tracks required for each instrument/part. Begin by loading *ScreenPlay-CTRL* into the first of the three tracks and use the following input/output settings: MIDI From - "TouchOSC Bridge"/"corresponding channel number with TouchOSC GUI for the given part"; MIDI To - "No Output"; Monitor - "In". Of the four TouchOSC GUI layouts provided on the accompanying data CD the MIDI channel for each is defined in the filename. To make more versions using different MIDI channels simply make copies of the existing layouts and alter the assigned MIDI channel of all of the "pads" on the grid-based playing surface in TouchOSC Editor.
2. In the second track load *ScreenPlay-GEN* and use the following settings: MIDI From - "name of track containing *ScreenPlay-CTRL*"/"Post Mixer"; MIDI To - "name of track containing *ScreenPlay-TRNS4M*"/"Track In"; Monitor - "Auto".
3. In the third track load *ScreenPlay-TRNS4M*, the chosen virtual instrument for a particular part, and *ScreenPlay-FX* in that order, and use the following settings: MIDI From - "name of track containing *ScreenPlay-CTRL*"/"Post Mixer"; Audio To - "Master"; Monitor - "In".
4. From here the setup procedure is broadly the same as it is for single mode from stage 5 onwards; although it is not necessary to turn on the "IP" button, and all three devices should be set to "Multi" - it may be necessary to manually press the "Reset Track ID" button on each of the devices after switching to multi mode, though not always. The IP address and incoming/outgoing OSC ports denoted in *ScreenPlay-CTRL* should also be unique to each touchscreen device/TouchOSC GUI used to control the individual instruments/parts. Finally, it is not necessary to manually arm the track containing the virtual instrument in order to play/record notes when in multi mode.

   If the trio of devices is switched back to single mode from multi mode it will be necessary to manually re-enable the "IP" button on *ScreenPlay-CTRL*. It should also be noted that, if any of the devices are moved to a new track after initially being loaded in another, it will be necessary to reset the track ID within the device.

The same setup instructions for single and multi mode as are listed here can be found by pressing the "Help" button on the *ScreenPlay-CTRL* Max for Live MIDI Device. Alternatively, skeleton Ableton Live Sets for both single and multi mode configurations have been included on the accompanying data CD.

8. Appendix 2: ScreenPlay GUI Instructions

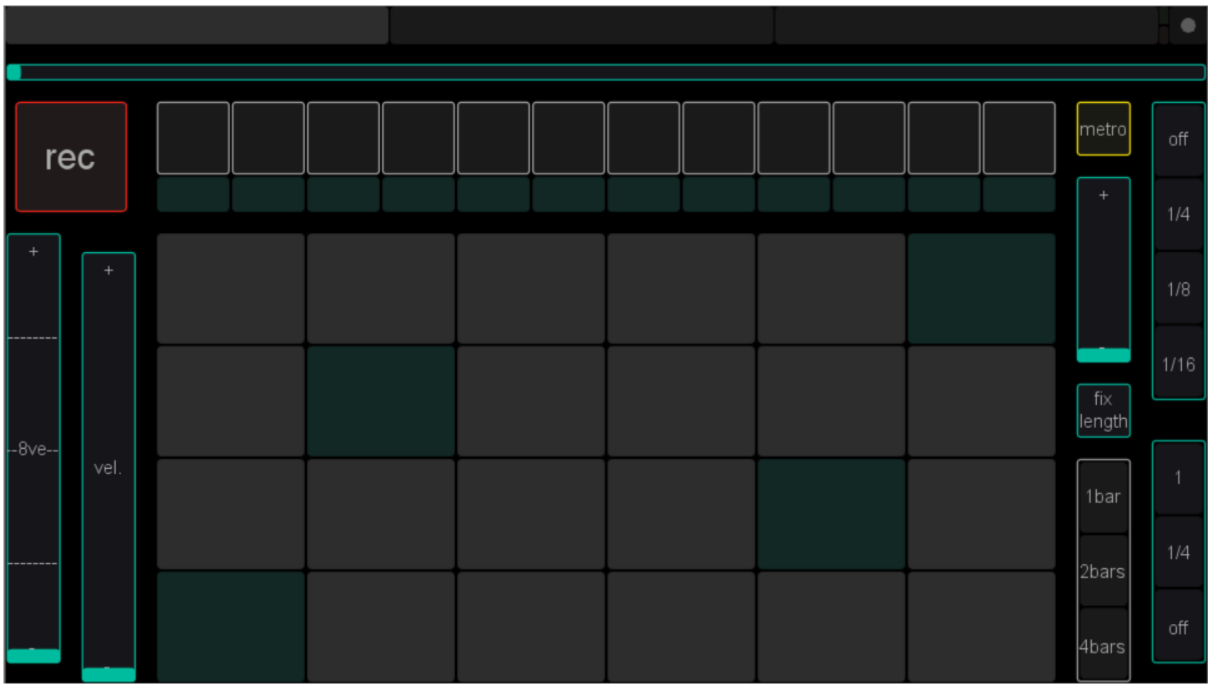


Figure 14. *ScreenPlay* GUI main control page

Along the top of the main control/performance interface page of the GUI is the timebar; the indicator of which moves from left to right over the course of a single bar in 4/4. The colour of the timebar changes depending on the playing/recording state of the clips stored in *ScreenPlay*. If all clips are inactive the timebar appears blue, if a clip is playing it will be green, and if recording is enabled it will be red.

Below the timebar are 12 clip slots which, when unoccupied, appear empty. Generating a new clip can be done in one of two ways, the first of which involves generating a new clip of predetermined length. To do this, begin by enabling "fix length" - located to the right hand side of the grid-based playing surface. Once this is done, touching a vacant clip slot will populate it with a new 1 bar clip. If the generation of a new clip fails the clip slot will revert to its initial empty state. This happens very rarely but, if it does, clicking once with the mouse on any clip slot in the Ableton Live Set on the computer so that it is highlighted should solve the problem - note that this step should be taken at the very end of the system setup and configuration process to ensure there is no issue with generating new clips.

Once a new clip has been generated the loop length selection strip below the "fix length" button will change from grey to blue, indicating that it is now functional, and will also display a loop length selection of 1 bar. The clip length can then be changed to either 2 bars or

4 bars if desired. Turning off "fix length" disables the duration selection strip, and its colour will return to grey and loop length selection be cleared to indicate as such. Likewise, triggering the newly generated clip from the GUI by touching the clip slot for a second time also disables the loop length selection strip, before activating the clip at the next step in the chosen clip trigger quantization value. Recording can be enabled at any time (either before or after a clip is generated/triggered) using the "rec" button to the left of the clip slots. In single mode the track of the selected instrument/part in the Ableton Live Set must be armed in order to play/record notes. Note that *ScreenPlay* implements overdub recording, meaning that new notes are added to the clip as it loops without overwriting those already contained within it.

The second method of generating a new clip can be used to record a clip of indeterminate length, although it differs slightly between single- and multi-user modes. To do so in single mode begin by disabling "fix length". So long as the track in which clips are stored for the selected instrument/part in the Ableton Live Set is armed, touching any empty clip slot will now begin recording a new clip at the next clip trigger quantization step, without having to first enable record on the GUI. Triggering on the GUI the newly generated clip for a second time will stop the recording process according to the clip trigger quantization value and loop back to the start of the clip, automatically beginning playback. Disabling record on the GUI before the clip has been triggered for a second time will not stop the process of recording.

In multi mode the process is broadly the same, although it is necessary to activate record on the GUI in addition to disabling "fix length" before triggering an empty clip slot. Also, in this instance, triggering the newly generated clip for a second time will still cause it to loop back to the start and begin playback, but it will not disable recording. This instead will need to be done via the "rec" button on the GUI, which, if disabled before the new clip has been triggered for a second time, will either crop or extend the clip to the nearest step as defined by the clip trigger quantization value. This difference in operational protocols for single and multi mode is simply by virtue of the differing ways in which it is necessary to handle recording in the two modes, and so long as the visual feedback provided via the GUI is used to monitor the current status of the system there should be no confusion surrounding the differences between the two.

Once populated by a clip, triggering a clip slot on the GUI changes the status of that clip from inactive to active and vice versa in accordance with the clip trigger quantization value. The currently active clip will appear blue on the GUI, while all others appear grey. In order to ensure all clips respond as expected the default launch mode in the Ableton Live preferences must be set to "toggle"; something which should have been done during the initial setup and configuration of the system in line with the instructions provided. The row of buttons positioned directly below the 12 clip slots on the GUI are used to delete the clips inside the corresponding clip slots.

To the right of the 12 clip slots is a yellow button labeled "metro", which controls the metronome in the Ableton Live Set. Below the "metro" button is a vertical +/- slider, which controls the tempo. To the right of the "metro" button and tempo slider is the record quantization selection strip; the options for which are "off", 1/4, 1/8 and 1\16. Below this and to the right of the loop length selection strip can be found the clip trigger quantization selection strip; the options for which are "off", 1/4 and 1 bar.

Positioned centrally and occupying the majority of the main control/performance interface page is the grid-based playing surface, which is locked in the key/signature scale denoted by the key signature/scale selection matrix, with root-notes being coloured blue. Note that chromatic scales contain more notes overall and therefore less root-notes are displayed at once. When a diatonic key signature/scale is selected any standard triad within that scale can be played using the same hand shape anywhere on the grid. To do this think of making a triangle between the index finger, middle finger and ring finger. The index finger and ring fingers should occupy the same horizontal row of "pads"/notes but be separated by a single "pad"/note. The middle finger can then be used to play the "pad"/note directly above the one which separates the index and ring fingers, thus completing the triangle.

To the left of the playing surface are two vertical sliders labeled "vel" and "8ve", which are responsible for controlling the output velocity of all notes on the playing surface and the octave range of the notes currently displayed. In single mode the octave and velocity settings are both global, as is the key signature/scale. In multi mode all three are part-specific. Metronome, tempo, clip trigger quantization and record quantization values are always global.

While multi mode enables up to 16 individual users to control a specific instrument/part within the musical output of the system via dedicated individual GUIs, single mode allows for up to 16 different instruments/parts to be controlled from a single GUI - providing the track in which clips are stored for any given part is selected inside Ableton Live. If using a mouse to change between instruments/tracks ensure to click on any clip slot within that track after selecting it to avoid issues with generating "fix length" clips. No such action is required if switching between tracks using a MIDI controller. The visual feedback provided via the GUI will update in accordance with the currently selected track.

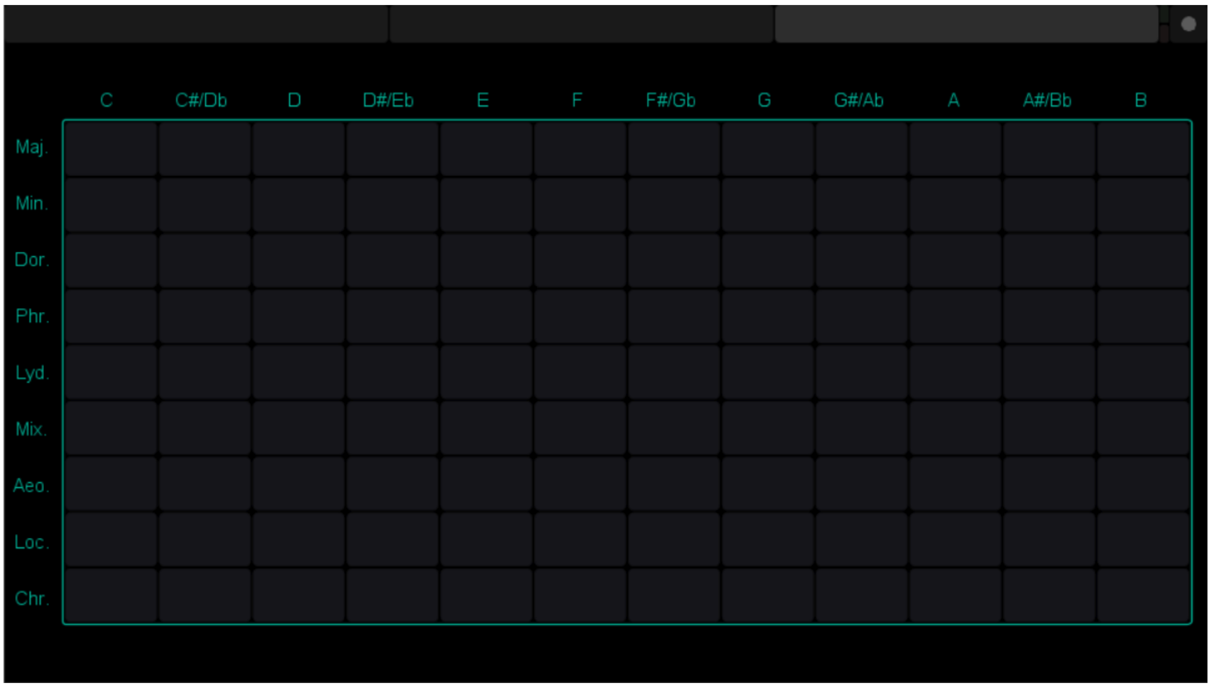


Figure 15. *ScreenPlay* GUI key selection matrix

The key signature/scale selection matrix is located on the final page of the GUI, and can be accessed by touching the corresponding tab at the top of the GUI display above the timebar. The available scales are: Major, Minor, Dorian, Phrygian, Lydian, Mixolydian, Aeolian, Locrian, Chromatic. The default setting when first initiating the system is C Major.

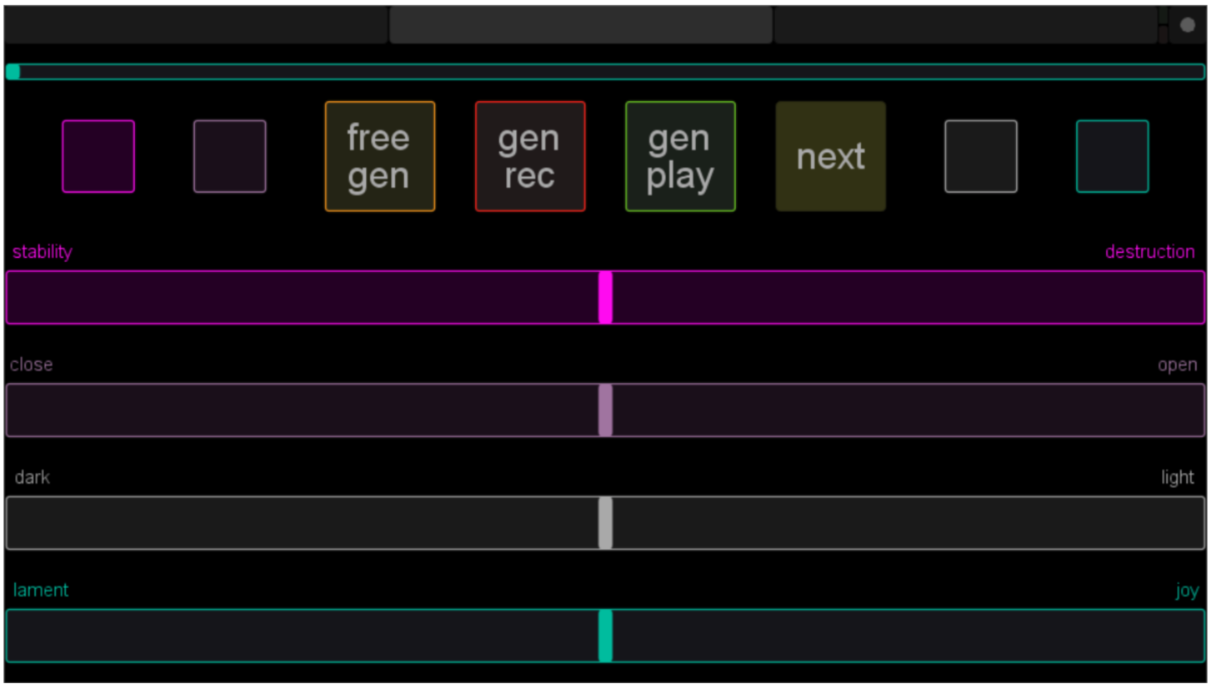


Figure 16. *ScreenPlay* GUI generative/transformative algorithm control page.

The generative/transformative algorithm control page can be accessed by touching the second tab at the top of the display (above the timebar). Positioned at the top and centrally are 4 buttons labeled "free gen", "gen rec", "gen play" and "next", all of which are used to control the Markovian generative algorithm. Activating "gen rec" when a clip is playing will collect all the note information contained within that clip during a single entire play through, after which the clip is automatically disabled and generated playback from the algorithm begins immediately. The "gen rec" button will turn off and the "gen play" button will turn on to indicate as such, while the timebar will also change from green to blue to indicate that all clips are currently inactive. Triggering a clip while "gen play" is enabled will switch the playback source at the next clip trigger quantization step and vice versa, updating the GUI accordingly.

Enabling "free gen" before "gen rec" allows for notes to be freely played and recorded into the system to act as the source material for the Markovian generative algorithm, rather than being collected from an active clip. "Gen rec" must then be manually disabled by the user before activating "gen play". Touching the "next" button restarts the generation of musical material in the event that it reaches a point from where it cannot continue.

The four horizontal sliders and colour-corresponding buttons at either side of the controls for the Markovian generative algorithm are responsible for controlling the topical opposition transformations; each of which has a varying effect on the musical output of the system depending on the position of the slider in relation to the two extremes of the topical opposition. The "stability-destruction", "open-close" and "light-dark" oppositional transformations impact the texture/timbre of the musical output, while the "joy-lament" transformation affects the melodic and rhythmic contour of a clip.

Although the "joy-lament" transformation can be applied to polyphonic/harmonic material it is designed primarily to work with monophonic melodic material, and adhering to these guidelines will help yield better, more musically coherent results. For the transformation to work on polyphonic/harmonic material all notes inside the clip must be quantized. Note also that any transformations should be carried out in the same key signature/scale as the target clip - or in a chromatic scale for more experimental transformations - to ensure a successful result.

The colour-coded buttons at the top of the GUI each enable/disable the corresponding topical opposition transformation. The "stability-destruction", "open-close" and "light-dark" sliders can all be moved freely while the transformation is enabled in order to modify the sound. Moving the "joy-lament" slider after the transformation has been triggered dynamically changes the probabilities used to calculate the results of the transformation so as to accurately represent the position of the slider at any given moment.

9. Appendix 3: Patch Documentation

9.1 Notes on Max

In the Max programming language [objects] carry out functions as denoted by their internal arguments/settings when receiving a bang or upon receiving a "message". "Messages" contain commands that relate to the functionality of [objects] and override any default internal arguments/settings within an [object]. A bang is a fundamental "message", of which the literary term has no significance, and is used to trigger the action of an [object] or the output of a "message".

It has been necessary to significantly scale down in size the images of the programming logic presented in the figures throughout the patch documentation appendix. However, all of the annotations are included in the final Max for Live MIDI Devices, which can be opened by clicking the leftmost button of the three in the top right hand corner of the devices when loaded into a MIDI track in Ableton Live.

When being referred from one section of the patch documentation appendix to another it is recommended that these instructions are followed in order to gain a better overall understanding as to the order in which the numerous processes happening simultaneously in each of the three Max for Live MIDI Devices that make up *ScreenPlay* take place,

9.2 ScreenPlay-CTRL

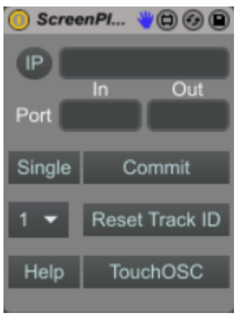


Figure 17. *ScreenPlay-CTRL* Max for Live MIDI Device.

ScreenPlay-CTRL is the primary Max for Live MIDI Device with which the TouchOSC GUI communicates when interacting with the system. It is responsible for actions such as: transposition of the MIDI input to facilitate the key-locking of the playing surface on the GUI;

creating, deleting, triggering and altering the length of clips; recording input; altering tempo, key signature and quantization value etc. Due to the size of the patch and the arbitrary nature of many of the functions, this documentation will focus purely on the procedures that are directly involved in the process of generating musical output from the system, such as those listed above. There is a large amount of programming dedicated solely to feeding back the information resulting from changes made by the user/other users to the GUI in terms of enabling/disabling toggles, colour changes of objects in relation to playing/recording status etc., which will not be covered. The same is true of *ScreenPlay-GEN* and *ScreenPlay-TRNS4M*.

9.2.1 Main patcher/ScreenPlay-CTRL 1.0

The programming logic within the main patcher is separated into a number of distinct sections. The majority of the functionality that visibly occurs at this level generally serves either to feed necessary data and information to the many sub-patchers in order for them to carry out their respective tasks, or is in some way involved in the procedures that occur during the setup and calibration of the system after all three *ScreenPlay* Max for Live MIDI Devices and the *ScreenPlay-FX* Effect Rack have been loaded into the Ableton Live Set.

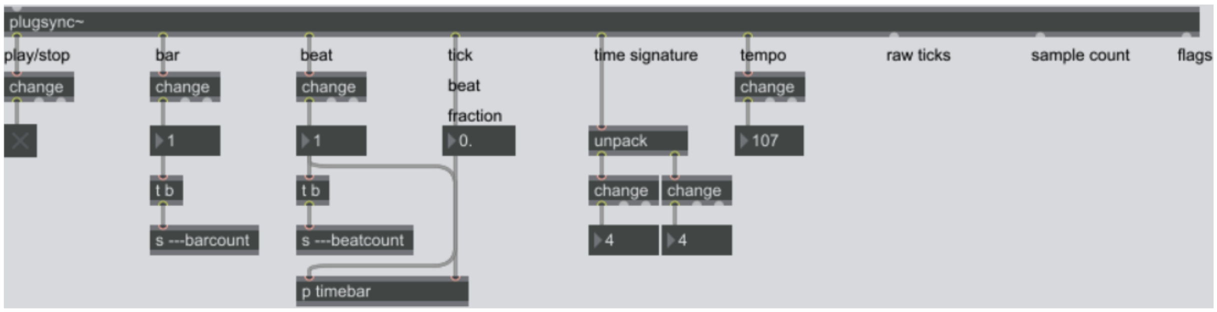


Figure 18. [plugsync~] object in main patcher window.

In the top right hand corner of the window is a [plugsync~] object, which is linked to the transport inside Ableton Live and relays all the data associated with it out of its numerous outlets. The beat and bar counts are sent via [s ---beatcount] and [s ---barcount] ([s] is an abbreviated version of [send], which pairs with [r]/[receive] and removes the need to use patch cords for transmitting messages between objects) respectively to various locations in the patch to be used in a number of different processes. The beat and bar count both trigger a bang from [t b] (abbreviation of [trigger bang]) that is sent via the [send] objects in place of the actual integer values outputted from [plugsync~]. The tick and beat counts also enter [p timebar], which uses the two values to drive the timebar on the TouchOSC GUI.

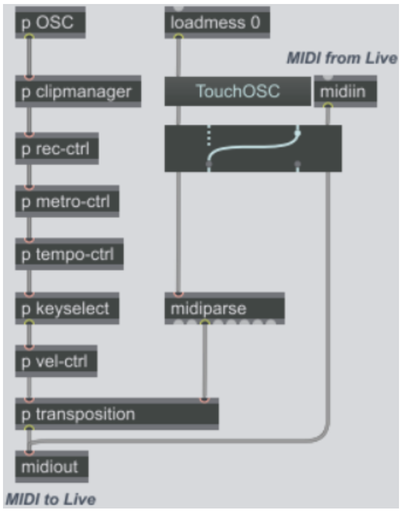


Figure 19. Core high-level programming in main patcher window.

To the left of [plugsync~], and positioned fairly centrally in the main patcher window, can be found the core high-level programming of *ScreenPlay-CTRL* where the vast majority of procedures take place. Both MIDI and OSC information sent from/to the TouchOSC GUI arrive at and are outputted from the device via [midiin]/[midiout] and [p OSC] ([p] is an abbreviation of [patcher], which contains a sub-patch) respectively. All other control operations such as clip creation and control, enable/disable recording, enable/disable metronome, tempo control, key selection, velocity control and key locking/transposition also occur here. The TouchOSC/Other button serves to bypass the key selection and transposition features and outputs any incoming MIDI information without interference in the event that the user wishes to interact with *ScreenPlay-GEN* and *ScreenPlay-TRNS4M* via an interface/MIDI controller of their choosing - note that all other control functionality afforded by the TouchOSC GUI is still enabled in this scenario.

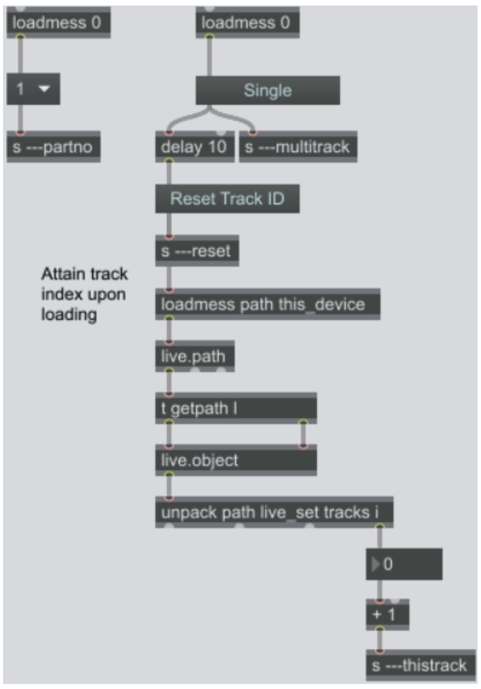


Figure 20. *ScreenPlay-CTRL* Max for Live device setup controls.

Further left still are the Max for Live device controls to switch between single- and multi-user modes and reset track ID. The programming connected to the "Reset Track ID" button ascertains not the ID of the track in Ableton Live within which *ScreenPlay-CTRL* is located but instead of the adjacent track. This is because, if the setup and configuration instructions have been followed correctly, the clip slots and clips that will be targeted by the device in response to the input of the user will be those contained within the next track along in the Ableton Live Set. Because of the complicated nature of the processes involved with using *ScreenPlay* it is necessary to utilise two tracks per instrument/part in single-user mode and three per user/GUI in multi-user mode. The bang released by the "Reset Track ID" button and the "0/1" released by the "Single/Multi" user button (0 for single, 1 for multi) are also directed elsewhere in the patch to fulfil other functions via [s ---reset] and [s ---multitrack] respectively. Changing from single to multi mode and vice versa also causes the track ID to be reset.

In the top left hand corner of the patcher is a drop-down menu used to select the instrument/part number in single mode and user/GUI number in multi mode. All three *ScreenPlay* Max for Live devices display the same option on their GUI, which ensures cross-communication between the three separate devices for each individual part is properly managed and controlled as appropriate to both single and multi modes.

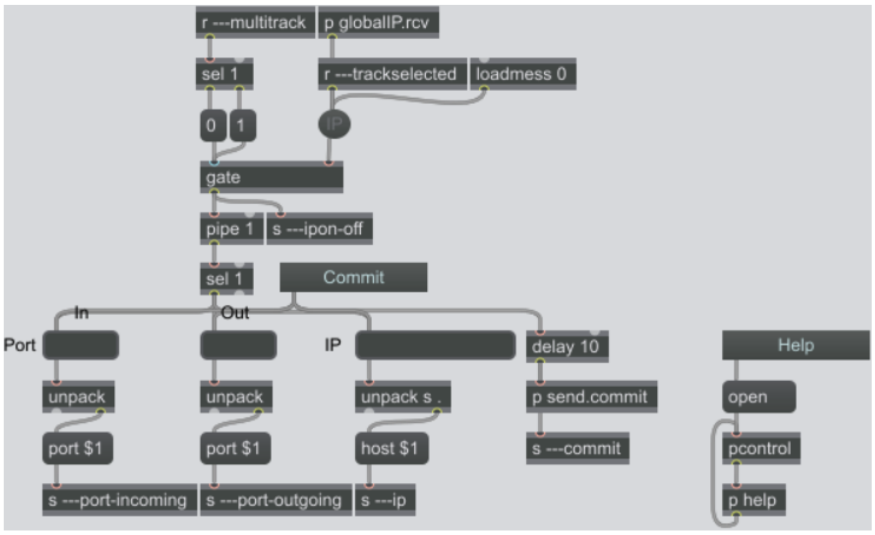


Figure 21. *ScreenPlay-CTRL* Max for Live device setup controls.

At the bottom of the patcher is the "Help" button, which opens a floating window in the Ableton Live Set ([p help]) that displays setup and configuration instructions for both single and multi modes. There are also text boxes into which the IP address of the touchscreen device hosting the TouchOSC GUI for a particular instrument/part, and the OSC input and output ports denoted in the TouchOSC settings, should be entered during setup and configuration of the system. Once the "Commit" button is pressed the contents of each of the three text boxes is sent via [s ---port-incoming], [s ---port-outgoing] and [s ---ip] into [p OSC] (main patcher → [p OSC]), where it is used to assign the appropriate IP address and OSC ports to both [udpsend] and [udpreceive].

Above the "Commit" button can be seen a circular "IP" button, which, when activated/illuminated, indicates in single mode that all three Max for Live devices for the selected instrument/part are currently active. This can be controlled manually by the user by switching the IP button on/off, or happens automatically when selecting the track in the Ableton Live Set containing the instrument/clips of a particular part. The visual feedback displayed on the TouchOSC GUI is also updated to display the current state of the chosen part when selecting it. In multi mode the "IP" button serves no purpose and can be either enabled or disabled.

9.2.2 [p transposition]

Main patcher → [p transposition]

It is inside [p transposition] that the MIDI input from the TouchOSC GUI is processed to ensure that the correct notes are applied to each pad in respect to the chosen key signature/scale and octave.

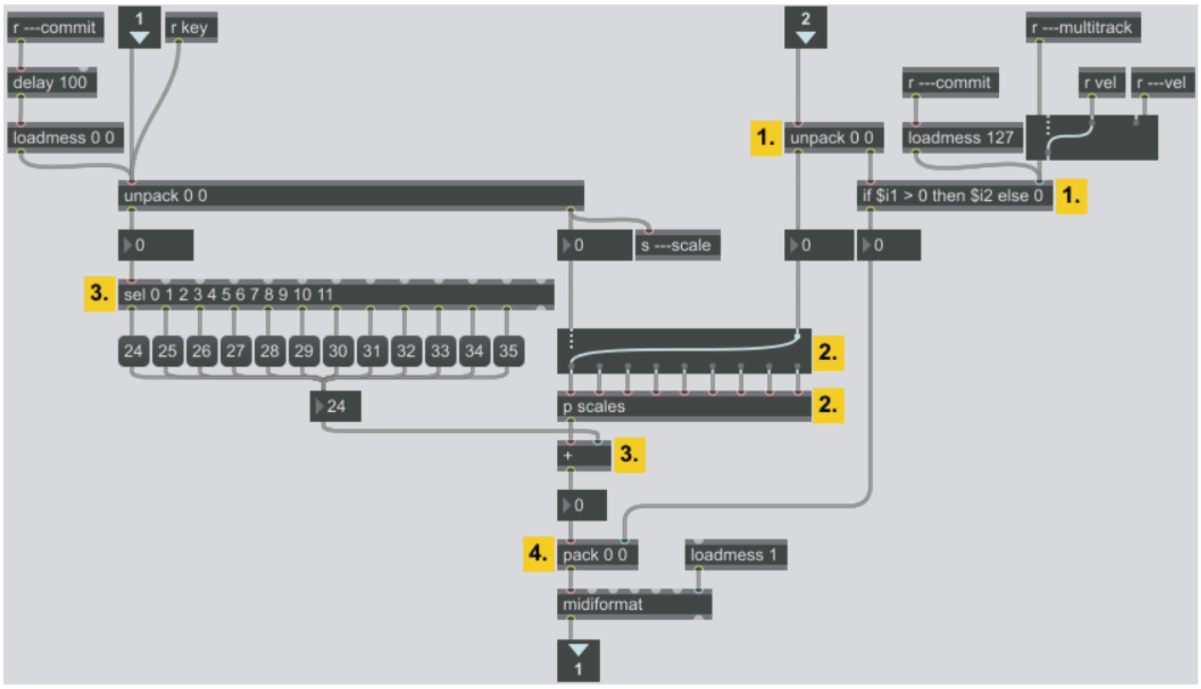


Figure 22. [p transposition].

1. The pads on the playing surface of the TouchOSC GUI output MIDI control change numbers ranging from 1-15 for Android or 1-21 for iPad - note that the pitch layout allowing for a triad to be played with the same hand shape in any position on the grid when utilising the key locking functionality of the *ScreenPlay* interface means that there is more than one of each note present on the grid. Upon arriving at [inlet] 2 of [p transposition] after being separated from the entire incoming MIDI message by a [midiparse] object in the main patcher window, the control change number and its value are separated from each other by an [unpack] object. The control change number goes on to be replaced with the correct pitch after passing through [p scales]. The control change value is replaced with the velocity value set by the user via the "vel" slider on the GUI before being paired back together with the new pitch value. The [if $i1 > 0 then $i2 else 0] object into which the control change value enters after passing through the right outlet of [unpack 0 0] compares its value to 0 (in the Max

programming language $ indicates a variable value) and, if found to be greater, outputs the value arriving from [r vel] (single mode) or [r ---vel] (multi mode) at its right inlet; otherwise outputting a 0. [r vel]/[r ---vel] receive the value of the "vel" slider on the TouchOSC GUI from [p vel-ctrl] (main patcher → [p vel-ctrl]). Once the appropriate value is released from [if] it enters the right inlet of [pack 0 0] at (4) and awaits the arrival of the pitch value with which it will be paired before exiting the patcher and, ultimately, being released from the Max for Live device via [midiout] in the main patcher window.

The difference between [r vel] and [r ---vel] is that, when the Max for Live device is first loaded into the Ableton Live Set, the "---" in [r ---vel] is replaced by three randomly generated integers specific to the device which are shared by all [send] and [receive] objects in the device with the same unique identifier. This means that, in multi mode, velocity control is specific to each individual instrument/part. The same is true also of key signature/scale and octave. In single mode, however, communication between [send] and [receive] objects without "---" preceding their unique identifier is possible with other devices in the Ableton Live Set, meaning that velocity, key signature/scale and octave become global parameters applied to all instruments/sounds.

2. After being released from the left outlet of [unpack 0 0] at (1) the control change number enters the right inlet of a [gswitch2] object and is outputted through one of nine outlets, depending upon the chosen scale, before entering [p scales] through one of nine inlets. The position of [gswitch2] is set by the integer value arriving at its left inlet, which first arrives as the second item in a two element list at [inlet] 1 of the patcher from [p keyselect] (main patcher → [p keyselect]). The two integer values contained in the list are released from [p keyselect] in response to the position of the selection toggle on the key selection matrix on the TouchOSC GUI, with the first value corresponding to the selected key as denoted by the horizontal position of the toggle and the second corresponding with the scale as denoted by the vertical.

   [p scales] contains eight more sub patchers - one for each diatonic scale made available by *ScreenPlay* - and a single [- 1] object to subtract 1 from the incoming control change number when a chromatic scale is selected. Each of the diatonic scale patchers contains a [select] object with arguments ranging from 1-36 (despite the control change numbers of the pads on the GUI never exceeding 21), to the outlets of the which are connected 36 individual messages containing each interval of the corresponding scale for that patcher up to five octaves. Upon being released from one of the eight diatonic scale patchers,or having passed through [- 1] for a chromatic scale, the correct interval value then enters the left inlet of a [+] object where either 0, 12, 24, 36 or 48 is added to it to result in a final interval value appropriate for the octave range selected by the user via the "8ve" slider on the TouchOSC GUI, which is then outputted from the patcher. The octave value arrives via [r 8ve]/[r ---8ve] from [p 8vercv] (main patcher → [p OSC] → [p 8vercv]).

3. After being released from [p scales] the pitch interval enters a [+] object where it is added to the root note of the chosen key, as denoted by the horizontal position of the selection toggle on the TouchOSC GUI key-selection matrix. The root notes start from MIDI note number 24 (C2), meaning this is the lowest possible note available when interacting with *ScreenPlay*.
4. The final pitch value then enters the left inlet of [pack 0 0] where it is paired with the velocity value released from the [if] object at (1), before being released from the patcher and then out of the Max for Live device into Ableton Live via [midiout] in the main patch window. Note that there is no output from [pack 0 0] or [+] until the arrival of the value at their left inlet. This is because, for the vast majority of objects in Max (although not all), the left inlet is known as the "hot" inlet and causes the object to act upon receiving information, whilst the other inlets are known as "cold" and do not activate the object. This differentiation can be seen due to the colour difference between hot (red) and cold (blue) inlets.

9.2.3 [p rec-ctrl]

Main patcher → [p rec-ctrl]

The responsibility of enabling and disabling recording lies with [p rec-ctrl] and, because of the limitations of Ableton Live, the process is not all that straightforward. Ableton Live is designed as a composition, production and performance software environment for a single user/musician. *ScreenPlay* allows for up to sixteen separate users to collaboratively interact with the system simultaneously when in multi mode - each of which is afforded individual control over when to enable/disable recording for their specific part/instrument. The result of this is the need to activate and deactivate recording in an unorthodox manner when multi mode is enabled.

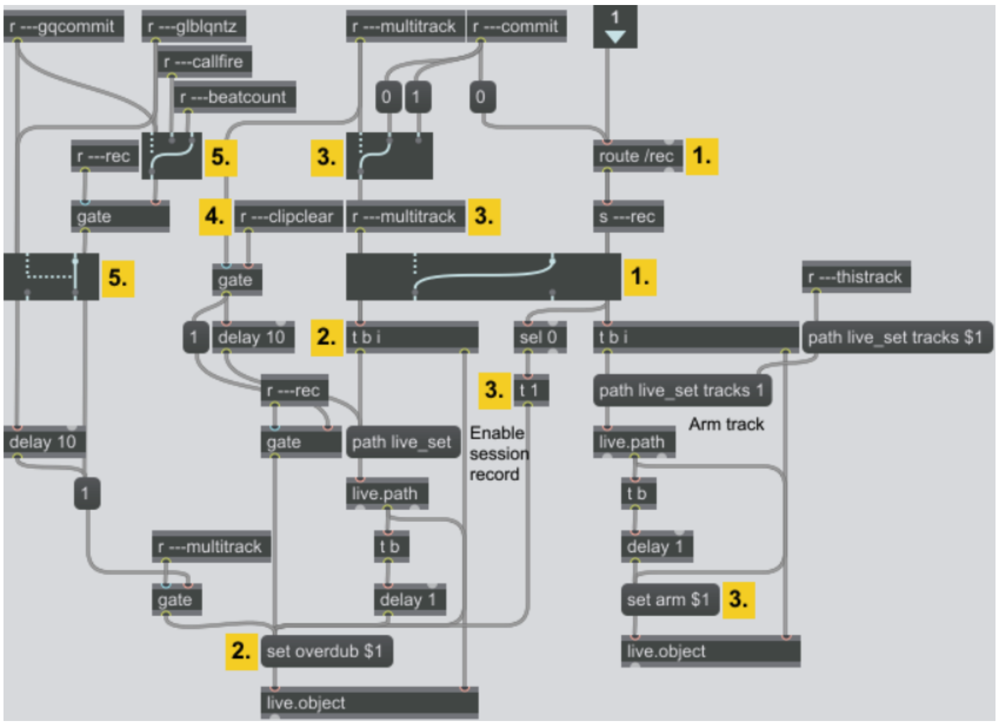


Figure 23. [p rec-ctrl].

1. After arriving through the [inlet] of the patcher the output from the "rec" button on the TouchOSC GUI (either a 1 or 0) first enters the right inlet of a [gswitch2] object. The [gswitch2] serves to direct the incoming value through one of two programming chains depending on whether single or multi mode is enabled. Its position is set by the value arriving at its left inlet from [r ---multitrack] and will output the value from "rec" on the TouchOSC GUI through its left outlet when single mode is enabled and its right when multi mode is enabled.
2. In single mode the value enters [t b i], which, in turn, releases the value from its right outlet followed by a bang from its left. The "rec" value is used to populate the variable value in the "set overdub $1" message that then enters the left inlet of [live.object]. The "set overdub" function instructs Ableton Live to enable/disable session record (recording of MIDI input into clips in session view) depending on whether or not it receives a 1 or a 0. If recording has been enabled by the user once before whilst interacting with the system the arrival of the function message at [live.object] immediately causes the resulting action. If it is the first time the user has enabled "rec" the action does not occur until the bang released from the left outlet of [t b i] has

triggered the "path live_set" message, which, in turn enters [live.path] and causes the output from its middle outlet of the target destination for the "set overdub" function, which is sent into the right inlet of [live.object]. The output from [live.path] then causes a bang to be released from [t b] that triggers the function message again, thus causing the action to occur.

3. If multi mode is enabled the value received from "rec" on the TouchOSC GUI via the [inlet] of the patcher exits [gswitch2] at (1) via its right outlet. The programming of the function that follows is identical to that triggered in single mode only its target is to arm/disarm the specific track in Ableton Live upon which the *ScreenPlay* devices are acting. In addition, enabling multi mode causes the value arriving at [r ---multitrack] at the top of the patcher to switch the input of the connected [gswitch] object from the left inlet to the right. The result is that, when the user presses the "Commit" button on the Max for Live device GUI after setting up the system, the bang arriving at [r ---commit] triggers a "1" that is sent into [t b i] at (2), which is connected to the left outlet of [gswitch2] at (1), and causes session record to be activated. In single mode the "0" triggered by the bang from [r ---commit] has the opposite effect. The output of [r ---multitrack] itself is also routed into [t b i] at (2) to ensure session record is enabled/disabled as appropriate for the multi/single mode setting.

   Disabling record in multi mode, as well as deactivating track arm, also re-enables session record when [sel 0] connected to the right outlet of [gswitch2] sends a bang in response to receiving a "0" and, in turn, causes [t 1] to send a "1" into the "set overdub $1" message. This is necessary because the act of disarming a track in Ableton Live has the knock-on effect of disabling session record. In order to enable multiple users to have independent control over when they start and stop recording of their individual parts in Ableton Live, session record must be continuously enabled, and the responsibility of starting/stopping recording instead falls to arming/disarming the target track of each of the parts. Allied with the way in which the user is directed to setup the system when using multiple interfaces - using three tracks per part/interface - this makes for a viable workaround of Ableton Live's typical behaviour.

4. Another trait of Ableton Live is to disable session record whenever a clip is deleted. As a result it is necessary when in multi mode to re-enable it in the event that a clip is deleted whilst "rec" is activated on the TouchOSC GUI, so as to avoid disrupting the

recording process for any other players who may have activated "rec" at the same time via their own GUI. The bang received from [r ---clipclear] triggers a "1" that enters the right inlet of a [gate] - which will only be opened upon receiving a 1 from [r ---rec] at its left inlet - before passing into "set overdub $1". To ensure the action is carried out the bang is then delayed by 10 ms when passing through [delay 10] before triggering the function via "path live_set". If "rec" is disabled [r ---rec] will receive a "0" and the attached [gate] will be closed.

5. Finally, Ableton Live also automatically disables session record whenever an active clip stops playing; therefore it is also necessary to re-activate it when this happens. The clip trigger quantization value received from both [r ---glblqntz] whenever there are any changes made to it by the user and [r ---gqcommit] after the "Commit" button on the device GUI has been pressed to finalise the setup process is used to select the input source of a [gswitch] object into which the outputs of [r ---callfire] and [r ---beatcount] are directed, and the outlet of a [gswitch2] object through which the data later passes.

   If clip trigger quantize is disabled the bang received from [r ---callfire] sent at the moment a clip is deactivated by the user is allowed to pass through [gswitch] via the left inlet. If recording is currently enabled it then passes through a [gate] object before entering the aforementioned [gswitch2]. The bang then passes through the left outlet of the [gswitch2] and, after being delayed by 10 ms to allow time for initial action of stopping the clip to take place, triggers a "1" that enters "set overdub $1" after passing through a [gate] (which is only open when multi mode is enabled) and reactivates session record.

   If clip trigger quantization is set to either 1 beat or 1 bar the bang received every beat from [r ---beatcount] passes through the right inlet of [gswitch] and, after passing through the [gate], out of the right outlet of [gswitch2] before triggering the "1" that reactivates session record. This ensures that session record is not re-enabled until after the clip has stopped at the beginning of either the next beat or next bar.

9.2.4 [p clipmanager]

Main patcher → [p clipmanager]

It is inside [p clipmanager] that all actions relating to the creation, deletion and control of clips occurs. There are a number of different sub-patchers, all of which fulfil a specific task: [p quantizesettings] is responsible for altering both the clip trigger quantization and record quantization values in accordance with the instructions relayed via the TouchOSC GUI from the user; [p cliptrig] takes care of creating and starting/stopping clips; [p clipclear] deletes clips; [p looplength] alters the length of newly created clips.

9.2.5 [p cliptrig]

Main patcher → [p clipmanager] → [p cliptrig]

Contained within [p cliptrig] are four more sub-patchers: [p getclipids], which ascertains the ID reference for the first twelve clip slots in the track being targeted by *ScreenPlay-CTRL*; [p clipcheck], which ascertains whether or not a clip slot already contains a clip when triggered in order to decide whether a new one should be created or the existing one should be fired, as well as checking if any clip slots contain clips upon loading the device/pressing the "Commit" button so as to relay the information to the TouchOSC GUI as visual feedback; [p newclip], which creates a new clip in an empty clip slot; and [p clipfire], which fires a clip in an already populated clip slot.

9.2.6 [p getclipids]

Main patcher → [p clipmanager] → [p cliptrig] → [p getclipids]

Upon first loading *ScreenPlay-CTRL* into an Ableton Live set, as well as when the "Commit" button on the device GUI is pressed at the end of the setup process, the track number of the track in which the clips will be stored is received from the main patcher window via [r ---thistrack], from where it enters "path live_set track $1". The resulting message is used to populate the message box below without causing it to output its contents by entering through the right inlet. The bang released from [loadbang]/[r ---reset], which arrives via the [inlet] of the patcher when "Commit" is pressed, then triggers the output of the newly created message, which, in turn, activates the "get clip_slots" function. The list outputted from [live.object] is unpacked and the ID integers of the first twelve clip slots in the track are sent via individual [send] objects into [p newclip].

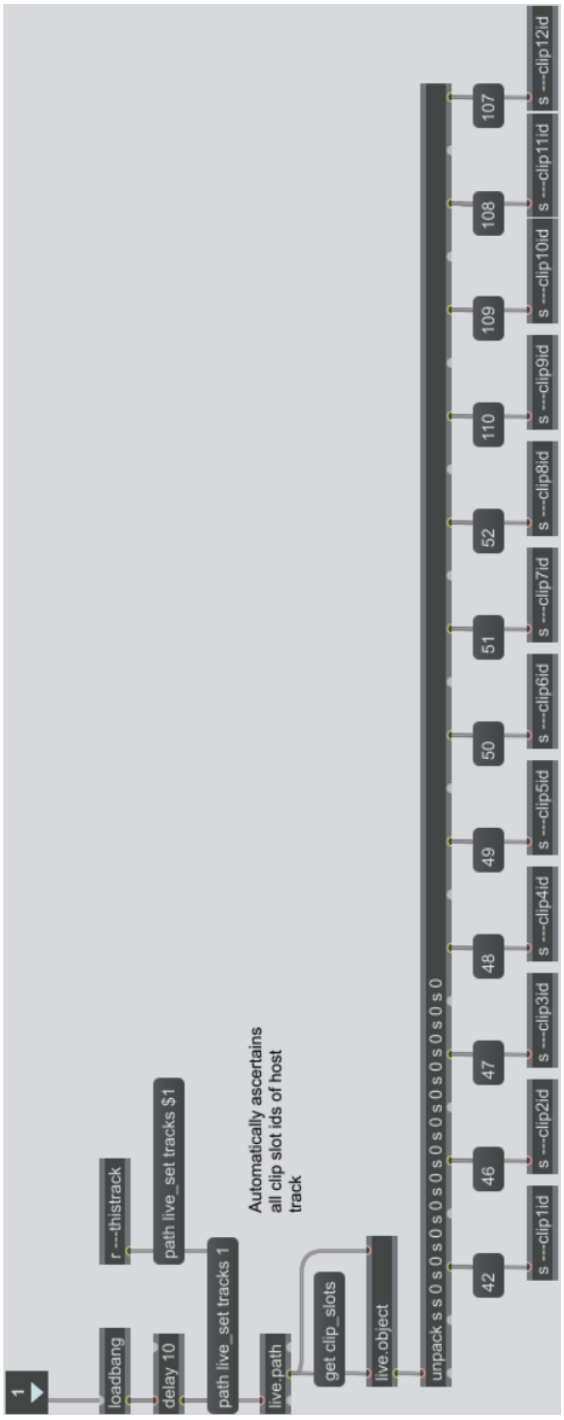


Figure 24. [p getclipids].

9.2.7 [p clipcheck]

Main patcher → [p clipmanager] → [p cliptrig] → [p clipcheck]

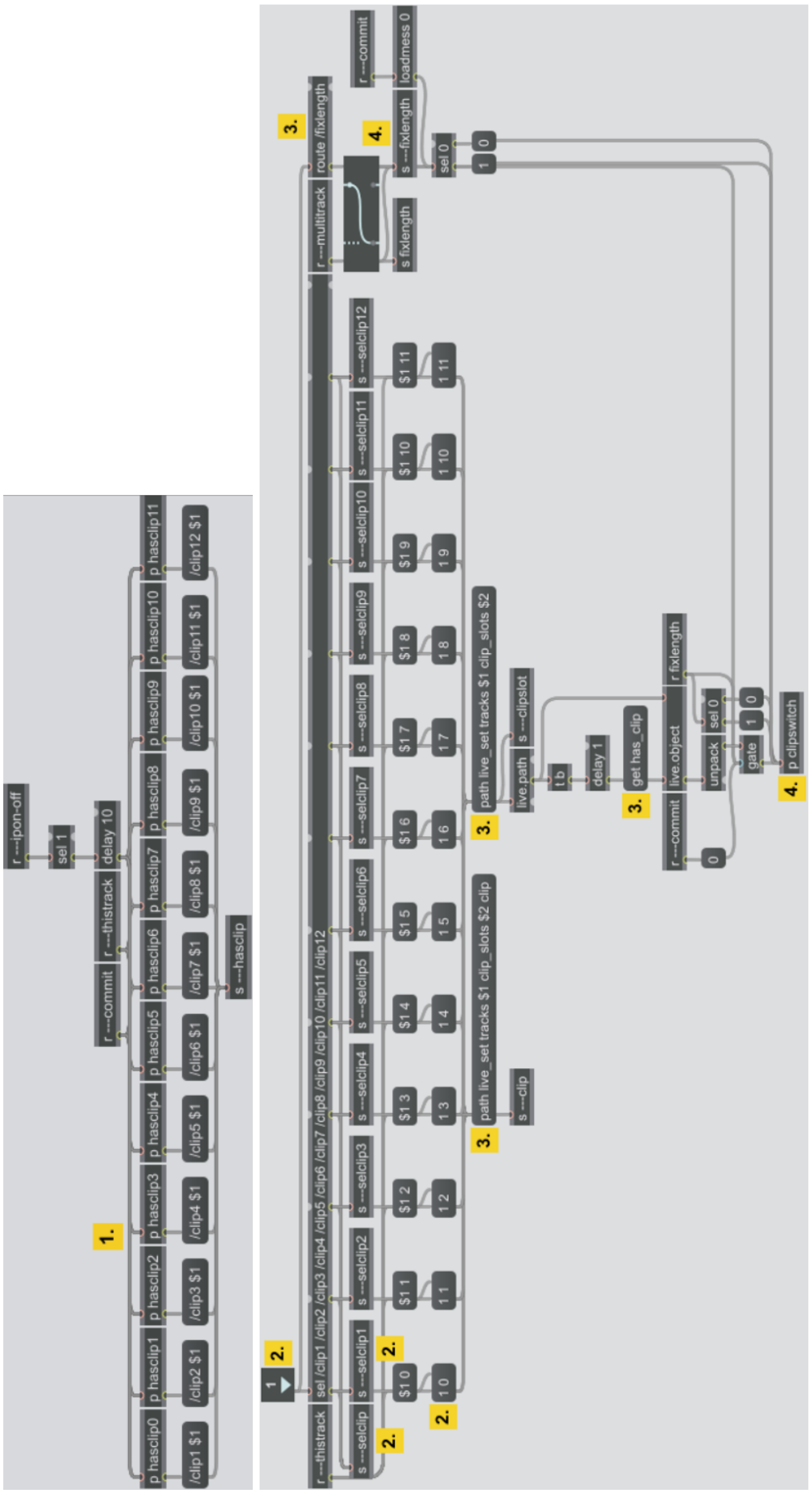


Figure 25. Two adjacent sections of [p clipcheck].

1. When the *ScreenPlay-CTRL* device is first loaded, the "Reset Track ID" or "Commit" buttons are pressed on the device GUI, or the "IP" button is enabled when in single mode either manually or as a result of selecting the target track in Ableton Live of *ScreenPlay-CTRL*, the track ID of the target track is received via [r ---thistrack] and sent into twelve different [p hasclip*n*] patchers - one for each of the clips available on the TouchOSC GUI. Each of the patchers ascertains whether the clip slot corresponding with the number in the patcher name currently contains a clip or not and outputs a 1 or 0 accordingly. The 1/0 outputted from each of the patchers then passes through a "/clip*n* $1" message before being sent via [s ---hasclip] into [p OSC] (main patcher → [p OSC]), where all the messages are sent out to the connected TouchOSC GUI to provide the user with a visual representation of any pre-existing clips.
2. Whenever a clip on the TouchOSC GUI is triggered by the user (either on or off) a bang is released from the [select] object connected to the [inlet] of the patcher via the corresponding output to the clip that has been triggered. The bang is first directed into the left inlet of a message box, which has been populated via its right inlet by the message box above with a two-integer list denoting the target track number in the Ableton Live Set - which arrives via [r ---thistrack] and replaces the $1 variable - followed by the clip slot number to be triggered. The bang is then sent via [s ---selclip*n*] into [p clipswitch].

   As well as being sent via their own dedicated [send] objects into [p clipswitch], the bangs released from every outlet of the [select] object are also sent to [p clipswitch] and [p cliploadedOSC] (main patcher → [p OSC] → [p cliploadedOSC]) via [s ---selclip]. [p cliploadedOSC] ensures that all of the TouchOSC toggle objects used as a visual representation of the clip slots in Ableton Live remain on (filled) when a the corresponding clip in the Ableton Live Set contains a clip. This is necessary to counteract the standard behaviour of the toggle objects, which are designed to simply switch from being on to off and vice versa every time they are touched/pressed by the user.
3. After being released from the message boxes at (2) the "track number; clip slot" lists pass through both the "path live_set track $1 clip_slots $2 clip" and "path live_set track $1 clip_slots $2" messages. The first of these messages, which denotes the specific clip that has been triggered, is sent via [s ---clip] into [p looplength]. The

second message, which denotes the clip slot containing the clip that has been triggered, is sent via [s ---clipslot] into [p clipfire], [p cliploadedOSC] and [p clipclearOSC] (main patcher → [p OSC] → [p clipclearOSC]) - the function of which is the opposite to that of [p cliploadedOSC], whereby it ensures that, if a clip in the Ableton Live Set is deleted by the user, the corresponding toggle object on the TouchOSC GUI used to represent that clip slot is switched off (emptied).

The clip slot message also triggers the "get has_clip" function for that particular clip slot. The result of the function - released from [live.object] as a two-integer list consisting of "clip slot number; 1 = has clip/0 = no clip" - enters the right inlet of a [gate] after being separated from the clip slot number by [unpack]. The [gate] will be open if the "fix length" button on the TouchOSC GUI is turned on and closed if it is turned off - as instructed by the 1/0 value sent to its left inlet from [route /fixlength]/[r fixlength] whether in multi or single mode respectively.

4. After passing through the [gate] the 1/0 value outputted from [live.object] enters [p clipswitch], along with the 1/0 value triggered by the [sel 0] object connected to [route /fixlength]/[r fixlength], which acts to swap the two values around (i.e. 0 triggers "1", 1 triggers "0"). The value outputted from [route /fixlength] is also sent to a number of different sub-patchers in the patch via [s ---fixlength]; two of which are [p looplength] and [p looplengthclrOSC] (main patcher → [p OSC] → (p looplengthclrOSC]). [p looplength] is responsible for changing the length of a newly generated clip. [p looplengthclrOSC] is responsible for changing the colour of the loop length selector on the TouchOSC GUI from grey to blue when it becomes active and back to grey again once changing the settings will no longer have an impact on the length of the current clip. This is one of the many perceived affordances implemented in the design of the GUI to indicate as best as possible the functionality of the system to the user.

9.2.8 [p clipswitch]

Main patcher → [p clipmanager] → [p cliptrig] → [p clipcheck] → [p clipswitch]

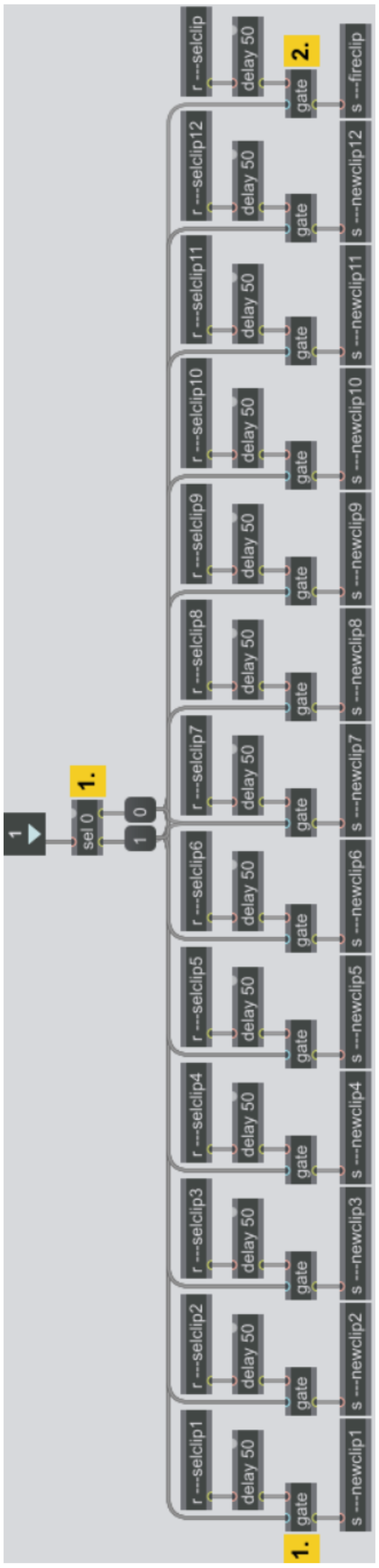


Figure 26. [p clipswitch].

1. Upon arrival inside [p clipswitch] the 1/0 value from the "get has_clip" function and the inverse of the 1/0 values outputted from [route /fixlength]/[r fixlength] at (3) in [p

clipcheck] are first directed into [sel 0], which triggers a "1" if the value is 0 (no clip in triggered clip slot/fix length on) or a "0" if the value is 1 (clip in triggered clip slot/fix length off). This value is then directed into the left inlet of one of twelve [gate] objects, through which the bang received via the [r ---selclip*n*] objects from [p clip check] when the corresponding clip to each of the objects on the TouchOSC GUI is triggered by the user is directed after being delayed by 50 ms. The [gate] will be open and allow the bang to pass through if "fix length" is enabled by the user AND if there is no clip currently in the clip slot at the time it is triggered. Either if "fix length" is disabled OR there is already a clip in the the clip slot at the time it is triggered by the user the [gate] will be closed and the bang will not be able to pass through. In the event that the bang is able to pass through the [gate] it is sent via [s ---newclip*n*] to [p newclip] where a new clip is generated in the empty clip slot.

2. The 1/0 value arriving at the patcher is also sent directly into the left inlet of the [gate] object through which the bang received via [r ---selclip] from [p clipcheck] whenever any of the clips on the TouchOSC GUI are triggered by the user is sent after being delayed by 50 ms. In this instance the [gate] will be open if "fix length" is disabled - regardless of whether the clip slot is already populated with a clip at the time it is triggered by the user - and closed if "fix length" is enabled AND the clip slot is empty when it is triggered by the user. If the bang is allowed to pass through it is sent via [s ---fireclip] into [p clipfire], from where the specific clip slot triggered by the user is fired.

9.2.9 [p newclip]

Main patcher → [p clipmanager] → [p cliptrig] → [p newclip]

1. Once the bangs sent by the individual [send] objects in [p clipswitch] are received at [r ---newclip*n*] they trigger the attached message, each of which is populated with the corresponding clip slot ID number received from [p getclipids] via [r ---clip*n*id]. The message then triggers the "set highlighted_clip_slot id $1" function, which acts to highlight the clip slot that has been triggered by the user.
2. The bang from [r ---newclip*n*] is then delayed by 50 ms before triggering the function on the left to generate a new 1 bar clip in the highlighted clip slot. Once the function has been completed the output from [live.object] causes a bang to be released from [t

b], which, in turn, is sent via [s ---newclip] into [p cliploadedOSC] (main patcher → [p OSC] → [p cliploadedOSC]) in order to ensure the toggle object on the TouchOSC GUI that corresponds with the clip remains on until the the clip is deleted, or - if for some reason the clip generation function has failed - to make sure the user is aware of this by automatically turning the toggle object off again. The bang is also sent to [p looplength] and [p looplengthclrOSC] (main patcher → [p OSC] → [p looplengthclrOSC]) to allow the user the choose the length of the newly generated clip and change the colour of the loop length selector on the GUI from grey to blue to indicate that it is now active.

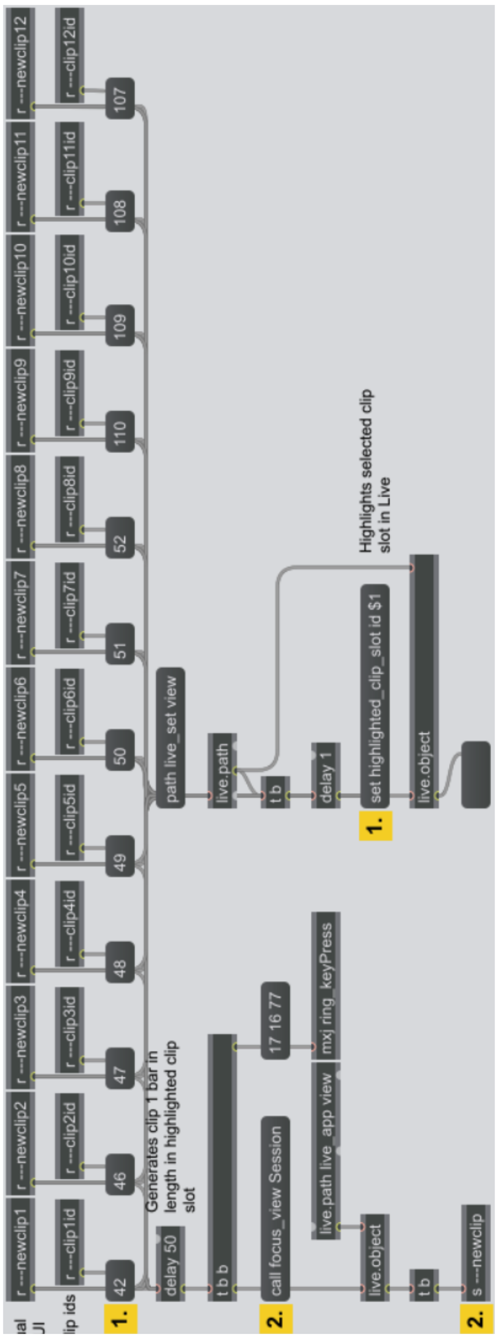


Figure 27. [p newclip] (LongestJourney, 2011; Slowik, S., n.d.).

9.2.10 [p clipfire]

Main patcher → [p clipmanager] → [p cliptrig] → [p clipfire]

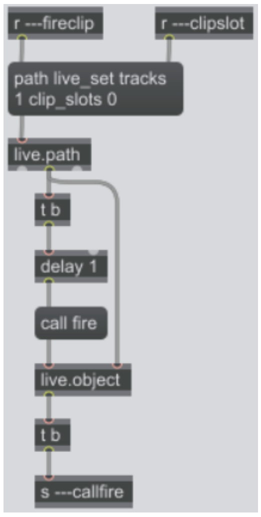


Figure 28. [p clipfire].

Upon receiving a bang from [p clipswitch] via [r ---fireclip] the connected message box containing the Live path to the clip slot triggered by the user sent from [p clipcheck] via [s ---clipslot] triggers the "call fire" function, and thus fires the corresponding clip slot in the Ableton Live Set with the clip triggered by the user on the TouchOSC GUI. Once the function is complete a bang is sent to various locations in the patcher via [s ---callfire] in order to complete a variety of tasks, such as changing the colour of the loop length selector back to grey from blue to show that it is no longer functional in the scenario that a new clip has been generated in an empty slot by the user before being fired after the loop length has been set.

If the clip slot is fired when it is empty as a result of "fix length" being disabled a new clip will begin recording in the empty clip slot - providing the user has enabled recording as instructed in the GUI instructions - at the next clip trigger quantization step, thus allowing users to record clips of any length they wish rather than only being able to choose between one, two and four bars as is allowed after generating a new clip using "fix length".

9.2.11 [p looplength]

Main patcher → [p clipmanager] → [p looplength]

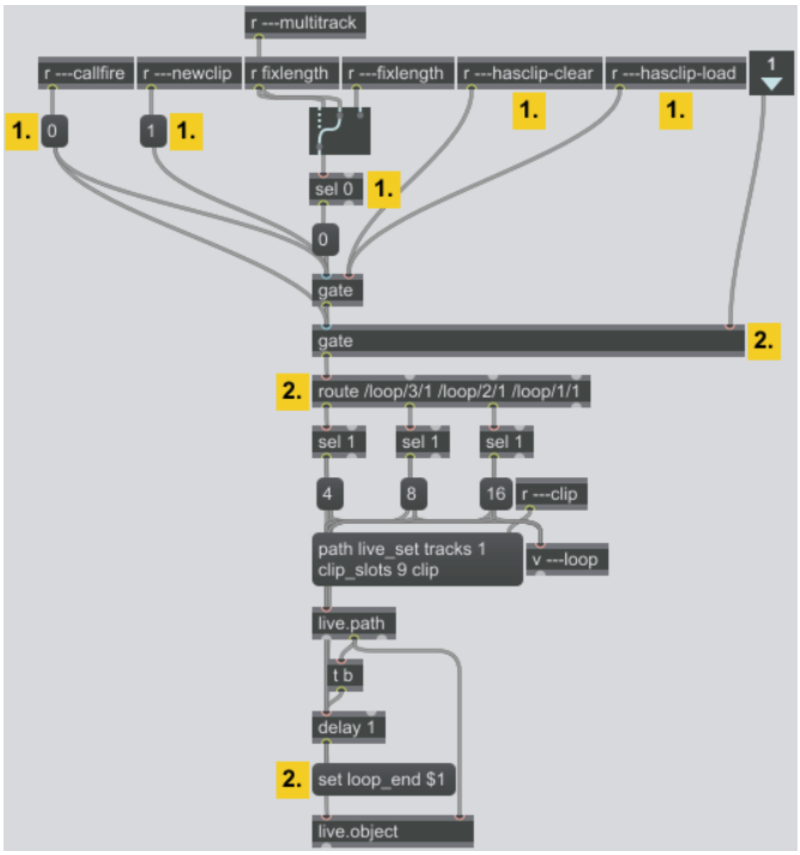


Figure 29. [p looplength].

1. The bang sent from [p newclip] via [s ---newclip] when a new clip is generated in an empty clip slot is received inside [p looplength] and is used to trigger a "1", which then opens a [gate]. Through the [gate] are directed the values arriving from [p clipclearOSC] and [p cliploadedOSC] (main patcher → [p OSC] → [p clipclearOSC]/[p cliploadedOSC]) via [r ---hasclip-clear] and [r ---hasclip-load] respectively, which, upon being released from the [gate] are then directed into the left inlet of a second [gate] in order to control whether or not it is to be opened or closed. The primary purpose of [p clipclearOSC] and [p cliploadedOSC] is to ensure the visual feedback provided to the user through the TouchOSC GUI accurately represents the current state of the Ableton Live Set at any given moment in time - specifically relating to the presence/absence of clips in the first twelve clip slots of the track. Here, after a clip has been successfully generated by the user, which opens the first [gate], a 1 will arrive via [r ---hasclip-load] and pass through the first [gate] into the left inlet of the second, thus opening it. If the current clip slot is cleared by the user after being

generated in an empty clip slot without having been fired the 0 arriving via [r ---hasclip-clear] to indicate that the current clip slot has been successfully cleared passes through the first [gate] and into the left inlet of the second, thus closing it.

If the clip is fired by the user after being generated in an empty clip slot then the bang received at [r ---callfire] triggers a "0" that closes both the first and second [gate]s. Likewise, if "fix length" is disabled by the user the 0 received via [r fixlength]/[r ---fixlength] depending on whether single or multi mode is enabled triggers a "0" and closes both [gate]s.

2. It is through the second [gate] that the incoming OSC data from the TouchOSC GUI is directed after entering the patcher via the [inlet]. The purpose of the extensive gating system is to control the times at which the the length of the clip can be altered by the user via the loop length selector on the TouchOSC GUI. Only after a clip has been generated in an empty clip slot with "fix length" enabled and before the clip has been fired can the length be altered. By default all clips are generated with a length of 1 bar but can be changed to either 2 or 4. As has been mentioned previously, in order to create clips of different lengths to 1, 2 or 4 bars the user can record a clip with "fix length" disabled. As has also previously been mentioned [p looplengthclrOSC] (main patcher → [p OSC] → [p looplengthclrOSC]) changes the colour of the loop length selector on the GUI to indicate to the user when it is activated (blue) and deactivated (greyed-out).

   Once the OSC data from TouchOSC has passed through the [gate] it triggers either a "4", "8" or "16" in accordance with the loop length selection of the user of 1, 2 or 4 bars, which, in turn, triggers the "set loop_end $1" function when entering the following message box. The integer value relating to the chosen loop length is also stored in [v ---loop] (abbreviation of [value], which is an object used to store a single value at a time that can be outputted from any number of instances of the object with the same unique identifier upon receiving a bang) and recalled in [p looplengthclrOSC] when selecting a particular instrument/part in single mode either by enabling the "IP" button on the device GUI or selecting the corresponding track in Ableton Live in order to update the visual feedback presented on the TouchOSC GUI.

9.2.12 [p clipclear]

Main patcher → [p clipmanager] → [p clipclear]

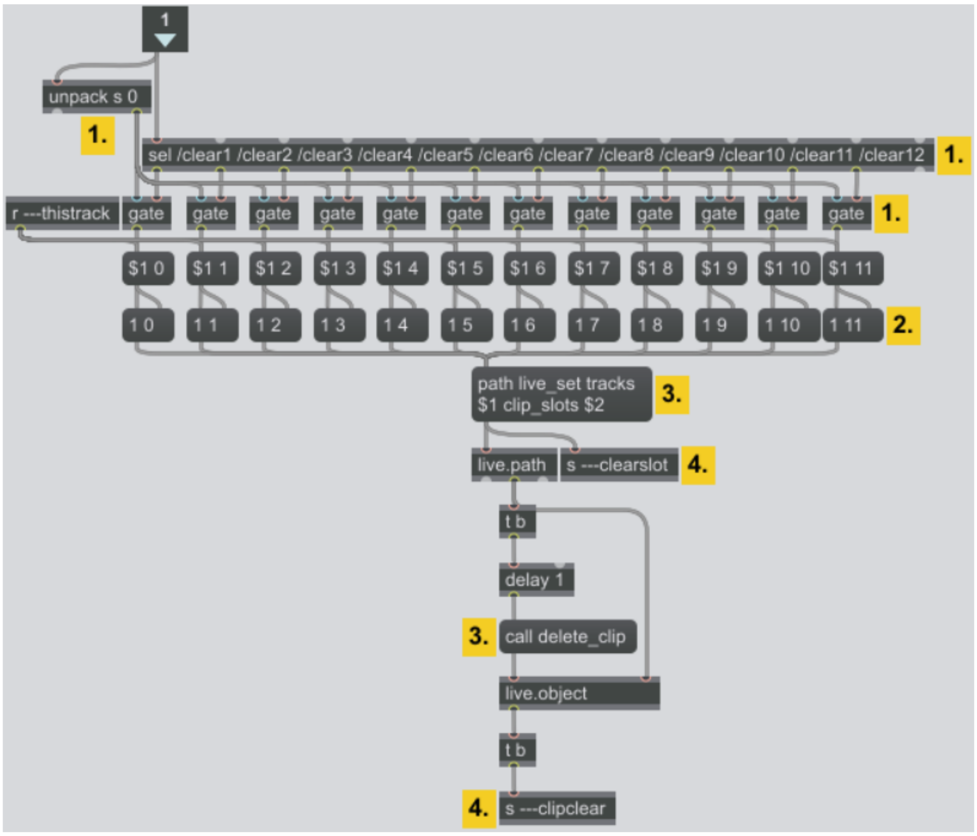


Figure 30. [p clipclear].

1. Each of the clip slots on the TouchOSC GUI is accompanied by a button positioned directly below to delete the currently stored clip. Each of the clip clear buttons has a unique OSC identifier of /clear*n* and an on/off value of 1/0. When one of the clear buttons is triggered by the user the integer value 1 sent from TouchOSC into the Max patch is separated from the identifier of that particular clear button and sent to the left inlet of all the [gate] objects connected to the [select] object in order to open them. The [select] object then recognises the unique identifier and releases a bang through the appropriate outlet when the user releases the clip clear button, which then passes through the opened [gate].
2. The bang triggers the connected message containing a two integer list of "track number; clip slot number", which has been populated via its right inlet by the message above upon the arrival at that message of the track number from [r ---thistrack], which replaces the $1 variable and outputs the message.

3. The message then enters the "path live_set tracks $1 clip_slots $2" message, replacing the variable values and triggering the "call delete_clip" function for that clip slot.
4. A bang is released from [t b] in response to the output from [live.object] upon completion of the function and sent via [s ---clipclear], along with the "path live_set tracks $1 clip_slots $2" message via [s ---clearslot], into [p clipclearOSC] (main patcher → [p OSC] → [p clipclearOSC]) so that the toggle object on the TouchOSC GUI representing the clip slot can be turned off to let the user know the clip no longer exists. The bang sent via [s ---clipclear] is also sent to (4) inside [p rec-ctrl] so that, if recording is currently enabled by the user, it can be switched back on immediately in the Ableton Live Set after being turned off by the software in response to the clip being deleted.

9.2.13 [p quantizesettings]

Main patcher → [p clipmanager] → [p quantizesettings]

1. It is inside [p quantizesettings] that the clip trigger quantization and record quantization settings are received from the TouchOSC GUI and relayed to the Ableton Live Set. The incoming OSC information arrives via the [inlet] of the patcher and, after passing through one of two [route] objects (one for clip trigger quantization and one for record quantization) in order to filter out all OSC messages not directly associated with the task fulfilled by the patcher, triggers a bang from the appropriate [sel 1] object for the given quantization value set by the user. This bang is then directed into the appropriate message box containing the integer used to denote that particular quantization value. Upon being released from the message box the integer triggers the "set clip_trigger_quantization $1"/"set midi_recording_quantization $1" function. Although the integer messages feeding the two functions present all the possible values for the two quantization settings made available in Ableton Live, the options afforded to the user by the TouchOSC GUI are limited to none, 1/4 notes and 1 bar for clip trigger quantization and none, 1/4, 1/8 and 1/16 for record quantization.

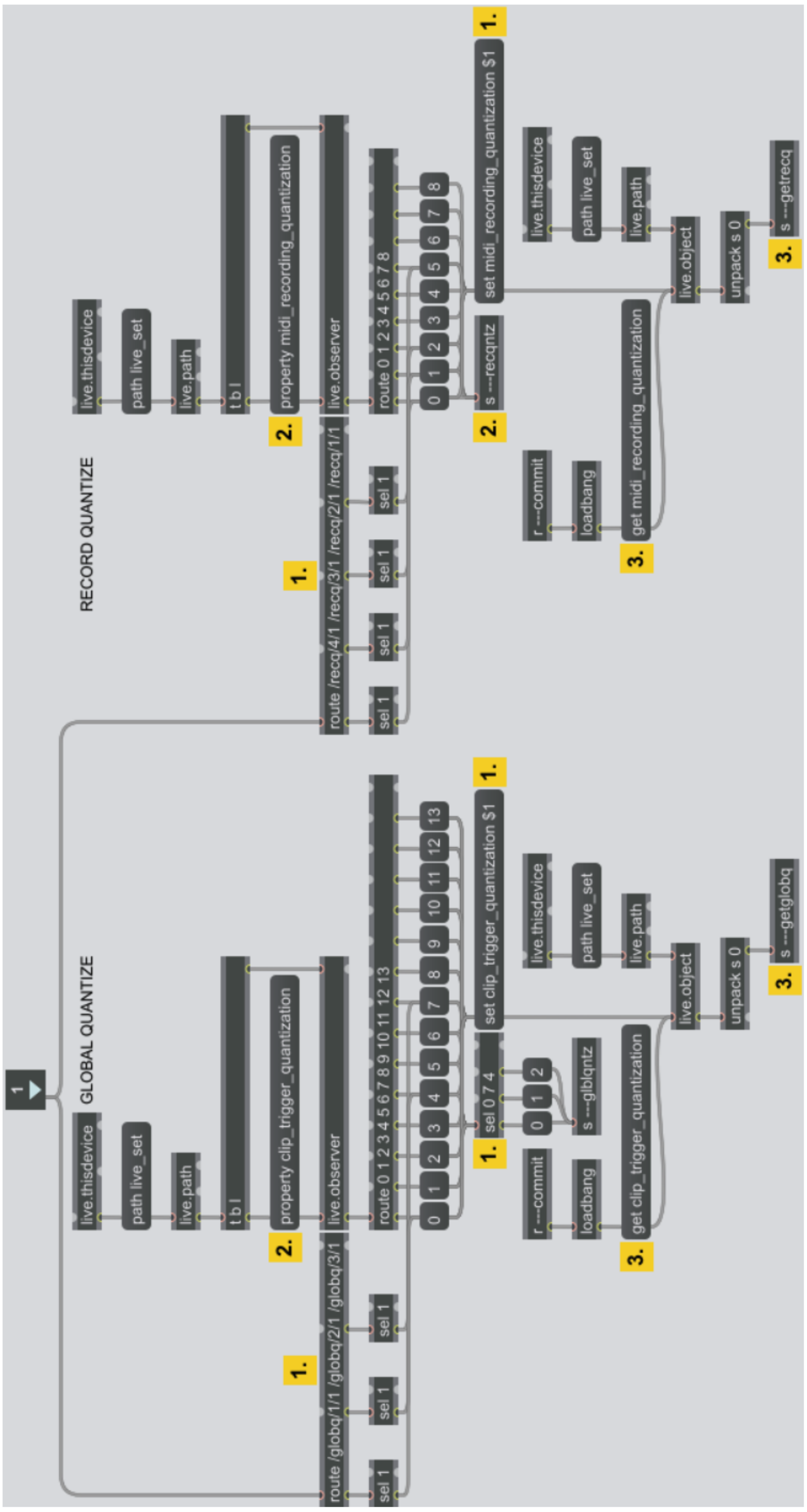
   In the case of clip trigger quantization the integer messages that are triggered by the clip trigger quantization selector on the GUI are also directed into a [select] object where they trigger a "0" for no quantization, "1" for 1/4 notes and "2" for 1 bar.

This value is then sent to various other sub-patchers to be used in other processes which have been previously discussed.

2. Upon loading the *ScreenPlay-CTRL* Max for Live device the "property clip_trigger_quantization" and "property midi_recording_quantization" observation functions are activated. The output of [live.observer] is passed through a [route] object, from where it triggers the appropriate message below to denote the current quantization value. The triggered integer message is sent to [p q-ctrlOSC] (main patcher → [p OSC] → [p q-ctrlOSC]) by [s ---recqntz] for record quantization and, after being converted into a 0-2 range of values by [sel 0 7 4] and the attached messages, [s ---glblqntz] for clip trigger quantization.

   [p q-ctrlOSC] is responsible for ensuring the quantization settings displayed on the TouchOSC GUI match those of the Ableton Live Set both in real time and upon first loading the device/when the "Commit" button is pressed on the device GUI by the user at the end of the setup procedure. Utilising the "property clip_trigger_quantization" and "property midi_recording_quantization" observation functions in this scenario means that, when the system is being used in multi mode, if one user changes the value for either clip trigger quantization or record quantization all the other users will receive a visual indication of this on their individual GUI. This is a simple example of the implementation of "feedthrough" information in *ScreenPlay*.

3. It is the "get clip_trigger_quantization" and "get midi_recording_quantization" functions that ascertain the two quantization settings after being triggered both when the device is first loaded and when the user presses the "Commit" button following the completion of the setup process. The integer value outputted from [live.object] (which adheres to the same range of integers used to set the quantization values) is sent to [p q-ctrlOSC] by [s ---getglobq] and [s ---getrecq] respectively. Upon arrival in [p q-ctrlOSC] the clip trigger quantization value is also converted into the same range of integers used at (1) when setting/observing the quantization value before being sent to [p q-ctrlOSC] by [s ---glblqntz]. The 0-2 value is sent out of [p q-ctrlOSC] to various different sub-patchers via [s ---gqcommit] in order to ensure data flow through the patch is directed appropriately in accordance with the state of the Ableton Live Set after the setup process is complete and the "Commit" button has been pressed by the user.



RECORD QUANTIZE
live.thisdevice
path live_set
live.path
t b l
property midi_recording_quantization
live.observer
route 0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8
s ---recqntz
set midi_recording_quantization $1
route /recq/4/1 /recq/3/1 /recq/2/1 /recq/1/1
sel 1
r ---commit
loadbang
get midi_recording_quantization
live.object
unpack s 0
s ---getrecq
GLOBAL QUANTIZE
property clip_trigger_quantization
route 0 1 2 3 4 5 6 7 8 9 10 11 12 13
0 1 2 3 4 5 6 7 8 9 10 11 12 13
sel 0 7 4
s ---glblqntz
set clip_trigger_quantization $1
route /globq/1/1 /globq/2/1 /globq/3/1
get clip_trigger_quantization
s ---getglobq
1.
2.
3.

Figure 31. [p quantizesettings] (Funk, B. & Stearns, L., 2012).

9.2.14 [p metro-ctrl]

Main patcher → [p metro-ctrl]

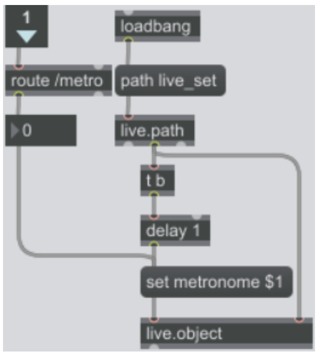


Figure 32. [p metro-ctrl].

[p metro-ctrl] is a simple patcher that routes the integer value outputted by the "metro" button on the TouchOSC GUI (1 = on, 0 = off) into the "set metronome $1" function, thus activating/deactivating the metronome. [p metro-ctrlOSC] (main patcher → [p OSC] → [p metro-ctrlOSC]) contains the "property metronome" observation function, which, after being triggered when the "Commit" button is pressed, continually monitors the state of the metronome in Ableton Live and updates the "metro" toggle on all TouchOSC GUIs currently connected to the Live Set so that all users are provided with visual feedback when the metronome is activated/deactivated.

Likewise [p timebarclrOSC] (main patcher → [p OSC] → [p timebarclrOSC]), [p tempo-ctrlOSC] (main patcher → [p OSC] → [p tempo-ctrlOSC]) and the [p hasclip*n*] patchers at (1) inside [p clipcheck] - in tandem with both [p cliploadedOSC] and [p clipclearOSC] (main patcher → [p OSC] → [p cliploadedOSC]/[p clipclearOSC]) - all contain continually ongoing observation functions that relay visual feedback relating to the playing/recording status of the clips in the Ableton Live Set (timebar colour appears blue if all clips are inactive, green if a clip is playing, and red if recording), tempo, and the presence of clips in clip slots to the connected TouchOSC GUI(s); with timebar colour and clips in the track being unique to each instrument/part and tempo being global.

9.2.15 [p tempo-ctrl]

Main patcher → [p tempo-ctrl]

[p tempo-ctrl] is almost identical to [p metro-ctrl] and is used to change the tempo of the Ableton Live Set in response to the tempo slider on the TouchOSC GUI.

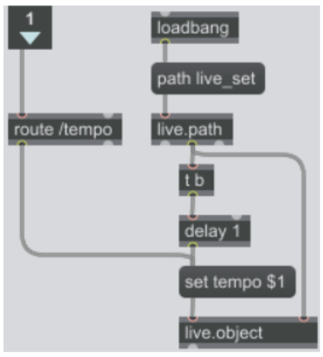


Figure 33. [p tempo-ctrl].

9.3 ScreenPlay-GEN

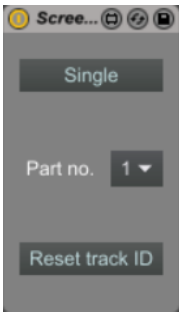


Figure 34. *ScreenPlay-GEN* Max for Live MIDI Device.

9.3.1 Main patcher/ScreenPlay-GEN 1.0

The main patcher window in the *ScreenPlay-GEN* Max for Live MIDI device shares many similar features with that of *ScreenPlay-CTRL*, such as the device setup controls for its GUI and the [plugsync~] object. Incoming OSC information is received from *ScreenPlay-CTRL* inside [p OSC], and outgoing information is sent back via [p mkvOSCsend]. Each of these sub-patchers contains sixteen [r OSC*n*]/[s OSC*n*] objects and a [gswitch]/[gswitch2] responsible for directing the incoming/outgoing information from/to the correct [receive]/[send] object in relation to the part number of the device set by the user.

ScreenPlay's generative algorithm makes use of both first and second order Markov chains. The next note to be generated in a first order Markov chain is determined by the previous note, while in a second order Markov chain the two preceding notes are taken into account when calculating the next note. The source material for the generative algorithm can either be taken from the currently active clip in the Ableton Live Set or played/recorded into the system directly by the user/performer in real time.

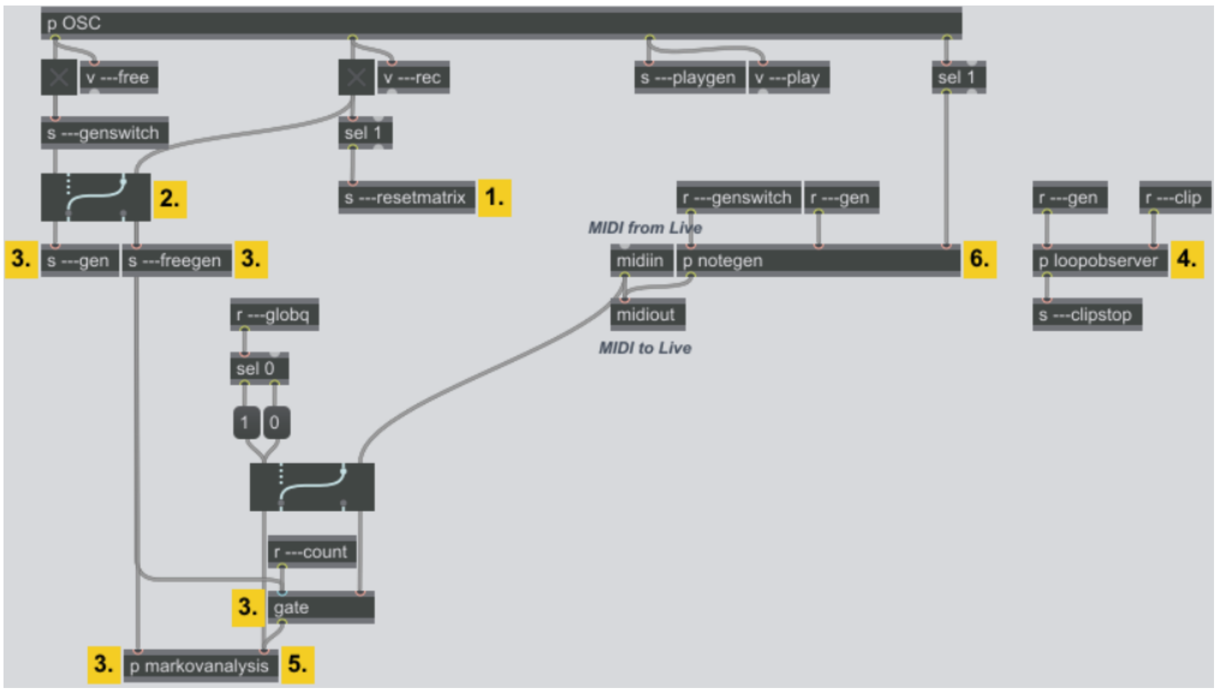


Figure 35. *ScreenPlay-GEN* main patcher window.

1. When the generative process is first triggered by the user via the "gen rec" toggle button on the TouchOSC GUI a "1" is released from the second outlet of [p OSC], which, after activating the [toggle] object, enters a [sel 1] object and causes a bang to be released from its left outlet and sent via [s ---resetmatrix]. This bang is sent to various locations in the patch in order to reset any values that may still be stored from previous generations. The value (whether 1 or 0) outputted from the second outlet of [p OSC] is also stored inside [v ---rec], from where it is recalled at the bottom of the main patcher window and sent via [p mkvOSCsend] back to *ScreenPlay-CTRL* upon the instrument/part with which the device is paired when in single mode being selected by the user so as to update the visual feedback presented on the GUI. The same is true of the 1/0 values arriving via the first and third outlets of [p OSC] relating to the state of the "freegen" and "playgen" toggles on the GUI, which are stored inside [v ---free] and [v---play] respectively. [v ---rec] and [v ---play] are also updated internally in case their status changes whilst another instrument/part is selected by the user and, as such, it is necessary to provide different visual feedback via the GUI upon re-selection of the part than that which was displayed when last it was selected.
2. Next, the "1" enters the right inlet of a [gswitch2], from where it is outputted through one of two outlets as dictated by a value of either 1 or 0 arriving at the left inlet from the [toggle] connected to the leftmost outlet of [p OSC], which communicates the state

of the "freegen" toggle on the GUI. When "freegen" is activated the user is able to record MIDI input live in real time to use as the source material for the Markovian generation process; when deactivated, the source material is collected from the currently active clip in the Ableton Live set at the time "gen rec" is enabled.

3. If "freegen" is enabled by the user the "1" arriving at the right inlet of [gswitch2] is released from its right outlet and, as well as being sent to multiple locations via [s ---freegen], enters the left inlet of both a [gate] object - thus opening it - and [p markovanalysis], from where it enters the left inlet of another [gate] object on the inside of the patcher. If freegen is disabled the "1" arriving at the right inlet of [gswitch2] is sent to a number of locations in the patch via [s ---gen].
4. First, the "1" sent via [s ---gen] arrives at [r ---gen] connected to the left inlet of [p loopobserver], while the clip index also arrives at the right inlet via [r ---clip]. ***Refer to 9.3.2 [p clipindex], then 9.3.3 [p loopobserver].***
5. Happening simultaneously with the actions inside [p loopobserver] are those inside [p markovanalysis], where the notes from the clip or those being freely played in/recorded by the user are analysed and stored. With regard to the individual analysis procedures for pitch, velocity, duration and delta time (time between note-on times) occurring inside [p pitchanalysis2nd], [p velanalysis1st], [p duranalysis1st] (main patcher → [p markovanalysis] → [p duranalysis1st]) and [p deltaanalysis1st] (main patcher → [p markovanalysis] → [p deltaanalysis1st]) respectively, only [p pitchanalysis2nd] and [p velanalysis1st] are addressed in the documentation due to the fact that pitch generation utilises a second order Markov chain - and so the method of analysing and storing pitch information differs slightly from that of velocity, duration and delta time - and velocity, duration and delta generation are all first order Markovian processes - meaning the collection methods for each are identical. ***Refer to 9.3.4 [p markovanalysis], 9.3.5 [p pitchanalysis2nd], then 9.3.6 [p velanalysis1st].***
6. Once the process of collecting, analysing and storing all the incoming MIDI note information (either from the active clip or being played in by the user) has been completed inside [p markovanalysis], the process of generating new musical material from that data can commence. If "freegen" is disabled this action is triggered inside [p notegen] when the bang released from [p loopobserver] is sent via [s ---clipstop]. Otherwise, if "freegen" is enabled, generative playback must be triggered manually by

the user via the "gen play" toggle on the TouchOSC GUI after deactivating the "gen rec" toggle. ***Refer to 9.3.9 [p notegen].***

9.3.2 [p clipindex]

Main patcher → [p clipindex]

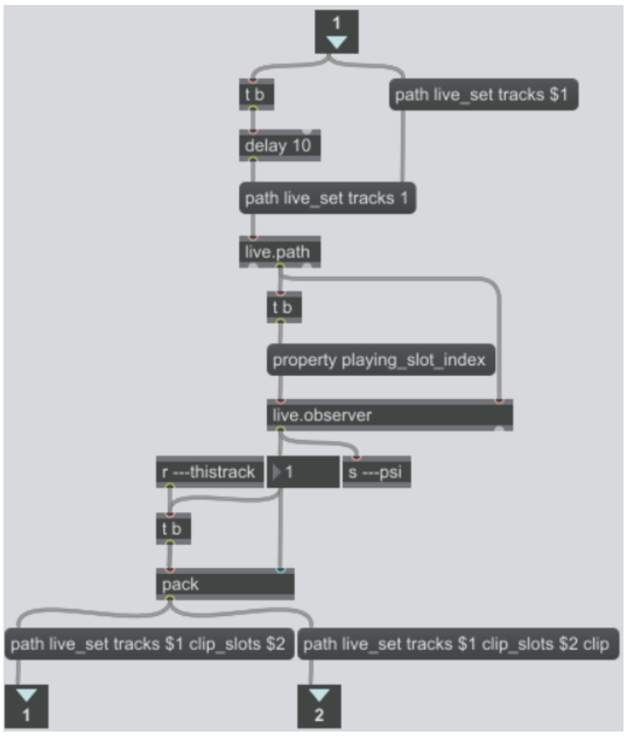


Figure 36. [p clipindex].

This patcher contains a function, which is triggered upon the arrival of the track index from [r ---thistrack] either when first loading the Max for Live device into the Ableton Live Set or when pressing the "Reset track ID" button on the device GUI, that observes the playing slot index of the target track for *ScreenPlay-GEN* in real time. The clip number is sent directly out of [p clipindex] through [s ---psi], while it is also packaged into two messages to be used for triggering other functions associated with either the currently active clip or clip slot. The clip message is released from the right outlet of the patcher and sent around the patch via [s ---clip], while the clip slot message is released from the left outlet and sent via [s ---clipslot].

9.3.3 [p loopobserver]

Main patcher → [p loopobserver]

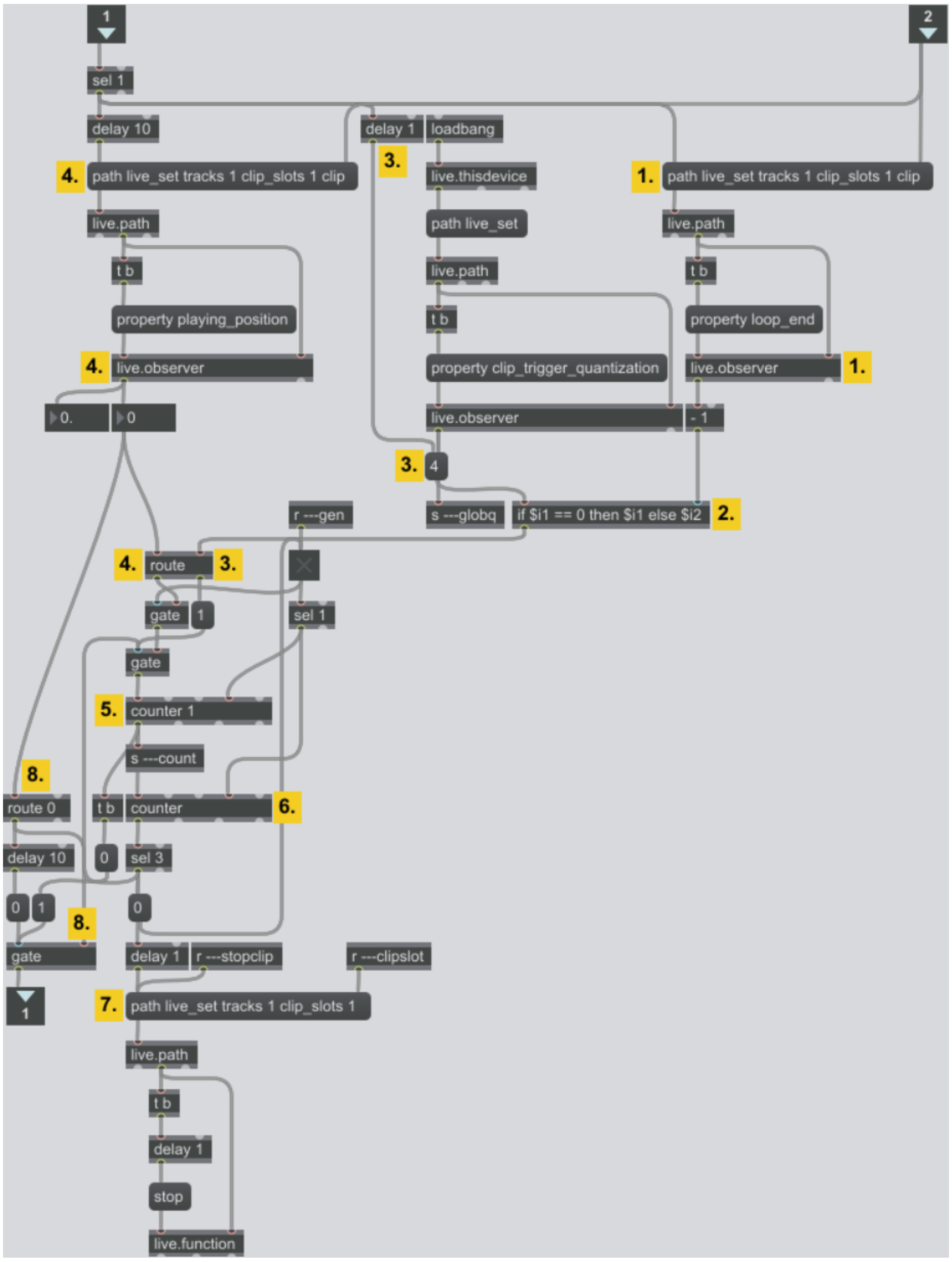


Figure 37. [p loopobserver].

1. Once "gen rec" has been activated by the user via the TouchOSC GUI a "1" enters the left [inlet] of [p loopobserver] and triggers a bang from [sel 1], which, in turn, begins by triggering the message containing the Live path details of the currently active clip.

This message box has been populated by the arrival of the message at its right inlet from the right [inlet] of [p loopobserver]; to which is attached [r ---clip]. Upon being released the message contained in the message box enters [live.path] and triggers a function that observes the position either of the end of the loop-section of the currently active clip or, if the clip is unlooped, the end of the clip.

2. The loop-/clip-end value is outputted from the left outlet of [live.observer] in beats and, after passing through [- 1] and having one beat subtracted from its actual value, enters the right inlet of an [if] object where it replaces $i2 in the arguments.
3. Next, the bang outputted by [sel 1] connected to [inlet] 1 of the patcher is delayed by 1 ms when passing through [delay 1] before triggering an integer value message associated with the current clip trigger quantization value from the Ableton Live Set, which is sent into the left inlet of the [if] object. If the value is equal to 0 (indicating that clip trigger quantization is disabled) then it passes back out through the outlet of [if]. If the value is not equal to 0 (meaning clip trigger quantization is activated) then the value arriving at the right inlet of [if] is outputted in place of the value arriving at the left inlet. This number is then directed into the right inlet of the connected [route] object, thus setting its arguments.

   The value stored in the message box denoting the current clip trigger quantization value from the Ableton Live Set is updated by the "property clip_trigger_quantization" observation function above, which is activated by [loadbang] when the Max for Live device is first loaded, whenever it is changed by the user. The clip trigger quantization value is also sent to various other locations in the patch via [s ---globq].
4. The bang from [sel 1] is then delayed by 10 ms before triggering the "property playing_position" function, which observes the playing position in beats of the current clip. The decimal floating point number outputted from [live.observer] is converted into an integer (decimal places are removed but the number is not rounded - i.e. 1.9 becomes 1, not 2) before entering the left inlet of [route] and, once it is equal to the value of the number that arrived at the right inlet of [route] from [if] (either 0 or (length of the loop/clip - 1) beats), causes a bang to be released from the [route]'s left outlet. This bang arrives at the right inlet of a [gate], which is opened upon receiving a "1" from the [toggle] connected to [r ---gen] when the generative algorithm is first activated by the user via "gen rec" on the TouchOSC GUI, and passes through into a

second [gate]. This [gate] is opened by the "1" message connected to the right outlet of [route], through which the value arriving at the left inlet is outputted if it does not match the value set at the right inlet. This ensures that, when the bang from [route] arrives after passing through the first [gate] object, the second [gate] is open in order to let it pass through.

5. The bang then enters a [counter 1] object, which is reset to 0 by the bang arriving at its fourth inlet from [sel 1] when the user first activates the generation process. The "1" argument denotes the maximum threshold of the [counter], and so the output from its leftmost outlet simply cycles between 0 and 1 as it reaches its maximum threshold and resets to 0. The value outputted is sent elsewhere in the patch via [s --count] as well as into a second [counter] object. It also triggers a bang from [t b], which then triggers a "0" back into the left inlet of the [gate] connected to its left inlet, thus closing it.

   This is necessary because, even though the floating point value outputted from [live.observer] is converted into an integer, [route] still receives multiple individual messages in a row containing that integer due to the fact that, before the conversion from floating point number to integer takes places, [live.observer] outputs a vast amount of six-decimal-place floating point values between each round integer. Every time one of these values passes through the [number] box connected to [live.observer], which serves to convert the floating point values to individual integer values, [route] compares the incoming integer with the integer received at its right inlet and, therefore, outputs a large stream of bangs. Closing the [gate] as soon as the [counter] increases ensures only one of those bangs can pass through. There is also a reason as to why the [route] object cannot, in this scenario, respond to a specific floating point value - for example 0.000000; the reason being that the floating point values outputted by [live.observer] increase far too quickly for the [route] object to accurately detect/register the specific floating point value that would be stored in its arguments.

   To use a [change] object immediately after the conversion of the output of [live.observer] from floating point value to integer, which allows a value to pass through only if it is different from the previous value, seems, on the surface, a more straightforward solution. However, in the rare circumstances that playback of a clip is triggered while another clip is still playing and the first value to be outputted from [live.observer] in relation to the new clip is the same as the last from the previous clip, and "gen rec" is immediately activated by the user before the value outputted value from [live.observer] has a chance to change from the intial value outputted in relation to the new clip, the generative process will not work as intended and the user will have to listen to two complete playthroughs of the clip/loop prior to the playback of generated musical material beginning instead of only one. Where experimental music is concerned and any of slower tempos, shorter loops/clips, odd time signatures, disabling clip trigger quantization etc. are more likely the chance of this issue with the [change] object arising is increased. Although fairly lengthy, the workaround used here is efficient and reliable and negates this ever being a problem

6. The next [counter], into which the output of [counter 1] is sent - and which is also reset to 0 upon activation of the generative algorithm - causes a bang to be released from [sel 3] once it receives three messages, regardless of value, from [counter 1]. The bang released from [sel 3], in turn, triggers a "0", which is sent into the [toggle] connected to [r ---gen] and causes it to close the first [gate] attached to [route] at (4), thus allowing no more bangs to pass through from [route] and into the second [gate] before entering [counter 1]; therefore halting the process of counting how many times the active clip has looped since the generative algorithm was activated.
7. The bang released by [sel 3] is delayed by 1 ms before triggering the "stop" function, which causes the clip slot containing the active clip to be fired and, therefore, the clip to stop playing at the end of the loop.
8. The bang from [sel 3] also sends a "1" into the left inlet of a [gate], thus opening it, through which a bang is sent from the connected [route 0] when the clip/loop finishes a complete playback cycle and the playback position returns to the beginning (zero beats). After passing through the [gate] this bang is sent out of the [patcher] and into [s ---clipstop], from where it is sent to [p notegen] in order to trigger the generative playback of the algorithm once all the notes within the clip have been collected/analysed. The bang from [route 0] is also delayed by 10 ms before sending a "0" to the [gate] causing it to close again.

The purpose of this procedure is to allow time for the active clip to play through once in its entirety from beginning to end in order for the Markovian analysis of the notes in the clip to take place, before triggering the clip to stop playing at the same time as activating the generation of new material from the notes which have been collected. ***Refer to 9.3.1 Main patcher/ScreenPlay-GEN 1.0, (5).***

9.3.4 [p markovanalysis]

Main patcher → [p markovanalysis]

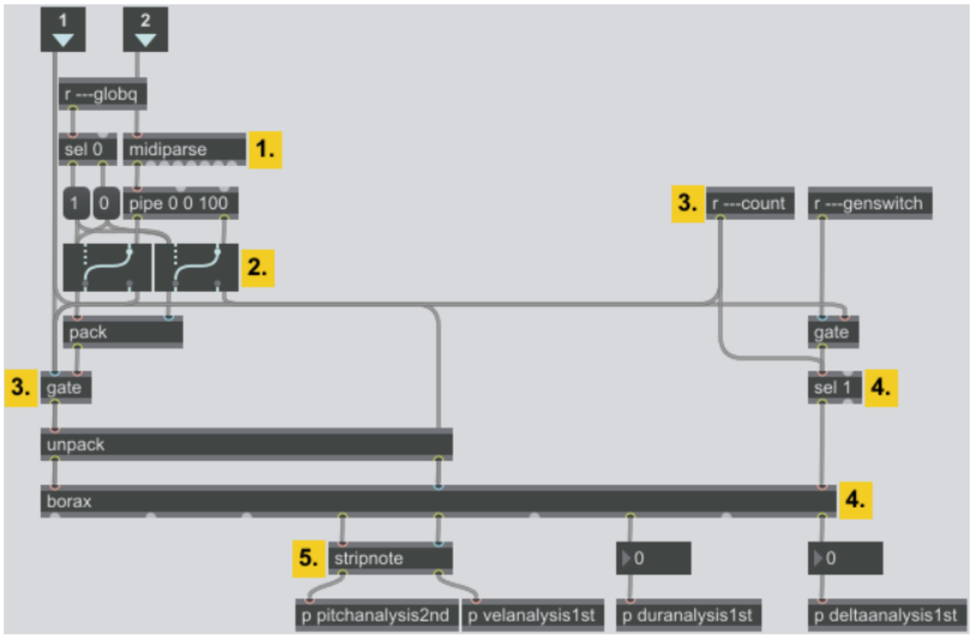


Figure 38. [p markovanalysis].

1. The incoming MIDI information, either from the active clip or being played in by the user, arrives at [inlet] 2 of the sub-patcher and the pitch and velocity information is separated from the rest of the MIDI message when passing through [midiparse], before being delayed by 100 ms and separated from each other into two individual messages when passing through [pipe 0 0 100].
2. Each of the two values then enters the right inlet of its own [gswitch2] object and are outputted from the right outlet if clip trigger quantization is disabled or from the left outlet if clip trigger quantization is enabled - regardless of the specific quantization value - as dictated by [r ---globq]-[sel 0]-"0"/"1". When coming out of the right outlet, with clip trigger quantization disabled, the two values are sent directly into the pitch and velocity inlets of [borax]. When coming out through the left outlet, with clip trigger quantization enabled, the two values pass through [pack] in order to reform the list originally outputted from [midiparse] before entering the right inlet of a [gate].
3. If "freegen" is enabled by the user a "1" arrives via [inlet] 1 of the patcher when "gen rec" is activated by the user on the GUI from the right outlet of the [gswtich2] at (2) in the main patcher window and is sent into the left inlet of the [gate], thus enabling the "pitch; velocity" list to pass through immediately before being unpacked and and directed into the pitch and velocity inlets on the [borax] object. If "freegen" is disabled

the [gate] is instead opened upon receiving a "1" from [r ---count], which is sent from (5) inside [p loopobserver] and cycles between 1 and 0; with 1 indicating the start of the active clip/loop and 0, which acts to close the [gate], indicating the end. As a result the pitch and velocity information for all the notes inside the clip is allowed to pass through the [gate] in order to be analysed and collected during a single playthrough of the clip/loop in its entirety, after which point the clip is stopped and the playback of newly generated musical material from the stored transition tables can commence.

4. Also of importance when accurately collecting the MIDI note information stored inside the active clip or being played/recorded in by the user is the activity occurring at the rightmost inlet of [borax], which, upon receiving a bang, turns off all sounding notes which may still be passing through it. This happens when [sel 1] receives a "1" either from [r ---count] at the beginning of the clip/loop if "freegen" is disabled, or, if "freegen" is enabled, directly from the [toggle] connected to the second outlet of [p OSC] (main patcher → [p OSC]) after passing through the right outlet of [gswitch2] at (2) in the main patcher via [inlet] 1 of the patcher, and after passing through the [gate] preceding [sel 1] (which is opened by a 1 arriving at its left inlet from [r ---genswitch] when "freegen" is enabled and closed by a 0 when "freegen" is disabled) in order to stop any overhanging notes being collected at the start of the analysis procedure prior to the first note(s) present in the clip or played by the user.
5. Before the pitch and velocity values can enter their respective analysis patchers it is necessary for them to pass through [stripnote], which removes note-off messages. Without passing through [stripnote] prior to analysis and collection, the pitch of each note in the clip would be duplicated - one for note-on and one for note off - and, along with the actual velocity of the notes - which would be paired with the note-on pitch - there would also be a velocity of 0 paired with the note-off pitch. No such problem exists with duration and delta time values and, as such, these values can enter directly into their respective analysis patchers.

9.3.5 [p pitchanalysis2nd]

Main patcher → [p markovanalysis] → [p pitchanalysis2nd]

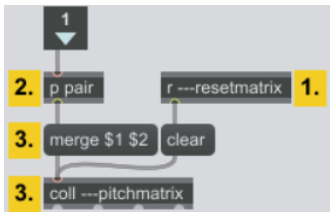


Figure 39. [p pitchanalysis2nd] (Acuma, 2010b).

1. When the generative algorithm is first activated by the user [r ---resetmatrix] receives a bang from (1) in the main patcher, which, in turn, triggers a "clear" message that enters [coll ---pitchmatrix] (the object used to store the analysed pitch data for the purposes of generating new musical material) and deletes any previously stored data in preparation for the arrival of new pitch data.
2. Next, the MIDI note information arriving at the [inlet] of the patcher enters [p pair]. ***Refer to 9.3.7 [p pair] (pitchanalysis2nd).***
3. Once the "preceding-note-pair; note" list is released from [p pair], it passes through the "merge $1 $2" message - with the two list elements populating the variable arguments in the message - before entering into the [coll] object to be stored and recalled during the process of generating new musical material from the collected note information. It is necessary for the pitches of the two preceding notes to be paired together into a single integer value so as to be able to act as an index inside the [coll], alongside which all the pitches that are preceded by a particular pair of notes can be stored and recalled as individual elements; thus creating a second-order Markovian transition table of values. The "merge $1 $2" message serves the purpose of ensuring that newly arriving pitch elements which share the same note-pair index as other elements already existing inside the [coll] are added to the list of elements rather than replacing those that are already present.

9.3.6 [p velanalysis1st]

Main patcher → [p markovanalysis] → [p velanalysis1st]

Apart from the simplified programming inside [p pair], given that the index values in the [coll] in this instance consist only of a single velocity value due to the implementation of a first-order Markov chain as opposed to second-order, the programming inside [p velanalysis1st] is exactly the same as that of [p pitchanalysis2nd]. ***Refer to 9.3.8 [p pair] (velanalysis1st).***

9.3.7 [p pair] (pitchanalysis2nd)

Main patcher → [p markovanalysis] → [p pitchanalysis2nd] → [p pair]

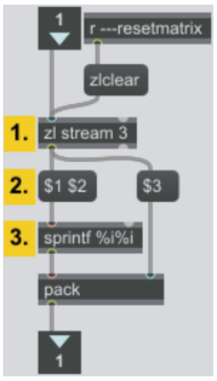


Figure 40. [p pair] inside [p pitchanalysis2nd] (Acuma, 2010b).

1. Once inside [p pair] the MIDI notes arriving from [p pitchanalysis2nd] first enter [zl stream 3], which, like the [coll] object inside [p pitchanalysis2nd], has also been cleared of any previously stored data upon activation of the generative algorithm; this time by the bang arriving at [r ---resetmatirx] triggering the "zlclear" message. [zl stream] creates a list from the last specified number of items to arrive at its inlet; in this case the number "3" can be seen to be defined in the arguments of the object. In practice this means that, when the object is completely empty, the first list is outputted once it has received three separate messages/items, after which point a new list is outputted consisting of the three most recent items every time a new item arrives.
2. This list is then split into a new list containing the first two elements and an individual message consisting of the final element when passing through the "$1 $2" and "$3" messages. As the Markov chain for pitch is second order, the result of this is to provide each pitch with a corresponding list of the two pitches that preceded it.
3. The list of two pitches is then paired (not added) together to create a single integer value (i.e. 60 64 becomes 6064) when passing through [sprintf %i%i], before a new two-element list consisting of the pitch-pair-integer and the final pitch of the original three-element list is created by [pack], following which it is released from the patcher and back into [p pitchanalysis2nd]. ***Refer to 9.3.5 [p pitchanalysis2nd], (3).***

9.3.8 [p pair] (velanalysis1st)

Main patcher → [p markovanalysis] → [p velanalysis1st] → [p pair]

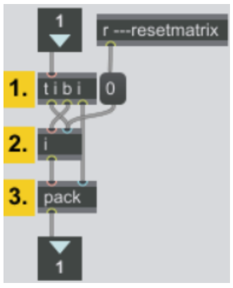


Figure 41. [p pair] inside [p velanalysis1st] (Acuma, 2010b).

1. As the velocity values arrive inside [p pair] from [p velanalysis1st] they first enter a [trigger] ([t]) object, which releases the incoming integer values from its right outlet into the right inlet of a [pack] object where it awaits the arrival of the preceding velocity value.
2. A bang is then sent from the middle outlet of [t i b i] into the left inlet of an [int] ([i]), which stores a single integer value inputted at its right inlet and releases it upon receiving a bang at its left inlet. [t i b i] then outputs the incoming integer from its leftmost outlet and into the right inlet of [int]. The ordering of these events is crucial and relies upon Max/MSP's right-to-left, bottom-to-top message ordering protocol. Because the bang released from the middle outlet of [t i b i] and sent into the left inlet of [int] occurs before the incoming velocity value is released from the leftmost outlet of [t i b i] and stored in [int] upon arriving at its right inlet, the arrival of the bang at [int] outputs the previously stored velocity value to the one that has just arrived at [t i b i], been released into the right inlet of [pack] and is about to be released into the right inlet of [int].

   Also of note is the fact that, each time the generative algorithm is activated by the user, the bang received via [r ---resetmatrix] resets the value stored in the integer to 0 so as to ensure no information is carried over from previously used source material.
3. The preceding velocity value released from [int] then enters the left inlet of [pack] to form a two-element list with the current velocity value before being released from the patcher to be stored inside the [coll] in [p velanalysis1st]. ***Refer to 9.3.1 Main patcher/ScreenPlay-GEN 1.0, (6).***

9.3.9 [p notegen]

Main patcher → [p notegen]

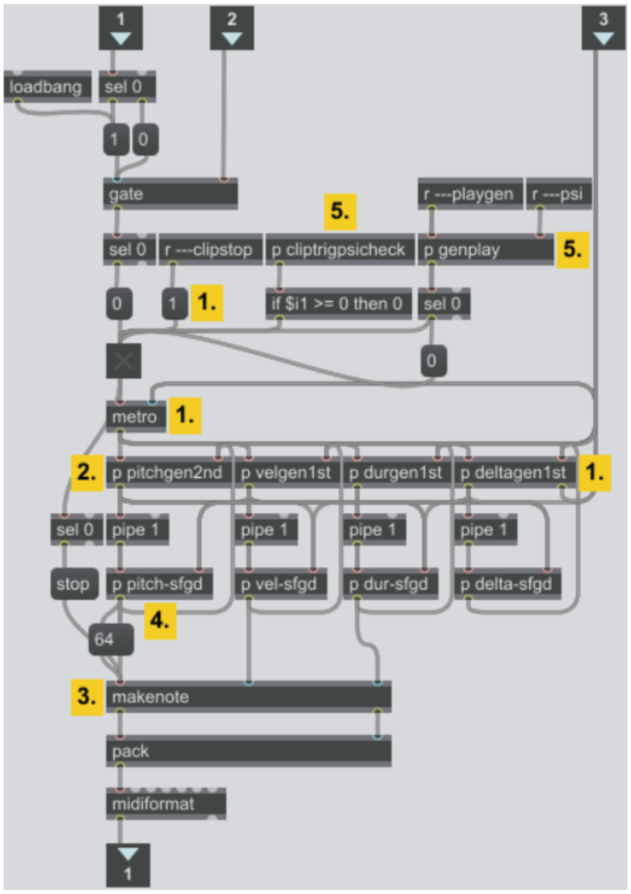


Figure 42. [p notegen].

1. The bang released from [p loopobserver] is received via [r ---clipstop] and triggers a "1", which, in turn, activates a [toggle] connected to a [metro] object, the tempo of which is set by the value outputted from the right outlet of [p deltagen1st] - the patcher responsible for generating the interval times between notes.
2. The bang outputted from the [metro] is used to systematically trigger the generation of a new set of pitch, velocity, duration and delta time values to be outputted by the Max for Live device every time it is released. For the same reasons as with the collection and analysis of the incoming MIDI note information, the focus here will be on the second-order generation of pitch values inside [p pitchgen2nd] and the first-order generation of velocity values inside [p velgen1st]. However, [p deltagen1st] will also be touched upon briefly due to some additional functionality not shared by the other two first-order generative patchers. ***Refer to 9.3.10 [p pitchgen2nd], 9.3.11 [p velgen1st], then 9.3.12 [p deltagen1st].***

3. Once the delta time, duration, velocity and pitch values for the current note have all been generated, the delta time is sent back up into the right inlet of [metro] at (1) to ensure that the release of the next bang to drive the generation of the next note happens at the right time. The duration, velocity and pitch values all enter the [makenote]-[pack]-[midiformat] sequence of objects where they are compiled into a usable MIDI note message before being outputted from the [patcher] and then, after passing through the main patcher window, the Max for Live device itself back into the Ableton Live Set where the note triggers whatever instrument shares the track with the *ScreenPlay-GEN* Max for Live MIDI Device.
4. After the individual note values have been sent to their primary destinations they are all delayed by 1 ms when passing through [pipe 1] before entering the left inlet of their own safeguard patcher; the purpose of which is to ensure that, if there are no values of any of the generated parameters that are preceded by the last value to be generated and the generation of that particular parameter is halted as a result, a bang can be sent up to the "next" message connected to the first [coll] at (1) inside the generative patchers and kick-start the generation of that particular parameter again. This works by routing the output of the pitch, velocity and delta time generative patchers into the right inlet of each of the safeguard patchers (excluding their own) and assuming that, if one or more generative patchers comes to a halt, at least one will remain active. If all the generative patchers do become stuck at the same time and as a result the system fails to automatically restart the generation process there is also a "next" button on the TouchOSC GUI that the user can press to manually reactivate the generation process.

   The reason the output of [p durgen1st] is not sent into the right inlet of the other safeguard patchers is due to the fact that, in particular when working with quantized material, it is the most likely generative patcher to come to a halt due to a small range of available durations and so cannot be relied upon to restart the other generative patchers. It should also be noted that the pitch value is the most important of the four as a result of the leftmost inlet on the [makenote] object being the "hot" inlet that causes the release of the compiled note message, and so no new pitch value means no new note. This is combatted by directing the pitch value released by [p pitchgen2] into the right inlet of a message box, the output of which is forced by the bang released from [p pitch-sfgd] and is sent into the left inlet of [makenote]. ***Refer to 9.3.13 [p pitch-sfgd].***

5. Also of great importance inside the [p notegen] patcher are the [p genplay] and [p cliptrigpsicheck] patchers, which serve to control playback of the generative algorithm in response to the triggering of clips inside the Ableton Live Set and vice versa. If [p genplay] receives a "0" - indicating that the playback has been turned off directly by the user via the "gen play" switch on the TouchOSC GUI - a bang is automatically released from [sel 0] that sends a "0" into the [toggle] at (1), which disables any further generation of new notes. However, the output of [r ---playgen] also enters the left inlet of [p genplay], along with the integer value indicating the playing slot index of the track in the Ableton Live Set at the right inlet via [r ---psi], in order to appropriately time the beginning of generative playback and stopping of the active clip in accordance with the clip trigger quantization value if the user activates "gen play" whilst a clip is still active. This helps to achieve a seamless transition between the two playback sources. ***Refer to 9.3.14 [p genplay], then 9.3.15 [p cliptrigpsicheck].***

9.3.10 [p pitchgen2nd]

Main patcher → [p notegen] → [p pitchgen2nd]

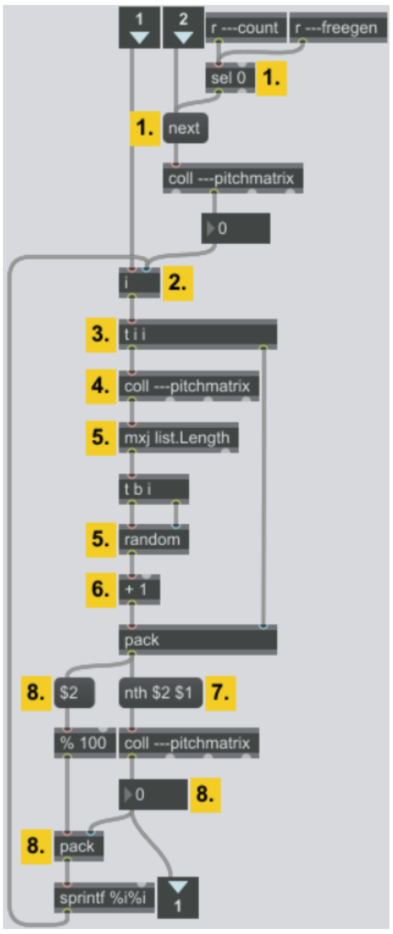


Figure 43. [p pitchgen2nd] (Acuma, 2010b).

1. The process of generating the first pitch value to be outputted by the system begins when [r ---count] receives a 0 from its counterpart in [p loopobserver] to signify that the collection of all note information in the clip has been completed or, if "freegen" is enabled, when [r ---freegen] receives a 0 from the main patcher indicating that the "gen rec" toggle button on the TouchOSC GUI has been deactivated by the user. However, with "freegen" enabled playback of the generated notes still needs to be activated manually by enabling the "play gen" toggle button on the TouchOSC GUI after disabling the "gen rec" toggle in order to turn on the [toggle] connected to the [metro] that drives the generation of notes at (1) in [p notegen].
2. Upon receiving the "next" message the [coll] releases the pitches of the first two notes contained in the clip in the form of a pitch-pair integer index value from its second outlet, which is then directed into the right inlet of an [int] object where it awaits the arrival of the first bang to be released from the [metro] that drives the generation process at (1) inside [p notegen] via [inlet] 1 of the patcher.
3. Once the pitch-pair index has been released from [int] it enters [t i i], from where it is first released out of the right outlet and into the right inlet of a [pack] object to await the arrival of the other value with which it will be paired in order to recall the next pitch value to be generated from the [coll].
4. Next, the pitch-pair value is released from the left outlet of [t i i] and enters a second copy of [coll ---pitchmatrix], from where the entire list of single pitch value elements associated with that index are released from the leftmost outlet (all the pitches that are preceded by the two pitches contained in the pitch-pair).
5. The outputted list of values then enters [mxj list.Length], which outputs the number of elements contained in the list (list length) and sends that value into [t b i]. From here, the list length value is first outputted from the right outlet and into the right inlet of [random] (an object which generates a random value within a certain range as denoted by its arguments), thus defining the range of values from which to randomly generate an integer. A bang is then released from the left outlet of [t b i] and triggers the generation of a random integer from [random] within the range defined by the list length value inputted at its right inlet.
6. The value outputted by [random] passes through [+ 1], where a value of 1 is added to it, before entering the left inlet of [pack] to form a two-item list with the pitch-pair value outputted from the first instance of [coll ---pitchmatrix] at (1)-(3).

7. The arrival of the randomly generated value at [pack] causes the newly formed list to be outputted, from where it passes through the "nth $2 $1" message and into a third instance of [coll ---pitchmatrix]. "nth" messages are used to recall a specific data element from a specific index within [coll], with the first value following "nth" denoting the index and the second value denoting the position in the list of elements associated with that index from where to recall the element. Hence, "nth $2 $1" serves to swap the order of the two elements in the list arriving at the inlet of the message box before outputting the message and sending it into [coll] by ordering the variables as "$2 $1" - the number of each variable indicating the position of the value in the incoming list by which they should be replaced when outputting the new message. This is because the incoming list is ordered as "(randomly generated value + 1); pitch-pair", whereas the data contents of the [coll] are organised with pitch-pair values as the index for each list of individual pitch values that follow each particular pitch-pairing in the source material. The reason for adding 1 to the randomly generated value at (6) is that the value 0 is not recognised by [coll] as a valid indicator for an element within an index, and [random] generates numbers within its specified range starting from 0 (i.e. [random 10] generates values ranging from 0-9). The position of data elements in each index are numbered in ascending order from left to right, beginning at 1.

   The process of randomly generating a value in order to choose the pitch value to be outputted by the system next works well because *all* individual pitches preceded by a certain pitch-pairing in the source material are collected and stored, including repeated notes. Therefore, the heightened probability of a pitch that more commonly follows two particular pitches than do the others in that index is accurately represented given that, even though there is an equal chance for any of the values available in the range of the [random] object to be generated, there will be multiple instances of repeated notes stored in that pitch-pair index - each of which is recognised by its own numerical positional indicator.
8. The pitch value recalled from [coll] is first released from the patcher before entering the right inlet of a [pack] object in order to be paired with the second pitch in the pitch-pair that preceded it, thus creating a new pitch-pair to be used to generate the next pitch in line. The pitch-pair used to generate the pitch of the current note is separated from the random value with which it was paired at (6) when passing through

the "$2" message before entering [% 100], which divides the incoming pitch-pair integer by 100 and outputs the difference, which, in this scenario, is the pitch value of the second note in the pair (example: 6072 / 100 = 60.72; difference when represented as an integer = 72). The value released by [% 100] enters the left inlet of [pack] to create a two-element list with the pitch value that has just been generated, which, in turn, is paired into a single integer value when passing through [sprintf %i%i]; much in the same way as at (3) inside [p pair] (markovanalysis2nd). This value is then sent back up into the right inlet of [int] at (2) where it awaits the arrival of the bang from the [inlet] of the patcher connected to the [metro] that drives the generation of MIDI notes so that the process can begin again.

9.3.11 [p velgen1st]

Main patcher → [p notegen] → [p velgen1st]

[p velgen1st] is exactly the same as [p pitchgen2nd], except that, as it is utilising a first order Markov chain, it is not necessary to create a new pitch-pair to send back into [int] at (2). Instead, the pitch value released from [coll] at (8) is directed straight back into [int] without any further processing. It should also be noted that the [coll] located at (1) outputs an element rather than an index. This is because the value stored in the [int] object at (2) inside [p pair] (velanalysis1st) is reset to 0 at the beginning of each generation, meaning that the very first index value is a 0. This does not cause any disruption to the generation process due to the fact that there will almost never be an actual velocity or duration value of 0 in the source material and so the "0" index will never be recalled, while a delta time of 0 will only be collected from polyphonic source material and, seeing as in this instance there will always be more than one other delta time in the [coll] that is preceded by "0", recalling the 0 index will not stop the generation of new note information.

9.3.12 [p deltagen1st]

Main patcher → [p notegen] → [p velgen1st]

The difference between [p deltagen1st] and the other first-order generative patchers comes after the initial action caused by the arrival of a "0" at either [r ---count] or [r ---freegen].

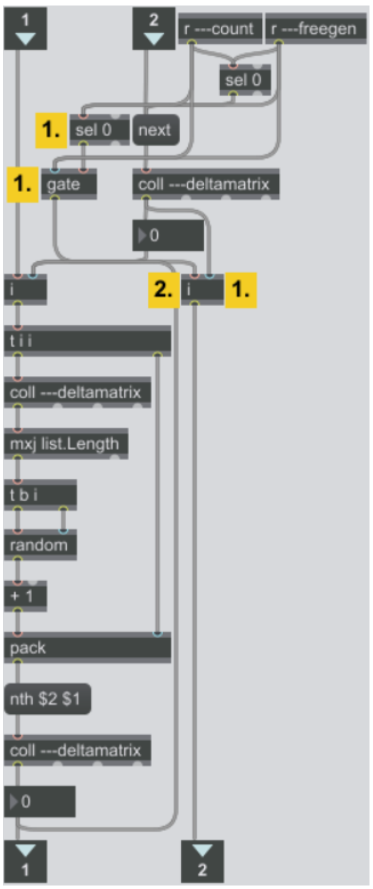


Figure 44. [p deltagen1st].

1. As well as being inputted into the right inlet of [int] at (2) in [p pitchgen2nd] after being recalled from [coll] by the "next" message, the first delta time value also enters the right inlet of another [int] object; while the value outputted by [r ---count]/[r ---freegen] also enters the left inlet of a [gate] object as well as a second [sel 0] object. Either when "gen rec" is activated by the user with "freegen" enabled, or when [r ---count] receives a "1" to signify that the collection of note information from the clip has begun, the [gate] is opened as a result of receiving a "1". Then, when "gen rec" is disabled by the user or the collection of note information from the clip is completed, [r ---count]/[r ---freegen] receive a "0", which first causes a bang to be released from [sel 0] and sent through the [gate] via its right inlet before arriving at the left inlet of [gate] and causing it to close again.
2. After the bang passes through the [gate] it enters the left inlet of [int] and releases the first delta time value from [outlet] 2 of the patcher, from where it is sent directly into the [metro] responsible for driving the note generation process.

This only happens once at the very beginning of the generation process and it ensures that the delta time value of the first note to be generated is not the the same as the last delta time value to be generated for the previous generation. If this section of patching was not included the first delta time value from the set of source material of the current generation would not be outputted and sent into the right inlet of the [metro] object to denote the time between the first and second notes until *after* the [metro] had already outputted its first bang causing the generation of the first note. ***Refer to 9.3.9 [p notegen], (3).***

9.3.13 [p pitch-sfgd]

Main patcher → [p notegen] → [p pitch-sfgd]

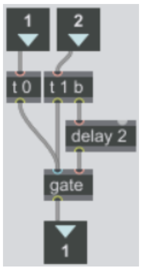


Figure 45. [p pitch-sfgd].

Because all the safeguard patches are identical the focus here will be on [p pitch-sfgd], the functionality of which is extremely simple. Due to the fact that the value outputted by the corresponding generative patcher (in this case [p pitchgen2nd]) is delayed by 1 ms by [pipe 1] before arriving at [inlet] 1 of the safeguard patcher, the first values to arrive are those of the other two generative patchers that are connected to [inlet] 2 (in this case [p velgen1st] and [p durgen1st]) - although only the arrival of a single value from one or the other is necessary to restart the corresponding generative patcher to any particular safeguard patcher. The arrival of this value at [inlet] 2 first triggers a bang from the right outlet of [t 1 b] that then enters a [delay 2] object on its way to the right inlet of a [gate]. This bang is followed by a "1" being released from the left outlet of [t 1 b] and sent into the left inlet of [gate]. Because the "1" is not delayed on its way to the [gate] it arrives before the bang and opens the [gate] in preparation for allowing the bang to pass through. However, because the value outputted by [p pitchgen2nd] is only delayed by 1 ms before arriving at [inlet] 1 of the patcher and the bang from [t 1 b] is delayed by 2 ms, the value from [p pitchgen2nd] tirggers a "0" from [t 0] upon its arrival at [inlet] 1. which, in turn, is sent into the left inlet of [gate] and closes it again before the bang is able to pass through. Only if [p pitchgen2nd] has failed to generate a new pitch value will the [gate] remain open and allow the bang to pass through, from where it will leave the patcher and return to the right inlet of [p pitchgen2nd] in order to trigger the "next" message at (1) inside and restart the generation process. ***Refer to 9.3.9 [p notegen], (5).***

9.3.14 [p genplay]

Main patcher → [p notegen] → [p genplay]

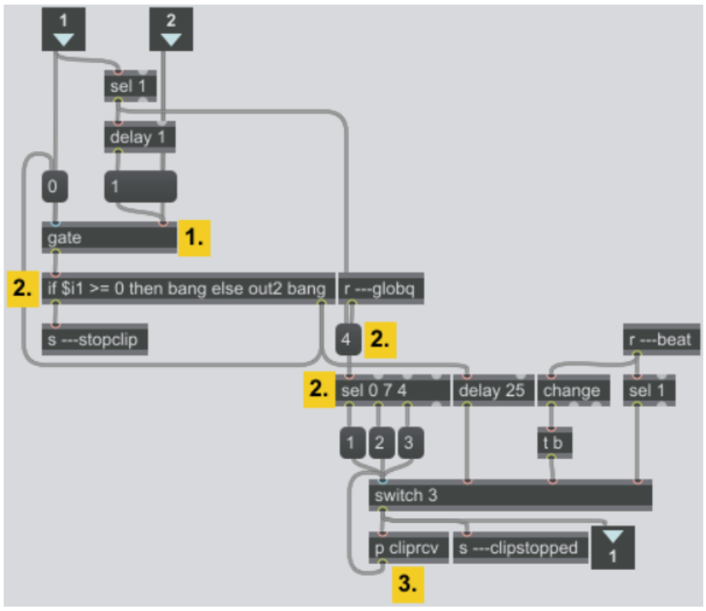


Figure 46. [p genplay].

1. Upon arriving at [inlet] 1, the value received by [r ---playgen] in [p notegen] (either a 1 or a 0) first arrives at a [sel 1], which, if the incoming value is a 1 ("gen play" activated by the user) releases a bang. After being delayed by 1 ms the bang reaches the left inlet of a message box - the contents of which are updated in realtime with the value denoting the playing slot index of the track in the Ableton Live Set that arrives at [inlet] 2 via [r ---psi] in [p notegen] from [p clipindex] - and sends the message into the right inlet of [gate]. Because the bang released from [sel 1] is delayed by 1 ms the [gate] will already have been opened when the 1 from [inlet] 1 arrived at its left inlet, thus allowing the clip index message to pass through.
2. The playing slot index value the enters an [if] object where, if it is found to be greater than or equal to 0 (this indicates that one of the clips in the track is currently active), a bang is sent out of the left outlet and into [s ---stopclip]. From here it is sent to (7) inside [p loopobserver] where it triggers a function that causes the active clip to stop playing at the next clip trigger quantization step.

The bang released from [sel 1] will also have triggered a message containing the integer value that denotes the current clip trigger quantization settings (either off, 1/4 note or 1 bar as per the available options on the TouchOSC GUI), received via [r ---globq]. This value enters a [select] object and, depending on the clip trigger quantization value, changes the inlet of the [switch] through which incoming data is allowed to pass. If the clip trigger quantization value is set to 1 bar a "3" is sent into the left inlet of [switch] and the rightmost inlet is opened. The beat of the global transport in the Ableton Live Set arrives via [r ---beat] from the [plugsync~] object in the main patcher and, as soon as a new bar begins and the beat value is equal to 1, the connected [sel 1] object outputs a bang that passes through the [switch] and out of the patcher to activate the [toggle] - and therefore the playback of the generative algorithm - at (1) in [p notegen]. If a "2" is received at the left inlet of [switch] - meaning the clip trigger quantization is set to a 1/4 beat - the third inlet is opened, through which a bang is sent every beat, and generative playback will begin at the next beat.

A "1" opens the second inlet of [switch] when clip trigger quantization is disabled. Because the bang sent via [s ---stopclip] causes the clip to stop playing immediately, the integer arriving at [inlet] 2, which denotes the playing slot index of the track in the Ableton Live Set, will immediately change to -2. As well as entering the right inlet of the message box connected to the right inlet of [gate] at (1), the value also passes directly into the [gate] itself. At this point the [gate] is still open, as it has not received any new values at its left inlet since the "1" from [inlet] 1 of the patcher when "gen play" was first triggered by the user at the beginning of the process, thus the value of -2 passes through and into the [if] object. Because the value is less than 0 a bang is released from the right outlet of [if] and, after being delayed by 25 ms, passes through the second inlet of [switch] and causes the generative playback to begin immediately. The bang released from the right outlet of [if] also triggers a "0" to be sent to the [gate] at (1) in order to close it again so as to block any new values arriving at its right inlet via [inlet] 2 of the patcher from passing through.

If when the user activates "gen play" on the GUI there is no clip currently active, the procedure is exactly the same albeit with the bang being released from the right outlet of [if] in the first place so that, if clip trigger quantization is disabled, generative playback starts immediately.

3. In either scenario, once the bang is released from [switch] to trigger the start of playback it also enters [p cliprcv] where it triggers a "0" that is sent back into [switch] in order to close it and stop any more bangs coming through. On the inside of the patcher [r ---OSC] receives the incoming OSC data stream from the TouchOSC GUI and, in the event that any of the clips are triggered by the user from there, the "0" is also released and sent into [switch] as a safeguard against simultaneous playback from both the active clip and the generative algorithm. In addition, the bang is sent via [s ---clipstopped] so that the visual feedback provided by the GUI can be updated accordingly when these actions occur.

Incorporating this system into the design of *ScreenPlay* not only ensures switching between playback from the generative algorithm and MIDI clips in the Ableton Live Set happens seamlessly and in synchronisation with playback of any other tracks/parts/instruments in Ableton, but also allows for the source material for the generative process to be stored temporarily. This means that, until a different set of source material for the algorithm is inputted either from a clip or played/recorded in directly by the user, generative playback can be triggered immediately after stopping it/switching to playback from a clip etc. without the need to collect MIDI note information to compile new transition tables every time.

9.3.15 [p cliptrigpsicheck]

Main patcher → [p notegen] → [p cliptrigpsicheck]

This patcher works as a safeguard against simultaneous playback from multiple sources by opening a [gate] every time one of the clips on the TouchOSC GUI is triggered, through which the value indicating the playing slot index of the track in the Ableton Live Set is sent. The playing slot index value passes straight out of the patcher and into an [if] object that, if the value is greater than or equal to 0, sends a "0" into the [toggle] at (1) in [p notegen], thus stopping generative playback. After passing through the [gate] inside [p cliptrigpsicheck], the playing slot index value also triggers a "0" from a [t 0] that is sent back into the left inlet of the [gate] in order to close it and stop any more playing state index values passing through until a clip on the TouchOSC GUI is triggered again.

9.4 ScreenPlay-TRNS4M

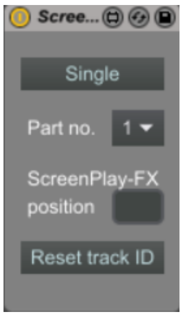


Figure 47. *ScreenPlay-TRNS4M* Max for Live MIDI Device.

9.4.1 Main patcher/ScreenPlay-TRNS4M 1.0

As with both *ScreenPlay-CTRL* and *ScreenPlay-GEN*, every aspect of *ScreenPlay-TRNS4M*'s programming can be accessed from the main patcher window via the various sub-patchers used to separate and organise the numerous distinct sections of the patch. Along with the same arbitrary setup controls found in both *ScreenPlay-CTRL* and *ScreenPlay-GEN*, such as single/multi mode, part number and reset track ID, there is also a small textbox ([textedit]) into which the user is required to input the position in the device-chain of the track in the Ableton Live Set of the *ScreenPlay-FX* Effect Rack, in order to form a connection that allows *ScreenPlay-TRNS4M* to communicate control data to it from the GUI.

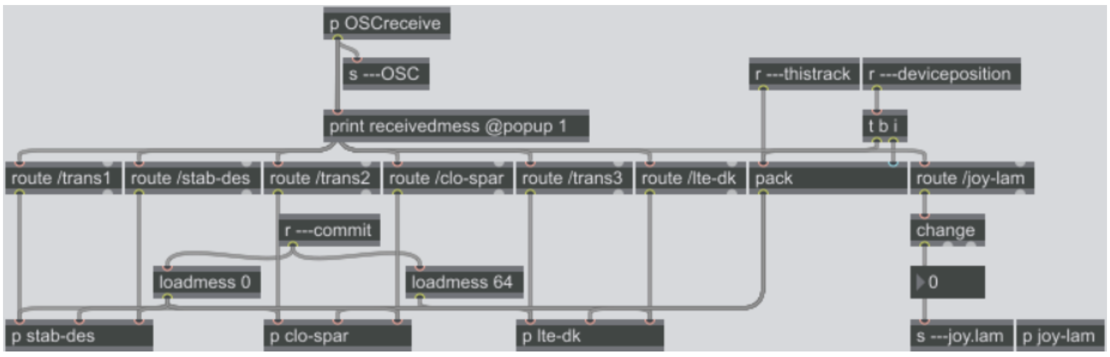


Figure 48. Core high-level programming in main patcher window.

The core programming of *ScreenPlay-TRNS4M* is found in the bottom left corner of the main patcher window, stemming from the [p OSCreceive] patcher. The [p stab-des], [p clo-spar], [p lte-dk] and [p joy-lam] patchers at the bottom of this section of programming house the programming logic used to carry out each of the four topical opposition transformations made available by *ScreenPlay*, in the form of "stability-destruction", "open-close" (formerly "closeness-sparseness"), "light-dark" and "joy-lament"; the first three being fairly simple and similar to each other in their design and the latter being far more complex.

9.4.2 [p stab-des]/[p clo-spar]/[p lte-dk]

Main patcher → [p stab-des]/[p clo-spar]/[p lte-dk]

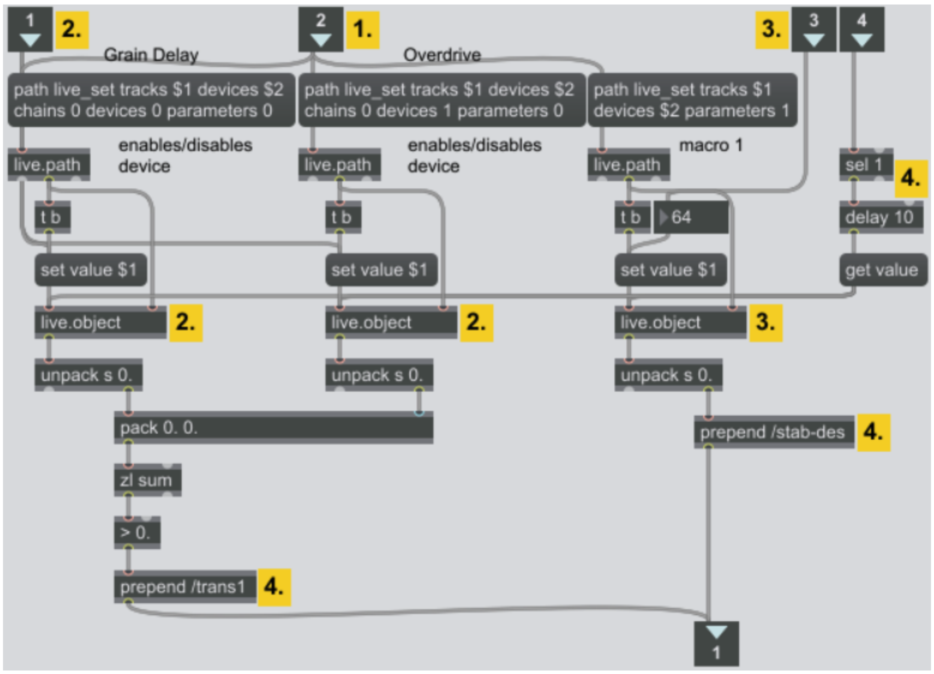


Figure 49. [p stab-des].

For the purposes of explaining the processes occurring with these three patchers the specific focus will be on [p stab-des], given that it showcases the simplest version of the programming shared by all three.

1. First, the values determining the track ID and position in the track of *ScreenPlay-FX* set by the user via the GUI of the Max for Live device during setup/calibration of the system are received at [inlet] 2 of the patcher and used to populate the variable arguments in the three message boxes. These, in turn, serve to direct the control functions sent from the TouchOSC GUI to the appropriate controls/parameters in *ScreenPlay-FX*.
2. The value of 1 or 0 received at [inlet] 1 from the "stability-destruction" activation toggle on the TouchOSC GUI is used to enable/disable specific devices within *ScreenPlay-FX* associated with the "stability-destruction" transformation. Here, there is only a Grain Delay effect and Overdrive effect, meaning this section of

programming appears only twice. In [p clo-spar] and [p lte-dk] it appears more times due to the implementation of more effects for the "open-close" and "light-dark" transformations.

3. Finally, a value of 0-127 is received from the "stability-destruction" control fader on the TouchOSC GUI, which is used to set the balance between the two topical oppositions by the user, and is sent to the appropriate macro control on *ScreenPlay-FX* - a single knob which, from there, is mapped to a number of parameters with differing ranges inside the Grain Delay and Overdrive effects.
4. Arriving at [inlet] 4 is a value of 1 or 0 from inside [p globalIP.rcv] (main patcher → [p globalIP.rcv]), which relates to whether or not a particular instrument/part is currently active when in single user mode. A 1, which indicates that the instrument/part with which a specific set of *ScreenPlay* Max for Live devices is paired has been enabled by the user either by turning on the "IP" button on the *ScreenPlay-CTRL* device GUI or selecting the track in Ableton Live that houses the instrument/clips for that part triggers a bang from [sel 1] that, in turn, triggers the "get value" function for the on/off status of Grain Delay and Overdrive and the position of the corresponding macro control in *ScreenPlay-FX*. These values pass through [prepend /trans1] and [prepend /stab-des] to result in two messages that are outputted from the sub-patcher and sent back to *ScreenPlay-CTRL* when entering [p transOSCsend] (main patcher → [p transOSCsend]) in order to update the visual feedback displayed on the TouchOSC GUI to represent the status of the newly selected part.

9.4.3 [p getnotes]

Main patcher → [p joy-lam] → [p joy-lam.transform] → [p getnotes]

Triggering the "joy-lament" toggle on the TouchOSC GUI begins the transformation process by ascertaining certain information about the active/last active clip, the indentity of which is continually monitored by the [live.observer] object inside the [p clipindex] sub-patcher (main patcher → [p clipindex]), which also appears in the *ScreenPlay-GEN* Max for Live device.

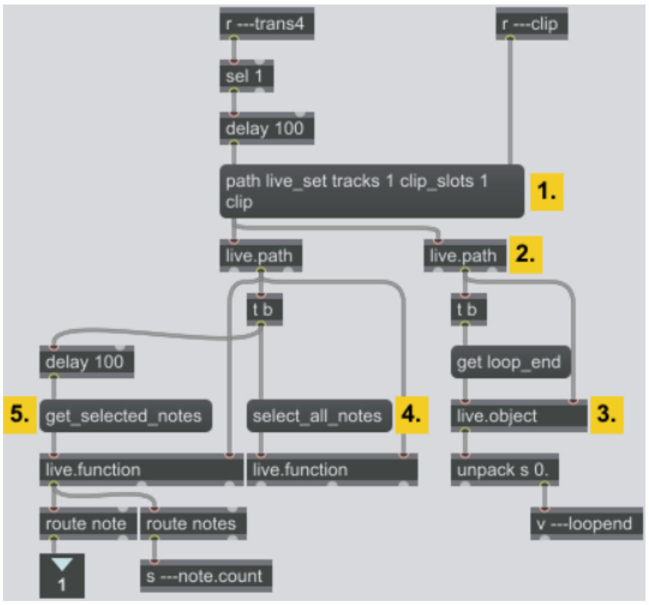


Figure 50. [p getnotes].

1. Once the playing slot index has been ascertained by [live.observer], it is packaged into the "path live_set track $1 clip_slots $2 clip" message and sent out of [p clipindex] via [s ---clip] and received in [p getnotes] by [r ---clip]. This message is inputted into the right inlet of the connected message box, thus populating it without being outputted. When a value of 1 is received from the "joy-lament toggle" on the GUI via [r ---trans4], the connected [sel 1] object detects this and outputs a bang accordingly. After being delayed by 10 ms the bang enters the left inlet of the message box and causes the output of its contents.
2. The "path live_set tracks $1 clip_slots $2 clip" message (with variables replaced) then travels through the two [live.path] objects and into the right inlet of [live.object] and the two [live.function] objects, informing them of the clip within the Ableton Live Set to be targeted when executing their respective commands. A bang is then released from the two [t b] objects, which triggers each of the command messages. The data-flow rules of right-to-left, bottom-to-top inherent to Max/MSP mean that the message identity of the target clip arrives at the right inlet of the [live.object] and [live.function]s before the command messages are triggered by [t b].
3. It is this rule also that means the first function to be executed is "get loop_end", the value of which is stored in [v ---loopend].
4. All notes within the clip are selected.

5. The total note count of the clip followed by the pitch, note-on time, duration, velocity and mute values for each individual note are outputted following the execution of the "get_selected_notes" function. The note count is outputted as an integer preceded by the word "notes", while the lists of parameters associated with each individual note are all preceded by the word "note". [route] allows only values associated with the identifier described in its arguments to pass through; therefore, the total number of notes is sent out of the patcher via [s ---note.count], while the lists of values associated with each individual note are sent separately out through the [outlet] of the patcher.

9.4.4 [p velocity.store]

Main patcher → [p joy-lam] → [p joy-lam.transform] → [p velocity] → [p velocity.store]

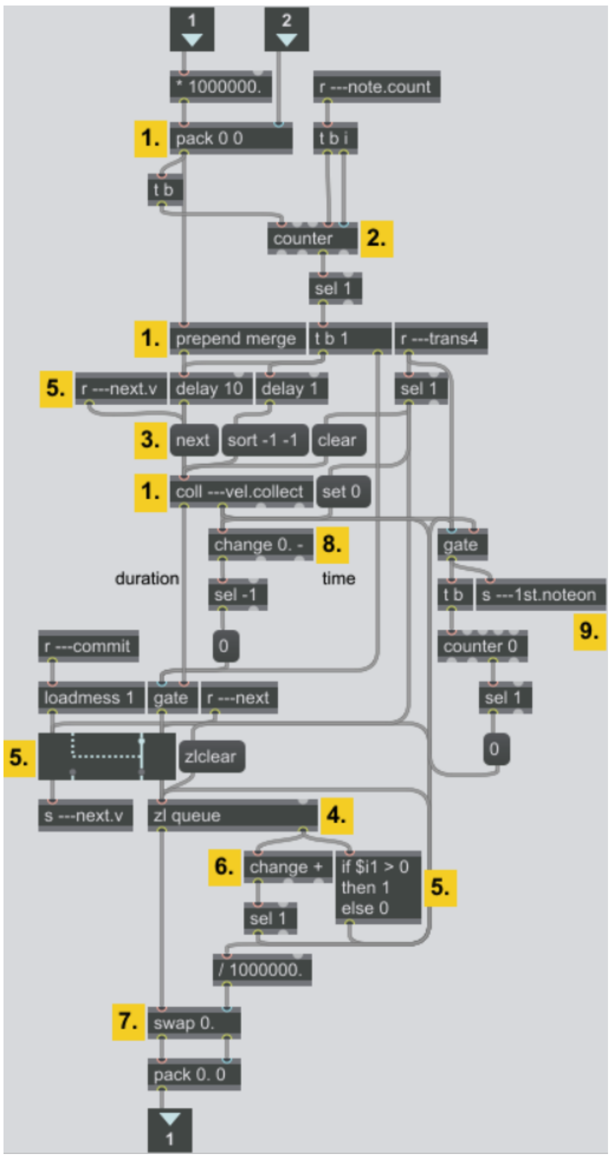


Figure 51. [p velocity.store].

Once the list of values for each note has been unpacked into its individual constituent elements inside [p joy-lam.transform] (main patcher → [p joy-lam] → [p joy-lam.transform]), the first value to be processed is velocity. The value enters [p velocity] through its right inlet and then, again, [p velocity.store] through its right inlet, with note-on time entering both sub-patchers through their left inlet.

1. Upon entering the patcher via [inlet] 2 the velocity value is directed into the right inlet of [pack 0 0], where it awaits the arrival of the note-on time from [inlet] 1 after it has been multiplied by 1000000. The list outputted from pack upon the arrival of the note-on time is sent through the [prepend merge] object and finally into [coll ---vel.collect]. The [coll] object stores data in the format of an index value followed by individual elements associated with that index. Here, the index value is the note-on time, while the elements are the velocities associated with a particular time. An index must be an integer value and so it is necessary to multiply the note-on time by 1000000 ([* 1000000]), as the "get_selected_notes" function carried out inside [p getnotes] outputs note-on time as a six-decimal floating point value. By passing the list of "note-on; velocity" through the [prepend merge] object before it arrives at the [coll], the command "merge" is added on the the beginning of the list message, meaning that velocities with the same note-on time as each other are compiled as multiple elements within the same index.
2. At the same time, each "note-on; velocity" pairing outputted from [pack 0 0] also triggers a bang when entering [t b], which, in turn, enters the leftmost inlet of a [counter] object; the functionality of which is to record the number of bangs/messages arriving at its leftmost inlet in real time. The value received from [r ---note.count] is used to set the maximum value of the [counter], meaning it will output a 1 from its third outlet when this value is reached. The number of notes received from [r ---note.count] first enters the [t b i] object, which proceeds to output the value arriving at its inlet from its right outlet, and then release a bang through its left outlet. The bang enters the fourth inlet of [counter], resetting it to a value of 0 to ensure the maximum value will only be reached once velocity and note-on values for all of the notes in the clip have been stored inside [coll].
3. Once the [counter] outputs a 1 a bang is triggered by [sel 1], which then triggers a 1 and another bang. The 1 is used to open a [gate] object, allowing the elements (velocities) outputted from the leftmost outlet of [coll] to pass through when they are

recalled. The bang is delayed first by 1 ms, before triggering the "sort -1 -1" message connected to [coll] - which serves to organise the stored data in ascending order with regard to the index value (note-on time), which is necessary due to the fact that the "get_selected_notes" function carried out inside [p getnotes] recalls note information from within the clip not in chronological order but from the lowest pitch present to the highest - and then delayed again by 10 ms before triggering the "next" message in order to recall the first element stored inside the [coll]. The [delay 1] and [delay 10] objects are necessary in order to provide sufficient time, first, for all "note-on; velocity" messages to arrive at [coll] before reorganising them in ascending order, and second, to allow the reorganisation to occur before recalling the first element (velocity of the first note in the clip). The "clear" message, also seen here connected to the [coll] object, is triggered when the "joy-lament" toggle on the TouchOSC GUI is first activated ([r ---trans4]-[sel 1]) and empties the [coll] of any previously stored data in preparation for the arrival of the note-on and velocity information of the clip to be transformed.

4. After the velocity values outputted from the left outlet of [coll] pass through the [gate] they enter the left inlet of a [zl queue] object, which stores all values that arrive at its left inlet and then outputs them one at a time through its left outlet upon receiving a bang. The reason for this is that all the elements associated with a particular index within [coll] are outputted simultaneously as a list, and the transformation process is carried out on a single note at a time. Like the [coll] preceding it, the [zl queue] is also emptied of any previously stored data when the "joy-lament" toggle on the GUI is triggered by sending it the "zlclear" message.
5. The right outlet of [zl queue] outputs an integer denoting the number of elements by which the object is currently occupied. This value first enters an [if] object that determines whether or not the value is greater than 0 and outputs a 1 if it is and a 0 if it is not. This value is then directed into the left inlet of a [gswitch2] object, through which a bang received from [p joy-lam.compile] via [r ---next] once the transformation of the previous note has been completed is routed. If [gswitch2] receives a 1 (i.e. [zl queue] is currently occupied by one or more velocity values), then the bang received from [r ---next] is routed through the right outlet of [gswitch2] and into the left inlet of [zl queue], thus triggering out the next stored velocity. If there are no velocities left inside [zl queue] and [gswitch2] receives a 0, the bang from [r

---next] is routed through the left outlet of [gswitch2] and sent via [s ---next.v] to [r ---next.v], which triggers the "next" message connected to [coll] and causes the velocity value(s) associated with the next note-on time to be released into [zl queue].

6. The value taken from the right outlet of [zl queue] is also sent into a [change +] object, which outputs a 1 when the incoming value is greater than the last. This causes [sel 1] to release a bang that is routed back into the left inlet of [zl queue], ensuring that, when new velocity values arrive at [zl queue] instead of a bang received from [r ---next] via the right outlet of [gswitch2], the first of the new velocities to arrive is released immediately.
7. As the velocity values are released one by one from [zl queue] they enter a [swap 0.] object, into which the note-on times taken from the second outlet of [coll] are also sent - after being divided by 1000000 in order to return them to their original six-decimal floating point format. [swap 0.] swaps the two values around and sends them into a [pack 0. 0] object, creating a "note-on; velocity" list and outputting it from [p velocity.store].
8. The index values from [coll] (note-on times), outputted from its second outlet, are also sent into a [change 0. -] object, which outputs -1 when the most recently received value is less than the previous value and triggers a bang from [sel -1], sending a "0" to the [gate] and causing it to close. This ensures that, once the velocity values for all the notes inside the clip have been processed and the bang received by [r ---next.v] at (5) causes the "next" message to be sent to the [coll] and recall the velocity value(s) associated with the first note-on time in the clip for a second time, they do not pass into [zl queue] at (4) and the transformation procedure is brought to an end.
9. The first index value stored inside [coll] - i.e. the first note-on time in the clip - is also sent through a [gate], which is opened when [r ---trans4] receives a 1 indicating that the "joy-lament" toggle on the TouchOSC GUI has been activated, and outputted from the patcher via [s ---1st.noteon]. Only the very first note-on time from [coll] is allowed through the [gate] as, when the value passes through the [gate] and into the [send] object, it also triggers a bang from [t b], which enters a [counter 0], immediately causing the object to reach its maximum value and output a "1" from its third outlet, which then causes [sel 1] to trigger a "0" message that is directed back into the left inlet of the [gate], thus closing it again.

9.4.5 [p velocity]

Main patcher → [p joy-lam] → [p joy-lam.transform] → [p velocity]

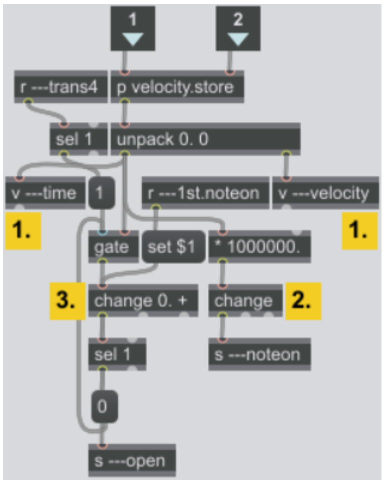


Figure 52. [p velocity].

1. Because note transformations only affect pitch and duration, not velocity and note-on time, these values for each note are inputted into [v ---velocity] and [v ---time] respectively, to be recalled once the pitch and duration transformations are complete.
2. The note-on time is also sent via [s ---noteon] into [p pitch.store] after, again, being multiplied by 1000000, where it is used to register in a [coll] the previous note to the one currently being transformed once a transformation has occurred on the previous note (i.e. its pitch is no longer what it originally was when it was entered by the user into the clip) after passing through a [change] object. [change] only allows a value to pass through if it is different to the value before it.
3. Note-on time also passes through a [gate], which is opened when [r ---trans4] receives a 1, indicating that the "joy-lament" toggle on the GUI has been activated, and then into a [change 0. +], which outputs a 1 when it receives the second note-on time contained in the clip after its initial value has been set to that of the first note-on time in the clip received from [p velocity.store] via [r ---1st.noteon] and passed through the "set $1" message. The 1 outputted from [change +] when it receives the second note-on time is first sent via [s ---open] to [p gate-ctrl] and [p joy-lam.compile] and then into [sel 1], which triggers a "0" message that, in turn, is routed back into the [gate] in order to close it.

9.4.6 [p gate-ctrl]

Main patcher → [p joy-lam] → [p joy-lam.transform] → [p gate-ctrl]

This patcher serves to ensure that the original pitch of the note(s) in the clip associated with the first note-on time preserve their pitch in order to keep the transformation grounded in relation to the characteristics of the original musical phrase.

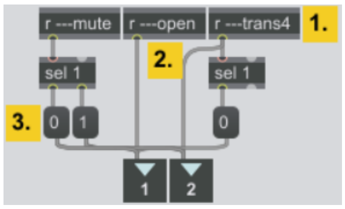


Figure 53. [p gate-ctrl].

1. When the "joy-lament" toggle on the GUI is activated, [p gate-ctrl] outputs a 0 via [outlet] 1, closing the necessary [gate]s inside [p pitch] and disabling any transformation calculation that would otherwise take place on the pitch value of the current note in order to keep the transformed musical phrase somewhat faithful to the original. Because duration calculations are allowed to occur for the first note(s) of a clip, the "1" received by [r ---trans4] when the transformation process is activated is sent directly out of [outlet] 2 and into [p duration], where it opens the necessary [gate] objects.
2. Once the note(s) associated with the first note-on time have been processed and [r ---open] has received a 1 from [change 0. +] at (3) inside [p velocity], this 1 is directed out of the first [outlet] and sent into the [gate] objects inside [p pitch] in order to open them up and allow pitch transformations to occur.
3. If the calculation process relating to the time difference between the note-off time of the previous note and the note-on time of the current note - which occurs from (5)-(8) inside [p nextnote.calc] - ascertains that the current note should be muted, then [r ---mute] receives a "1" from [p duration] and a "0" is sent out of both [outlet]s 1 and 2, closing the corresponding [gate] objects inside [p pitch] and [p duration] and stopping any transformation calculation from occurring. A "0" received at [r ---mute], indicating the current note should be unmuted, outputs a "1" through both [outlet]s of the patcher and, therefore, opens the [gate]s.

9.4.7 [p duration.store]

Main patcher → [p joy-lam] → [p joy-lam.transform] → [p duration] → [p duration.store]

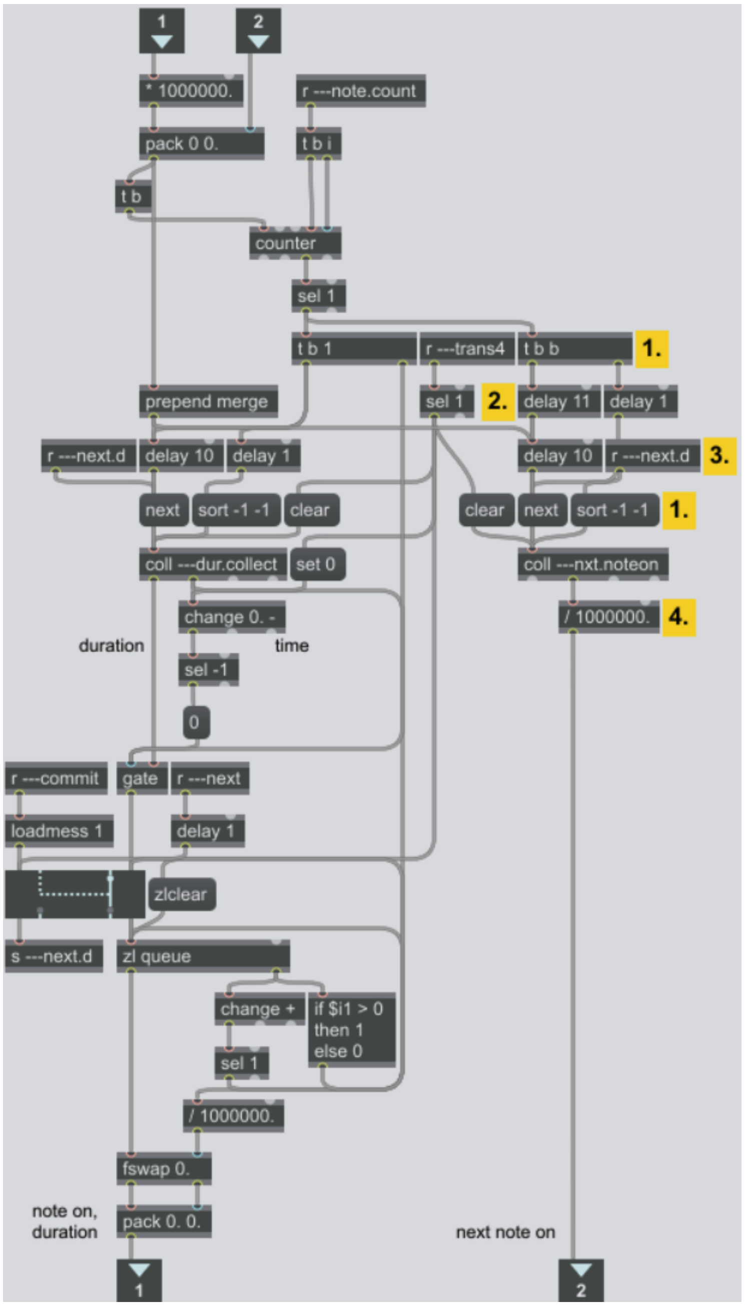


Figure 54. [p duration.store].

For the most part, [p duration.store] is exactly the same as [p velocity.store], except for the fact that it also uses [coll. ---next.noteon] to store and output the note-on time of the following note in order to decide if the difference between the note-off time of the current note and the note-on time of the next note should result either in passing notes being added between the two notes or, if the note-off time overlaps that of the next note-on time by a certain threshold, the next note being muted.

The storing and recalling of values from [coll ---next.noteon] works in the same way as it does for [coll ---dur.collect] and [coll ---vel.collect], with the addition of an extra bang in [t b b] and [delay], which are triggered once the note-on times for all the notes within the clip have been collected by the [coll].

1. As before with [p ---dur.collect] and [p ---vel.collect], the first bang outputted from the right outlet of [t b b] passes through [delay 1] and then triggers the "sort -1 -1" message, organising the contents of the [coll] in ascending order with regard to the index (note-on time) value.
2. Whereas with [p ---dur.collect] and [p ---vel.collect], where the same bang then passes through a [delay 10] before triggering the "next" message and outputting from the [coll] the first stored index and its elements, the second bang from [t b b] also follows this path before also passing through a [delay 11] and triggering the "next" message again; effectively bypassing the first note-on time and jumping straight to the second.
3. From this moment on, every time the "next" message is triggered by the bang received by [r ---next.d], [coll ---nxt.noteon] outputs from its second outlet the note-on time that follows that which is outputted by [p ---dur.collect], [p ---vel.collect] (and also [p ---pitch.collect], which has not yet been touched upon).
4. The note-on time released by [coll ---nxt.noteon] is then divided by 1000000 in order to return it to its original six-decimal floating point form and outputted from [p duration.store] through its right [outlet].

9.4.8 [p duration]

Main patcher → [p joy-lam] → [p joy-lam.transform] → [p duration]

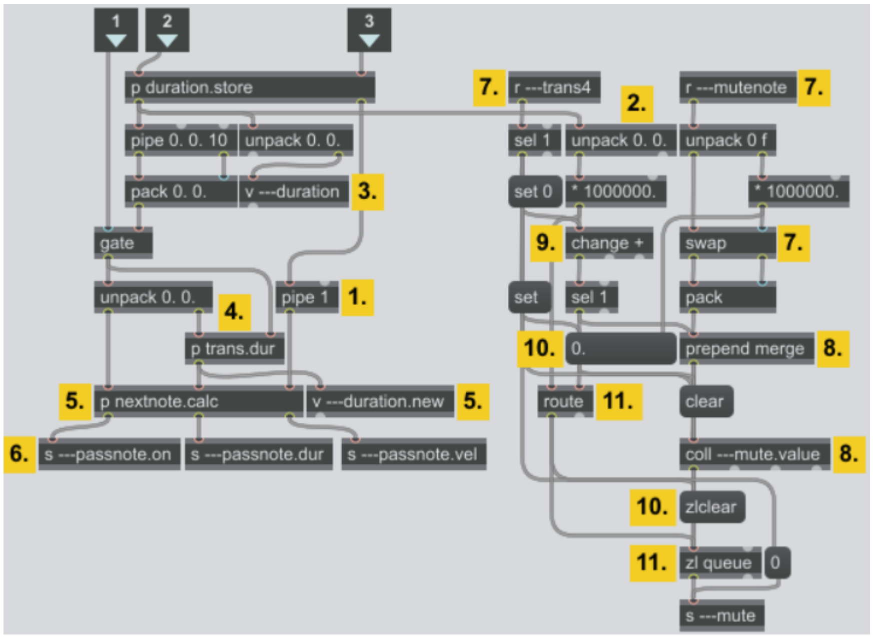


Figure 55. [p duration].

1. Upon leaving the right outlet of [p duration.store] the next note-on time is delayed by 1 ms by [pipe 1] before entering the right inlet of [p nextnote.calc]; the patcher responsible for calculating whether passing notes should be added or the next note should be muted.
2. The "note-on; duration" list outputted through the left outlet of [p duration.store] is first unpacked and the note-on time multiplied by 1000000, before being used in the process for outputting the mute values (1 if note is muted, 0 if unmuted) generated inside [p nextnote.calc] for each note in the clip - to be explained later.
3. Secondly, the list is unpacked and the duration value is stored in [v ---duration] from where, if there is no change in duration applied to the current note during the transformation process, it can be outputted at the point of compiling the final note list inside [p joy-lam.compile].
4. Finally, after being delayed by 10 ms by [pipe 0. 0. 10], the "note-on; duration" list arrives at the right inlet of a [gate] - the state of which is subject to the value outputted from the right outlet of [p gate-ctrl], which, in turn, is dictated by the original mute value of the current note - (3) inside [p gate-ctrl]. If it is calculated during the transformation process that the note should be muted as a result of the transformation of the previous note then the [gate] is closed - ensuring the original duration of the

note is retained. If it is calculated that the note should be unmuted then the [gate] is open - allowing the list to pass through and into the right inlet of [p trans.dur] in its entirety. The list is also unpacked and the duration value alone is directed into the left inlet of [p trans.dur]; so as to calculate whether or not the duration will be elongated, shortened or remain unaltered. ***Refer to 9.4.9 [p trans.dur].***

5. Once the new duration has been calculated and outputted from [p trans.dur] it is first sent into [v ---duration.new] to be recalled at the end of the transformation process for the current note in order to compile the final note MIDI message list. Secondly, it enters the middle inlet of [p nextnote.calc] along with the note-on time, which enters through the left inlet, and the next note-on time through the right inlet. ***Refer to 9.4.12 [p nextnote.calc].***
6. Once the velocity, duration and note-on times for the passing notes have been generated inside [p nextnote.calc] they are outputted from the [patcher] in that order and sent to [p joy-lam.compile] via [s ---passnote.vel], [s ---passnote.dur] and [s ---passnote.on] respectively in order to be inserted into the clip.
7. If it is ascertained at (5) inside [p nextnote.calc] that the next note is to be muted/remain unmuted, [r ---mutenote] receives a "1" or "0" respectively, along with the next note-on time. This list is unpacked into its constituent elements before the next note-on time is multiplied by 1000000 and the two elements are swapped around and packed into a new list in the order of "next note-on; mute value".
8. The newly formed list then passes through the [prepend merge] object before entering [coll ---mutenote.value], which ensures all mute values for notes with the same note-on time are stored under the same index within the [coll]. The [coll] object is cleared of any data that may have been stored in it from a previous transformation when the new transformation is first triggered. [r ---trans4] receives a "1", which triggers a bang from [sel 1] and, in turn, sends the "clear" message to the [coll]. This is also true of the [zl queue] object at (11), which has the "zlclear" message sent to it.
9. Now, the (note-on time x 1000000) at (2) enters a [change +] object, the starting value of which is set to 0 when the transformation is first triggered ([r ---trans4]-[sel 1]-"set 0"), and - assuming it is not the first note(s if they have the same note-on time), which cannot be muted - causes the output of a "1", which triggers a bang from [sel 1].
10. This has two consequences. First, the "zlclear" message is sent to the [zl queue] object, clearing it of any information that remains from the note(s) at the previous note-on

time. Next, it triggers the message box below which has been populated by the (next note-on time x 1000000) taken from [unpack 0 f]-[* 1000000] at (7); the output of which is first sent into the [coll] at (8) - recalling all the elements associated with that index value - and then into the right inlet of a [route] object, thus setting its arguments.

11. The list of elements for that note-on time is then stored in [zl queue] and outputted one at a time by the bang released from the [route] object. [route] first triggers a "0" (to the right of [zl queue], which is overridden by the outputted value from [zl queue] but serves to ensure any notes without a mute value generated for them remain unmuted and are not potentially omitted from the clip entirely following the transformation. Individual mute values outputted by [zl queue] are sent to [p joy-lam.compile] (among other places) where they are added to the final MIDI message list for their corresponding note and inserted into the clip once the transformation of that note has been completed.

This process ensures that every note in the clip has assigned to it a mute value that has been specifically generated for it. By populating the message at (10) with the (next note-on time x 1000000) from (7) (a process which takes places during the transformation of the note(s) from the previous note-on time - the *current* notes - inside [p nextnote.calc]), it ensures that the mute values stored in the [coll] object at (8) for the next note(s) are not outputted and applied to any notes until the *current* (note-on time x 1000000) at (2)/(9) is the same as the *previous* (next note-on time x 1000000) at (7) - i.e. the storing and outputting of mute values from the [coll] at (8) and [zl queue] at (11) is staggered over two constantly changing note-on times. ***Refer to 9.4.18 [p pitch.store].***

9.4.9 [p trans.dur]

Main patcher → [p joy-lam] → [p joy-lam.transform] → [p duration] → [p trans.dur]

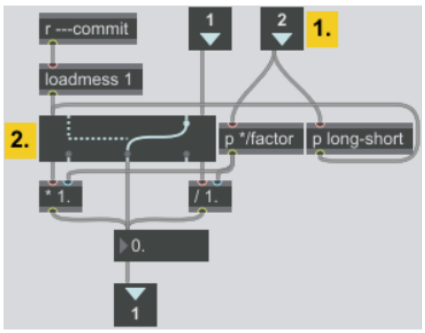


Figure 56. [p trans.dur].

1. Upon entering [p trans.dur] through its right inlet, the "note-on; duration" list first enters [p long-short], which calculates whether the note should be elongated, shortened or remain the same length, and then enters [p */factor], which calculates the factor by which the note should be elongated or shortened in relation to its original duration. ***Refer to 9.4.10 [p long-short].***
2. The value outputted by [p long-short] (0, 1 or 2) is directed into the left inlet of the [gswtich2] object. If a 0 is received, the duration value that arrives at the first [inlet] of [p trans.dur] is directed out of the left outlet of [gswitch2] and multiplied by the factor outputted from [p */factor] (once it has been calculated) before being outputted from [p trans.dur]; thus being elongated. If a value of 1 is received at the left inlet of [gswitch2] the duration is directed straight back of out [p trans.dur] unaltered. If a value of 2 is received at the left inlet of [gswitch2] the duration value is divided by the factor outputted from [p */factor] before being outputted from [p trans.dur]; thus being shortened. ***Refer to 9.4.11 [p */factor].***

9.4.10 [p long-short]

Main patcher → [p joy-lam] → [p joy-lam.transform] → [p duration] → [p trans.dur] → [p long-short]

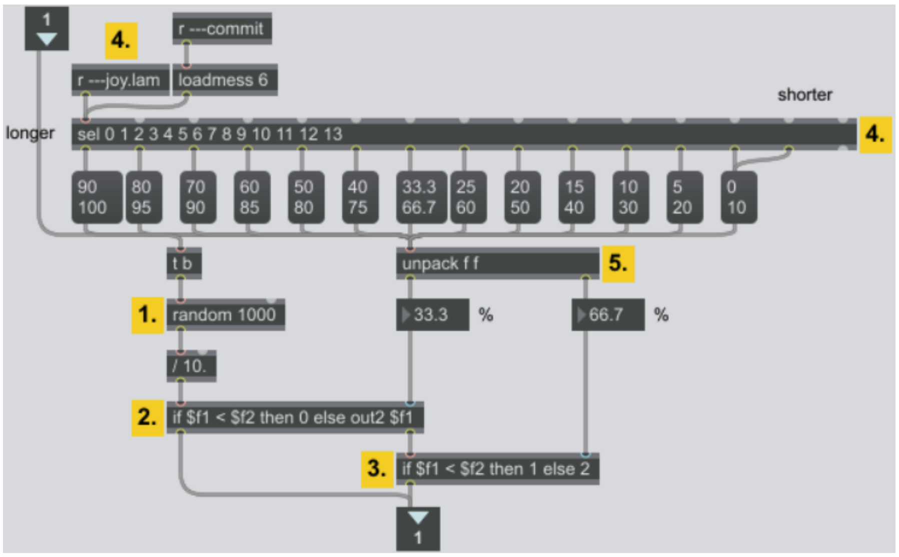


Figure 57. [p long-short].

1. Once it has entered [p long-short], the "note-on: duration" list triggers a bang from [t b], which, in turn, causes [random 1000] to generate a random number from 0-999.

2. The newly generated value is divided by 10 when passing through [/ 10.] to give a single decimal place floating point number, before entering an [if] object through its left inlet; the purpose of which is to compare two values with regard to the commands laid out in its arguments. Here, the arguments [if $f1 < $f2 then 0 else out2 $f1] indicate that, should the value entering the object through its left inlet be less than that entering through the right inlet, a value of 0 should be outputted from the left outlet of the object, which is then sent out of [p long-short]; otherwise, the incoming value received by the left inlet passes through the object and is outputted from its right outlet, from where it enters a second [if] object through its left inlet.
3. Again, the comparison as to whether the value arriving at the object's left inlet is less than that arriving at its right inlet is made, although this time outputting a 1 if the first value is less than the second and a 2 if it is not; which is then sent out of [p long-short].
4. The value that enters the right inlet of the two [if] objects and against which the value entering the left inlet is compared is determined by the position of the "joy-lament" slider on the TouchOSC GUI. The value outputted by the slider, which ranges from 0-13, is received by [r ---joy.lam] and enters the [select] object - the arguments for which range from 0-13. When a bang is sent out from one of the outlets of [sel] it triggers a message containing a list of two single decimal place floating point values. These values represent the percentage probability that the note will either be shortened, elongated or remain unchanged. For instance, when a value of 6 is sent from the "joy-lament" slider on the GUI (the middle value given that values of 12 and 13 both trigger the final set of percentages - the reason for which is necessary but irrelevant in discussing how the process works) a bang is released from the seventh outlet of [sel] and triggers the "33.3; 66.7" list.
5. The list is then unpacked into its separate elements, with the first and second values being sent into the right inlet of their respective [if] objects. The result here is that there is an even chance that the note length will be increased, decreased or left unchanged. If the value generated at (1) by [random 1000] is: (a) < 33.3 - a 0 is outputted from [p long-short] and the note length is increased; (b) >= 33.3 but < 66.7 - the value passes through the right outlet of the first [if] object and into the second, causing it to output a 1, which is sent out of [p long-short] and results in the original

duration of the note being retained; (c) >= 66.7 - the second [if] object outputs a 2 and sends it out of [p long-short], causing the note length to be decreased.

This is the basic outline of the process by which all probability-based decisions occur in the "joy-lament" topical opposition transformative algorithm, albeit with some minor variations, differing probabilities, numbers of potential outcomes and, therefore, numbers of [if] objects and list lengths triggered by the [select] object at (4). In this instance, when the "joy-lament" slider on the GUI is positioned all the way at "joy", there is a 10% chance that the note length will remain unchanged and a 90% chance it will be decreased; at the opposite end of the spectrum it is a 90% chance the note length will be increased. ***Refer to 9.4.9 [p trans.dur], (2).***

9.4.11 [p */factor]

Main patcher → [p joy-lam] → [p joy-lam.transform] → [p duration] → [p trans.dur] → [p */factor]

This particular probability calculation sub-patcher is unique from all the others in *ScreenPlay-TRNS4M*, due to the fact that the probability ranges triggered by the [select] object are the same on either side of the centre point. The reason for this is that, as the slider is moved towards either "joy" or "lament", the factor of division/multiplication needs to increase in both cases in order for the likelihood of the note being made shorter or longer to increase. The possible factors of division/multiplication are 1.5, 2 and 3, with the extremes of the "joy-lament" scale allowing a 90% chance for a factor of 3, 10% for a factor of 2 and 0% for a factor of 1.5. ***Refer to 9.4.8 [p duration], (5).***

9.4.12 [p nextnote.calc]

Main patcher → [p joy-lam] → [p joy-lam.transform] → [p duration] → [p nextnote.calc]

The purpose of [p nextnote.calc] is to calculate, depending on the duration between the note-off time of the current note and the note-on time of the next, whether or not to mute the next note or insert passing notes between the two notes.

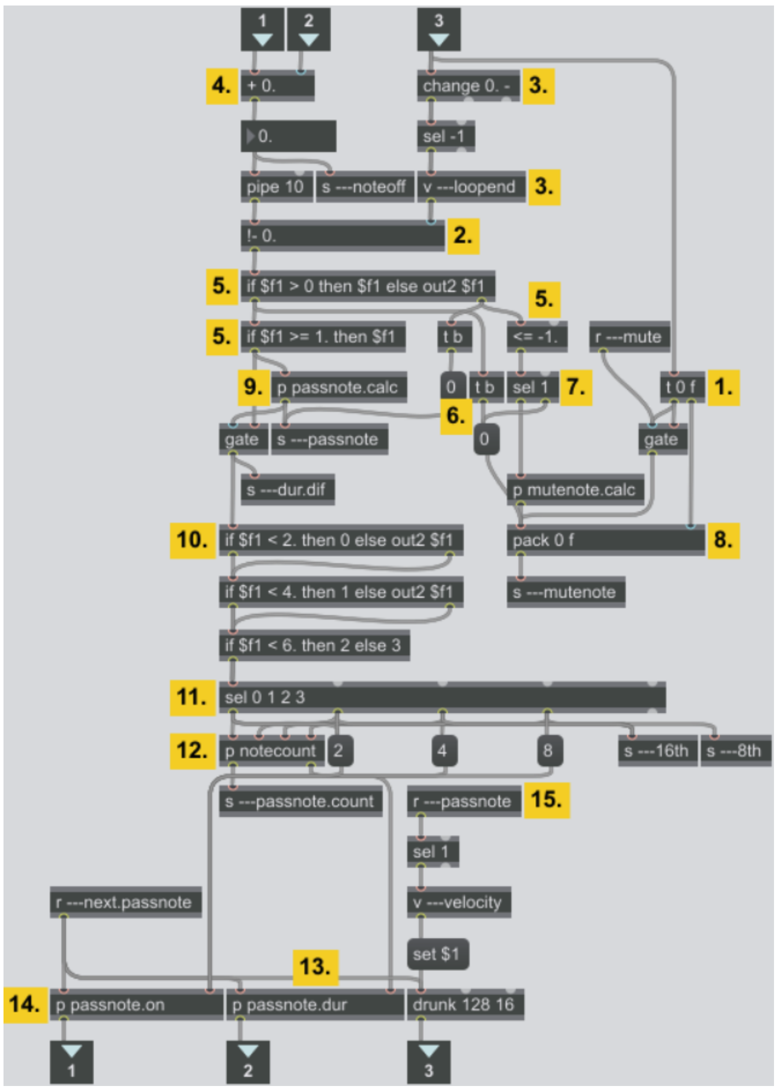


Figure 58. [p nextnote.calc].

1. When arriving at the third [inlet] of [p nextnote.calc], the next note-on time is sent to three different locations. First, it arrives at a [t 0 f] object, which outputs the incoming floating point value from its right outlet and a 0 from its left outlet. The outputted floating point value enters the right inlet of [pack 0 f] and awaits the arrival of the mute value to be assigned to it, which is calculated inside [p mutenote.calc]. The 0 enters the right inlet of a [gate] followed immediately by the left inlet. By default the [gate] is closed, meaning that the 0 is not able to pass through. If, however, the previous note has already been muted, a 1 arrives at the left inlet of the [gate] via [r ---mute], thus opening it and allowing the 0 to pass through and into the left inlet of [pack 0 f], which, in turn, releases the newly compiled list of "mute value; next note-on time" and sends it out of [p nextnote.calc] via [s ---mutenote]. The reason for this is that, if a note has already been muted, it is not subject to transformation and, as

such, the calculation as to whether or not the next note should be muted is bypassed and it retains its default unmuted state. Once the 0 from [t 0 f] has passed through the right inlet of the [gate] and into [pack 0 f] it immediately enters the left inlet of the [gate], thus closing it again until another muted note causes it to be re-opened by a 1 arriving at [r ---mute].

2. Secondly, the next note-on time enters the right inlet of [!- 0.] and sets the argument of the object. In Max the "!" reverses the standard functionality of an object, meaning that when a value arrives at the left inlet of [!- 0.] it is subtracted from the value denoted in the object's arguments and the result is outputted, as opposed to the value set in the objects arguments being subtracted from the value arriving at the left inlet.
3. Finally, the next note-on value arrives at [change 0. -], which, in turn, is connected to [sel -1] and then [v ---loopend]. The purpose of this is so that, when the last note of the clip is being transformed and the next note-on time released by [coll ---nxt.noteon] inside [p duration.store] is that of the first note in the clip and, therefore, the value arriving at the third [inlet] of [p nextnote.calc] is lower than the previous value, the time in beats for the end of the clip will be sent into the right inlet of [!- 0.] instead of the note-on time of the first note in the clip.
4. Next, the note-on time and duration values arriving at [inlet]s 1 and 2 respectively are added together when passing through [+ 0.], in order to ascertain the note-off time of the current note. This value is then sent into [p passnote.on] via [s ---noteoff] before being delayed by 10 ms by [pipe 10] and entering the left inlet of [!- 0.] at (2) in order to be subtracted from the next note-on time and calculate the difference between the two values.
5. If the value outputted from [!- 0.] is greater than 0 (i.e. the note-off time of the current note comes before the note-on time of the next note) then it is outputted via the left outlet of the [if] object. The value then enters another [if] object which, this time, determines whether the value is greater than or equal to 1. (i.e. a quarter note); in which case it is allowed to pass through the object in order for a calculation to occur that decides whether or not to insert passing notes in the gap between the two notes and, if so, how many. If the value outputted by [!- 0.] is less than or equal to 0 it is outputted from the [if] object via its right outlet, this time entering [<= -1.] to determine whether or not the note-off time of the current note overlaps the note-on time of the next note by a duration greater than or equal to a quarter note.

6. In both scenarios, whether the value from [!- 0.] is outputted from the first [if] object at (5) via the left or right outlet, a [t b] causes a "0" to be sent via either [s ---mutenote] - after entering the left inlet of [pack 0 f] - or [s ---passnote]; indicating either that the current note is to remain unmuted if passing notes are potentially to be added or that there will be no passing notes inserted between the two notes if the next note is potentially to be muted.
7. In the case that the note-off time of the current note does overlap the note-on time of the next note by at least a quarter note, [<= -1.] releases a value of 1, which, in turn, causes [sel 1] to release a bang from its left outlet and enter [p mutenote.calc] where it triggers a calculation to determine whether or not the next note will be muted. If the note-off time of the current note overlaps the note-on time of the next by less than a quarter note, [<= -1.] releases a 0, which passes through the right outlet of [sel 1] and causes a "0" to be sent into the left inlet of [pack 0 f], thus indicating that the next note will not be muted. ***Refer to 9.4.13 [p mutenote.calc].***
8. Once the calculation whether or not to mute the next note has been carried out inside [p mutenote.calc] the resulting value (1 for muted, 0 for unmuted) is directed into the left inlet of [pack 0 f], where it is compiled into a list of "mute value; next note-on time" and outputted from [p nextnote.calc] via [s ---mutenote].
9. In the event that the note-off time of the current note falls short of the next note-on time by at least a quarter note the time difference value outputted by [!- 0.] at (2) passes through the two [if] objects at (5) and triggers the calculation inside [p passnote.calc] to determine whether or not passing notes should be added in between the two notes. ***Refer to 9.4.14 [p passnote.calc].*** The outcome of this calculation (1 for passing notes, 0 for no passing notes) is first sent elsewhere via [s ---passnote] before entering the left inlet of a [gate] object and allowing the time difference value to pass through the right inlet of the [gate] if passing notes are to be added, or blocking the path if they are not.
10. After passing through the [gate] the time difference between notes is first sent via [s ---dur.dif] to be used in the calculation process inside [p passnote.on] before entering a series of [if] objects, which output a 0 if the time difference is less than two beats, a 1 if it is greater than or equal to two beats but less than four beats, a 2 if it is greater than or equal to four beats but less than six beats, and a 3 if the difference is greater than or equal to six beats.

11. The values outputted by the [if] objects then enter a [sel] object and are used to determine the potential number of passing notes to be added, their potential note-on times, and the quantization value to which they should adhere. If the [sel] object receives a 0 (time difference < 2.): the bang outputted by the first outlet of [sel] is sent via [s ---16th] to [p passnote.on] to inform the calculation that all generated note-on times should be quantized to the nearest 16th note; the bang then triggers a "2", which is directed into the right inlet of [p passnote.on] and causes the time difference between the current note-off time and the next note-on time to be divided by two in order to ascertain the possible note-on times to be made available when generating notes; finally, the bang enters the first inlet of [p notecount], which calculates both the number of passing notes to be added and the range of potential durations to be available during the generation of each passing note. If [sel] receives a 1 (time difference >= 2. and < 4.): the bang is released from the second outlet of [sel], is, again, sent via [s ---16th] to [p passnote.on]; "2" is again sent into right inlet of [p passnote.on]; bang enters second inlet of [p notecount]. If [sel] receives a 2 (time difference >= 4. and < 6.): bang sent via [s ---16th]; "4" sent into right inlet of [p passnote.on]; bang enters third inlet of [p notecount]. If [sel] receives 3: bang sent via [s ---8th] into [p passnote.on]; "8" sent into right inlet of [p passnote.on]; bang enters fourth inlet of [p notecount]. ***Refer to 9.4.15 [p notecount].***
12. [p notecount] outputs two separate integer values. The first, outputted from its right outlet and sent to the right inlet of [p passnote.dur], relates to the possible range of durations to be made available when generating passing notes. The second, outputted via its left outlet and sent to both [p joy-lam.compile] and [p replacenotes] via [s ---passnote.count] represents the number of passing notes that are to be added to the clip in between the current note and the next.
13. The calculation that occurs inside [p passnote.dur] in order to determine the duration of each individual passing note is triggered by a bang received from (6) inside [p joy-lam.compile] via [r ---next.passnote]. It is originally triggered inside [p replacenotes] once all the newly generated information for the current note has been compiled into a single list and received in [p joy-lam.compile] by [r ---note.done]. The bang passes through a [gswitch2] object that, ordinarily, directs it through its left outlet into [s ---next]. Only when the [gswitch2] receives a 1 at its left inlet from [s

---passnote] at (9) inside [p nextnote.calc] does it direct the bang out through its right outlet and into [s ---next.passnote]. ***Refer to 9.4.16 [p passnote.dur].***

14. Once the passing note duration has been calculated and outputted from [p nextnote.calc] via the second [outlet] of the patcher, the calculation process inside [p passnote.on] is also triggered by the bang arriving at [r ---next.passnote] in order to determine the note-on time for the passing note; before being outputted from the patcher via the first [outlet]. ***Refer to 9.4.17 [p passnote.on].***
15. The other calculation to be triggered by the bang received from [r ---next.passnote] is that which generates the velocity values for the passing notes. The velocity values of the original notes in the clip remain unaltered, but a very simple system to create a range of velocities when generating passing notes is implemented. First, when it is established at (9) that passing notes will be added between the current note and the next note in the clip, [r ---passnote] receives a "1" that triggers a bang from [sel 1], which, in turn, triggers the output of the velocity value of the current note in the clip from [v ---velocity]. This value then replaces the "$1" argument in the "set $1" message and is sent into the left inlet of a [drunk 128 16] object. The [drunk] object is extremely similar to the [random] object in that is outputs an integer value within the range denoted by its first argument (in this case, 128 - the standard velocity range of MIDI). However, the second argument (in this case, 16) acts to restrict the value that is generated when the object receives a bang to within a specific range either side of the value that preceded it. Now, every time [drunk] receives a bang from [r ---next.passnote] it outputs a new velocity and releases it from [p nextnote.calc] via the third [outlet] that is likely different to the one before it whilst avoiding large spikes and troughs in the amplitude of the generated passing notes. ***Refer to 9.4.8 [p duration], (6).***

9.4.13 [p mutenote.calc]

Main patcher → [p joy-lam] → [p joy-lam.transform] → [p duration] → [p nextnote.calc] → [p mutenote.calc]

Because there are only two possible outcomes of the calculation carried out by [p mutenote.calc] the 0-13 range of values outputted by the "joy-lament" slider on the GUI is sent through a [scale] object, which scales the range down to 0-9. This is the same for all calculations with only two possible outcomes. The extremes of the "joy-lament" scale, in this case, have a 90% chance of the note being muted for "lament" and a 10% chance of the note being muted for "joy". The [if] objects that ordinarily process the results of the calculation are replaced with a [<] object, which outputs a 1 if the next note is to be muted and a 0 if it is to remain unmuted. ***Refer to 9.4.12 [p nextnote.calc], (8).***

9.4.14 [p passnote.calc]

Main patcher → [p joy-lam] → [p joy-lam.transform] → [p duration] → [p nextnote.calc] → [p mutenote.calc]

This [patcher] is exactly the same as [p mutenote.calc], albeit with the range of probabilities being reversed. Here, at the "lament" extreme of the "joy-lament" scale there is a 10% chance that a passing note will be added, while there is a 90% chance of passing notes being added at the "joy" extreme. A 1 outputted by the [patcher] indicates passing notes are to be added and a 0 indicates there are to be no passing notes added. ***Refer to 9.4.12 [p nextnote.calc], (10).***

9.4.15 [p notecount]

Main patcher → [p joy-lam] → [p joy-lam.transform] → [p duration] → [p nextnote.calc] → [p notecount]

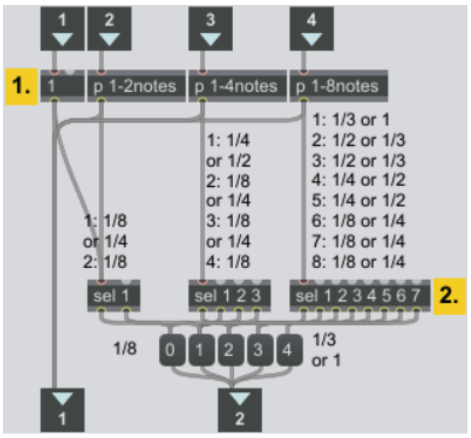


Figure 59. [p notecount].

1. The possible number of passing notes to be generated is dependent upon whether the bang from the [sel] object at (11) inside [p nextnote.calc] arrives at the first [inlet] (duration between note-off and note-on times of the current and next note respectively >= 1. beat and < 2. beats), second [inlet] (time difference >= 2. and < 4. beats), third

[inlet] (>= 4. and < 6. beats), or fourth [inlet] (>= 6.) of [p notecount]. If the bang arrives at [inlet] 1 there can only ever be 1 passing note added in the gap between notes. A bang received at [inlet] 2 triggers the calculation inside [p 1-2notes], which ranges from a 100% chance of a single passing note being generated at the extreme of "lament" to a 100% chance of two passing notes being generated at the extreme of "joy", with the probability shifting in 10% increments. The calculation inside [p 1-4 notes] is triggered if the bang arrives at [inlet] 3 and has a 55% chance of generating a single passing note and a 45% chance of generating two passing notes for "lament", and a 55% chance of four passing notes and a 45% chance of three at "joy". A bang at [inlet] 4 triggers [p 1-8notes]: "lament" - 40% one, 30% two, 20% three, 10% four; "joy" - 40% eight, 30% seven, 20% six, 10% five.

2. The number of notes to be generated is sent out of [p notecount] via the first [outlet]. It is also directed into the corresponding [select] object for each of the individual [patcher]s. The purpose of the [select] objects is to determine the potential note durations to be made available when generating the passing notes, with the notes likely to be longer in relation to the size of the time difference gap between the two notes and the closer the "joy-lament" slider is positioned towards "lament", and shorter the closer the slider is positioned towards "joy". The actual diminutions made available to each number of notes and each time difference range can be seen in the comments made inside [p notecount] itself (see Fig. 59). Once the duration range has been established the corresponding integer value message used to denote a specific duration range is outputted from the second [outlet] of the patcher. ***Refer to 9.4.12 [p nextnote.calc], (12).***

9.4.16 [p passnote.dur]

Main patcher → [p joy-lam] → [p joy-lam.transform] → [p duration] → [p nextnote.calc] → [p passnote.dur]

As there are, at most, only ever two possible note durations made available when generating passing notes for each time difference range/number of passing notes combination, the calculation process inside [p passnote.dur] appears, for the most part, very similar to all other probability calculations with only two possible outcomes. At the extremes of "lament" and "joy" there is a 90% chance that the note duration will respectively be either the longer or shorter of the two possible options. Where this patcher differs from the other probability calculations patchers with two possible outcomes is with the addition of a [gswitch2] object and multiple [sel 1] objects and duration messages following on from [<]. The outlet of the [gswitch2] through which the value outputted by [<] (1 or 0) is directed once a calculation has been triggered is determined by the integer value arriving at [inlet] 2 of the patcher from the right outlet of [p notecount], which is used to denote the range of potential durations for passing notes specific to the time difference range/number of passing notes combination. If a 1 is outputted from [<] it triggers a bang from the left outlet of the corresponding [select] object, which is connected to the shorter of the two possible durations. A 0 passes through the right outlet of the [select] object and triggers the longer of the two durations. The one exception is the [t b] connected to the leftmost outlet of the [gswitch2], which always triggers the 8th note duration regardless of the outcome of the probability calculation. This only ever occurs either when it is calculated inside [p notecount] that two passing notes should be inserted into a time difference between notes of less than a half note in duration, or four passing notes should be inserted into a time difference between notes of less than a whole note. Once the duration for the current passing note has been calculated it is outputted from [p passnote.dur] back into [p nextnote.calc]. ***Refer to 9.4.12 [p nextnote.calc], (14).***

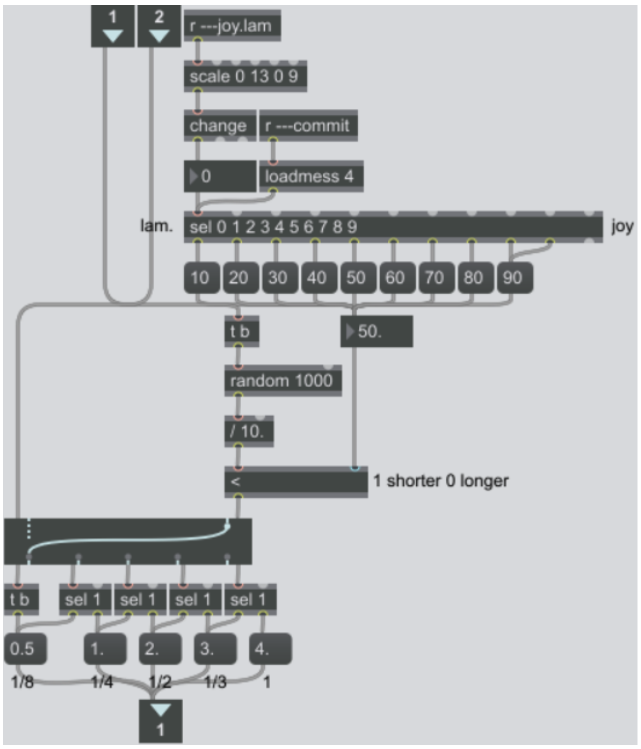


Figure 60. [p passnote.dur].

9.4.17 [p passnote.on]

Main patcher → [p joy-lam] → [p joy-lam.transform] → [p duration] → [p nextnote.calc] → [p passnote.on]

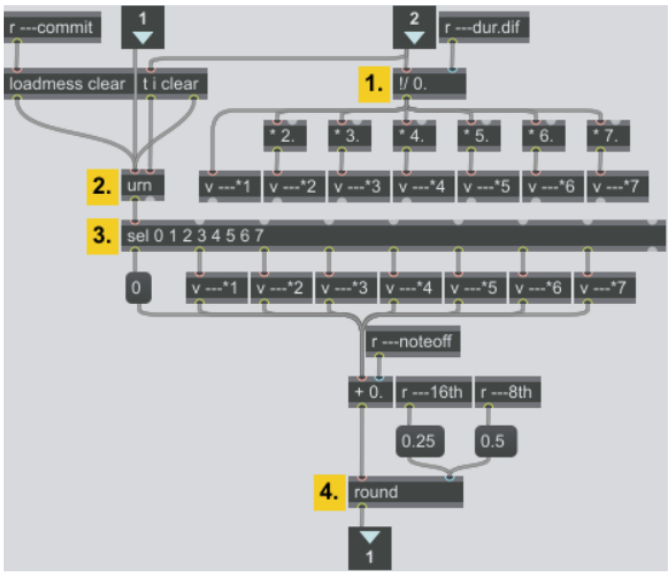


Figure 61. [p passnote.on].

1. The maximum potential number of passing notes to be added in between the current and next note triggered out of the "2" (1 and 1-2 notes), "4" (1-4 notes) and "8" (1-8 notes) messages attached to [sel 0 1 2 3] at (11) in [p nextnote.calc] enters [p passnote.on] through [inlet] 2, where it is immediately used to divide up the time difference between the note-off and note-on times of the current and next note. This value is then multiplied by a range of values from 1 (no multiplication)-7, the results of which are each stored in individual [value] objects in order to be recalled later in the calculation process.
2. After passing through the left outlet of [t i clear] the maximum number of passing notes to be added is also directed into the right inlet of the [urn] object. [urn] is similar to [random] in that it generates a random integer within the range set by its arguments upon receiving a bang but, unlike [random], it never generates duplicate values (the same number twice). This means that once one of the available note-on times in the gap between the two notes on either side of the generated passing notes has been occupied the same note-on time cannot be assigned to another newly generated passing note.

   Here, [urn] is triggered by the bang arriving at the patcher's left [inlet] from [r ---next.passnote] for the generation of each new passing note. The "clear" message, denoted as the second element in the [trigger] object's arguments through which the integer value sent into the right inlet of [urn], is outputted from the right outlet of [t i clear] and sent into the left inlet of [urn] before the maximum number of passing notes to be added reaches the right inlet following its arrival inside the patcher via [inlet] 2. "clear" resets [urn] when all the possible integers within its arguments have been generated so that it can begin outputting numbers once again upon receiving a bang. Because "clear" is outputted from the right outlet of [trigger] and enters [urn] before the integer value denoting the maximum number of passing notes, if [urn] receives the same integer value to set its arguments twice or more in a row when generating passing notes for different gaps between notes that appear during the transformation process, it is reset and is able to continue outputting note-on times for the passing notes in a new gap even though that gap is to be populated by the same number of passing notes as the previous gap.

Note that a bug that is seemingly inherent to the [urn] object can sometimes cause issues with all passing notes between two notes being generated with the same note-on time and, therefore, being stacked on top of each other. Restarting Max and Ableton Live will ordinarily solve this issue but, if not, simply replacing the [urn] object with a new version should fix the problem.

3. The value outputted by [urn] is directed into a [select] object with arguments ranging from 0-7 (eight being the maximum number of passing notes that can be added between any two notes in the clip). Each outlet of the [select] object triggers a different multiplication of the value resulting from the calculation at (1), which divides the time difference between notes by the maximum number of passing notes to be added, therefore providing sufficient unique note-on times for each of the passing notes if the maximum number able to be generated is generated. Once triggered from one of the [value] objects or the "0" message the note-on time is added to the note-off time of the current note, which arrives at the right inlet of [+ 0.] via [r ---noteoff], thus providing a rough final note-on time for the current passing note. For example, if it is calculated that only two passing notes should be added between the current and next note, the [urn] object receives the argument "2" from the second [inlet] of the patcher and, therefore, is only able to generate a value of either 0 or 1. If a 0 is released from [urn], a bang is released from the first outlet of the [select] object that triggers the "0" message, which is then added to the note-off time; meaning that the eventual note-on

time of the current passing note will fall roughly around the same time as the note-off time of the current note. If [urn] generates a 1 it triggers the original division value of the time difference between the current and next note as outputted by [!/ 0.] at (1) and stored in [v ---*1], which is added to the note-off time of the current note meaning that the note-on time for the passing note would fall roughly in the middle of the time difference between the two notes. Likewise, if there were to be eight passing notes added and [urn] generated a value of "3", the resulting note-on time would also fall roughly in the middle of the gap between the current and next note.

4. In order to ensure the passing notes generated during the transformation process are in time with the rest of the notes in the clip, the rough note-on times outputted by [+ 0.] are passed through a [round] object, the purpose of which is to round any incoming values either up or down to the nearest numerical diminution informed by its arguments. The factor by which the rough note-on times are rounded up or down in order to quantize them is subject to the time difference duration between the current note-off time and the next note-on time, which is ascertained at (10) in [p nextnote.calc] and causes a bang to be sent via [s ---16th] or [s ---8th] at (11) and trigger either the "0.25 or "0.5" message respectively. The final quantized note-on time for the passing note is then outputted from [p passnote.on]. ***Refer to 9.4.12 [p nextnote.calc], (15).***

9.4.18 [p pitch.store]

Main patcher → [p joy-lam] → [p joy-lam.transform] → [p pitch] → [p pitch.store]

Once the calculations inside [p duration] have been completed for the current note the calculation to determine the pitch of that note (and of any passing notes that may have been generated during the [p duration] calculations) can take place. Again, [p pitch.store] is very similar to both [p duration.store] and [p velocity.store], excepting three sections of programming logic responsible for storing and outputting the original and transformed pitch values of the note preceding the one currently being processed, as well as ascertaining and sending via [s ---low] and [s ---high] into [p range.sfgd] the highest and lowest note values present in the clip when the transformation is first triggered.

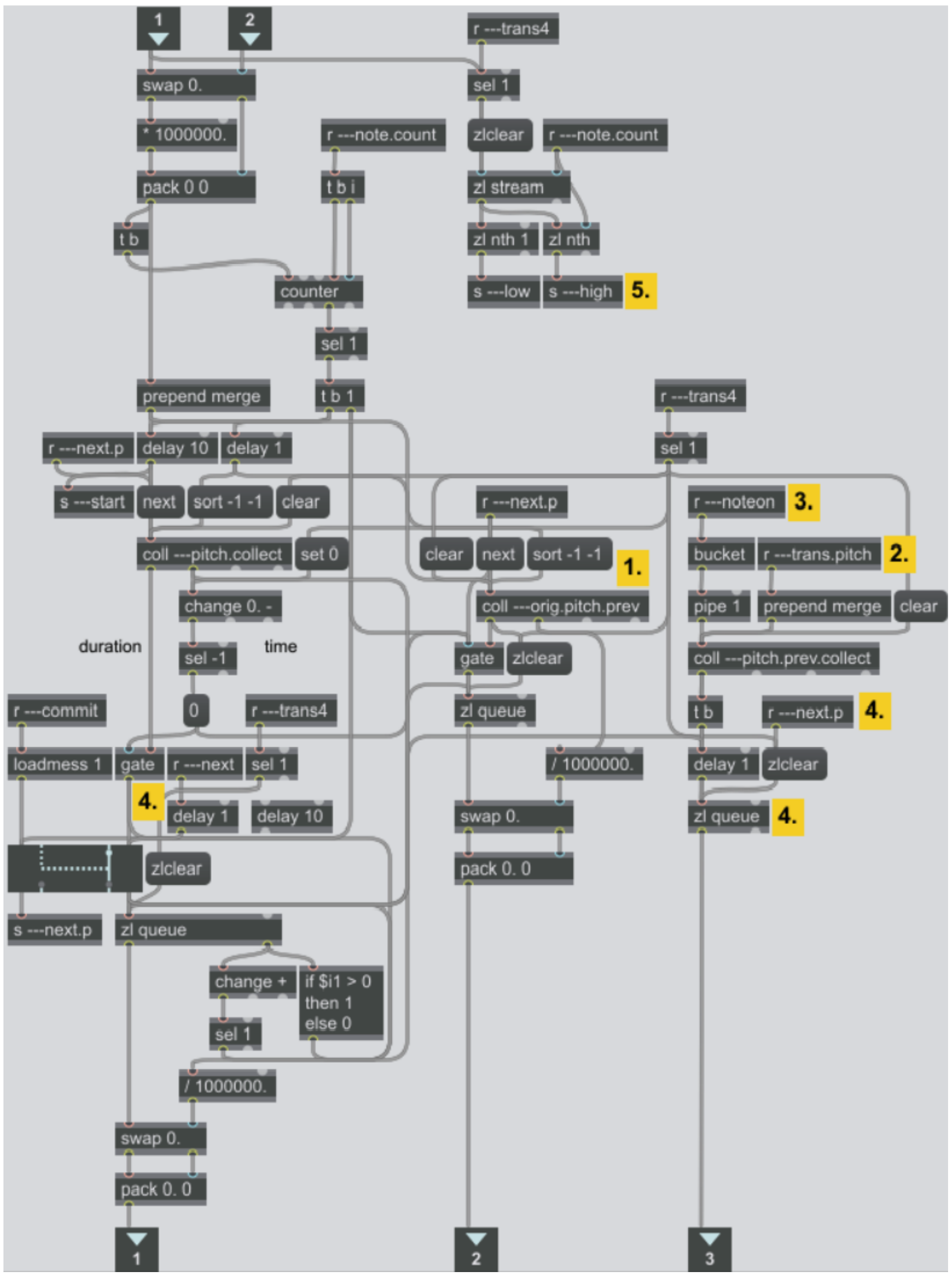


Figure 62. [p pitch.store].

1. The process of storing and recalling the original pitch of the note that precedes the current one from [coll ---orig.pitch.prev] is the same as that of storing and recalling the pitch of the current note from [coll ---pitch.collect] (as well as that of the duration and velocity of the current note from [p dur.collect] and [p vel.collect] inside [p duration.store] and [p velocity.store] respectively), aside from the fact that the "next" message connected to the input of the [coll] is only triggered once the transformation of the note(s) associated with the first note-on time in the clip have been processed and not as soon as all the pitch information from inside the clip has been stored and

organised in chronological order. This means that every time the next pitch (or group of pitches with the same note-on time) to be transformed is released from [coll ---pitch.collect] after the "next" message is triggered by a bang received via [r ---next.p] the original pitch of the preceding note(s) is released from [coll ---orig.pitch.prev]

2. Once the transformation process for the current note is complete, the new pitch of that note, along with the note-on time, is received from [p joy-lam.compile] via [r ---trans.pitch] and inputted into [coll ---pitch.prev.collect] (which is cleared of any previously-stored data from the last transformation to occur upon triggering the process via the "joy-lament" toggle on the TouchOSC GUI) - after passing through [prepend merge] to ensure all pitch values with the same note-on time are stored alongside each other within the same index (note-on time) as opposed to replacing each other upon arrival.
3. The note-on time of the current note is received from [p velocity] via [r ---noteon] and inputted into a [bucket], which outputs the last received value upon the arrival of a new one. The note-on time value outputted from [bucket] (that of the previous note(s) at the time of the transformation of the current note) is delayed by 1 ms by [pipe 1] before entering [coll ---pitch.prev.collect] and outputting all the elements (pitch values) associated with the corresponding note-on time index.
4. Once the transformed pitch value(s) of the previous note(s) have been recalled from the [coll] they are stored in a [zl queue], which is cleared by the bang received from [r ---next.p] before the arrival of any newly transformed pitch values in order to avoid errors in the calculation process resulting from there being fewer pitch values associated with one note-on time than there were with the note-on time that preceded it. The first value received by [zl queue] is immediately recalled after the output of [coll] also triggers a bang from [t b], which is delayed by 1 ms before entering [zl queue] and causes the output of the pitch value. The sequential output of the next previously transformed pitch is then triggered by the bang received from [p joy-lam.compile] via [r ---next] upon completion of the transformation of the current note.
5. At the very top and to the right of the patcher can be seen a sequence of objects responsible for ascertaining and outputting via [s ---low] and [s ---high] the highest and lowest pitch values present in the clip. When the pitch values first enter the

patcher via [inlet] 1 they enter [zl stream], which is cleared of any previous data by the "zlclear" message when the transformation is first triggered, and which serves to output a list of collected values once the total number of individual values denoted in its arguments has been received at its left inlet. Here this value is set by the number of notes originally present in the clip at the time the transformation is first triggered, which is received from [p getnotes] via [r ---note.count] and sent into the right inlet of [zl stream]. The two [zl nth] objects that follow, the argument of the first of which is 1 and the second of which is the number of notes in the clip arriving at [r ---note.count], act by outputting the value in a list which appears in the position within that list defined by their arguments; here being the pitch values of the first and last notes. Because the "get_selected_notes" function inside [p getnotes] retrieves notes from the clip in Ableton Live in ascending order of pitch, the first and last notes in the list collected and outputted by [zl stream] represent the highest and lowest pitch values present in the clip.

9.4.19 [p pitch]

Main patcher → [p joy-lam] → [p joy-lam.transform] → [p pitch]

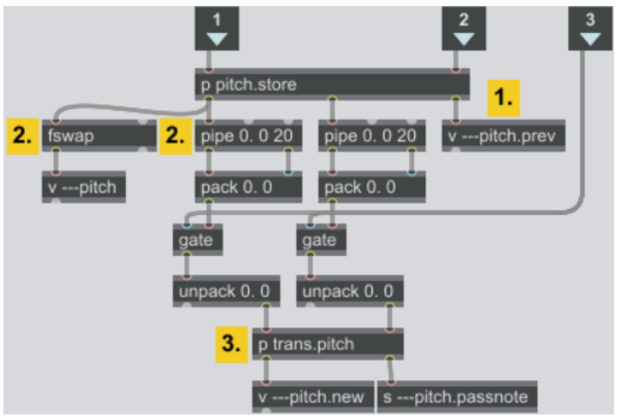


Figure 63. [p pitch].

1. The transformed pitch of the previous note is outputted from the right outlet of [p pitch.store] and stored within [v ---pitch.prev], which is used to recall the value during the calculation processes inside [p trans.pitch], [p up-down] and [p joy-lam.compile].
2. The pitch of the current note and the original pitch of the previous note are outputted in the form of a "note-on time; pitch value" list from the first and second outlets of [p pitch.store] respectively, before being delayed by 20 ms when passing through [pipe 0.

0 20]-[pack 0. 0] and then entering a [gate] - the state of which is subject to the output from [p gate-ctrl] inside [p joy-lam.transform] arriving through the third inlet of the patcher (see **9.4.6 [p gate-ctrl]**). The order of the "note-on; pitch" list for the current note is also flipped by [fswap] and the original pitch value stored inside [v ---pitch] to be recalled inside [p joy-lam.compile] if necessary.

3. Once the "note-on; pitch" lists for the current and previous (original) notes pass through their respective [gate]s, they are unpacked and the pitch values alone are directed into [p trans.pitch], where the pitch transformation/generation calculations for the current note and any passing notes to be added take place. The resulting pitch values of these calculations are outputted from the left and right outlets of the patcher respectively and directed into [v ---pitch.new] and [s ---pitch.passnote] to be used inside [p joy-lam.compile].

9.4.20 [p trans.pitch]

Main patcher → [p joy-lam] → [p joy-lam.transform] → [p pitch] → [p trans.pitch]

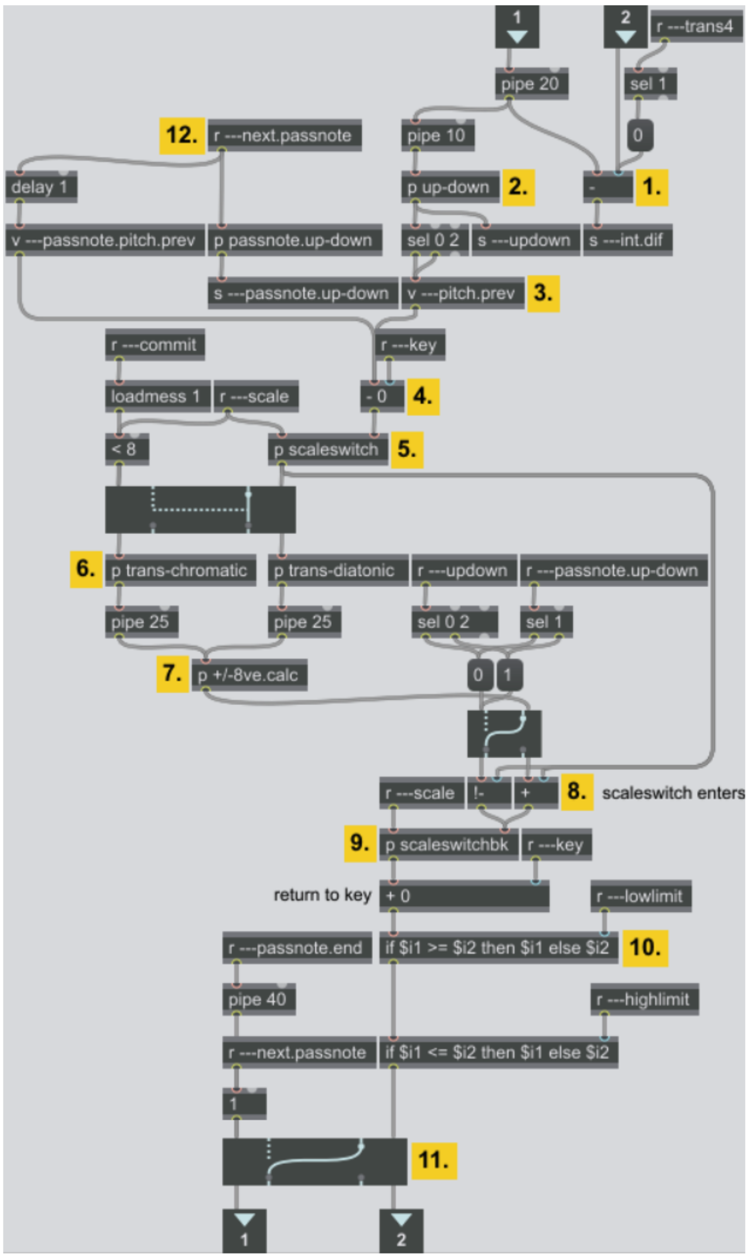


Figure 64. [p trans.pitch].

1. The original pitch of the preceding note to the current one arrives at [inlet] 2 of [p trans.pitch] and enters the right inlet of [-]. After being delayed by 20 ms when passing through [pipe 20] the pitch of the current note, which arrives at the patcher via [inlet] 1, is directed into the left inlet of [-] and the value of the original pitch of the previous note is subtracted from it. The sum of this calculation is sent via [s ---int.dif] into [p +/-8ve.calc]. When the transformation is first triggered by the "joy-lament" toggle on the TouchOSC GUI a "0" is directed into the right inlet of [-] in order to

clear any original previous pitch values that may still be registered by the object in its arguments.

2. After the integer difference between the original pitch of the current and previous notes has been ascertained the original pitch of the current note enters [p up-down], after being delayed by a further 10 ms, in order to decide whether the current note will move up or down in pitch from the previous note or remain unchanged from its original pitch (regardless of whether that is above, below or the same as the previous pitch.) ***Refer to 9.4.21 [p up-down], then 9.4.22 [p range.sfgd].***
3. Once [p up-down] has calculated whether or not the the current note will move up or down in pitch from the previous note or remain unchanged the result is sent via [s ---updown] to (8) and [p joy.lam] before entering [sel 0 2], which, only if [p up-down] has calculated that the current note should move up or down in pitch from the previous, sends a bang to [v ---pitch.prev] and releases the transformed pitch of the previous note.
4. In order for the transposition either up or down in relation to the previous note to occur, the incoming pitch values first need to be normalised so that the root note of the chosen scale becomes C. Removing the variation that exists in relation to the exact placement of the semitone intervals across the entire MIDI note number range between notes within a scale with different root notes means that all scales, regardless of their original key, can be subjected to the same transformation process. This happens by subtracting the semitone interval difference between C and the root note of the chosen key signature/scale of the incoming notes. This interval difference is received via [r ---key] from [p keyselect] inside [p joy-lam], which works in exactly the same way as [p keyselect] in *ScreenPlay-CTRL*.
5. The newly normalised pitch value of the previous note then enters the right inlet of [p scaleswitch], into which an integer denoting the specific scale of the current clip is also sent through its left inlet via [r ---scale]. Inside the pitch value is converted again into an integer that is usable in a universal calculation process which serves to add/subtract intervals from any given pitch in all the key signatures/scales made available to the user(s) by *ScreenPlay*. ***Refer to 9.4.23 [p scaleswitch].***
6. Once the pitch conversion process has taken place inside [p scaleswitch] the outputted integer value passes through a [gswitch2] object and into either [p trans-chromatic] or [p trans-diatonic], depending on the currently selected key signature/scale. A diatonic

scale received via [r ---scale] has an integer identifier of value 0-7 and, therefore, triggers a "1" from [< 8] and directs the incoming values from [p scaleswitch] through the right outlet of [gswitch2] and into [p trans-diatonic]. A chromatic scale has an integer identifier of 8 and causes the output of "0" from [< 8], which, in turn, directs the incoming values from [p scaleswitch] at the right inlet of [gswitch2] out of the left outlet from where it passes into [p trans-chromatic]. ***Refer to 9.4.24 [p trans-diatonic]/[p trans-chromatic].***

7. Once the interval to be added/subtracted to/from the pitch of the previous note has been calculated inside either [p trans-chromatic] or [p trans-diatonic] the result is delayed by 25 ms when passing through [pipe 25] before entering [p +/-8ve.calc], which adds an octave to the interval outputted from either [p trans-chromatic] or [p trans-diatonic] if the interval difference between the original pitch of the current note and the original pitch of the previous note is greater than an octave in order to retain some of the character of the source material inside the clip. ***Refer to 9.4.25 [p +/-8ve.calc].***
8. After the final pitch interval to be added/subtracted to/from the pitch of the previous note in order to transform the pitch of the current note has been outputted by [p +/-8ve.calc] it enters the right inlet of a [gswitch2] object before being directed out of one of two outlets, dependent on the value received via either [r ---updown] or [r ---passnote.up-down] for a passing note. The path down which the pitch interval value travels dictates whether the new pitch should be higher or lower in register than the one preceding it, after which it is either added to or subtracted from the pitch of the previous note outputted from [p scaleswitch] at (5).
9. The resulting value is then converted back into the corresponding MIDI note number with the chosen scale with a root of C when passing through [p scaleswitchbk], which works in exactly the opposite way to [p scaleswitch], before being returned to the correct key by adding the value from [r ---key] that was subtracted at (4).
10. The newly generated pitch value then passes through a sequence of objects very similar to those contained within [p range.sfgd]. The role of the value from [r ---lowlimit]/[r ---highlimit] and the attached [if] objects is exactly the same as in [r range.sfgd] - to be compared against the pitch value arriving at the left inlet of the two [if] objects - only that, in this instance, the two [if] objects will release the pitch value arriving at their left inlets if it is greater than or equal to/less than or equal to the value

arriving at their right inlets; otherwise releasing the value from their right inlets in its place - i.e. the lowest/highest root note pitch values possible transformation range denoted by [p setrange] inside [p range.sfgd] in [p up-down]. This acts as a secondary safeguard to the process inside [p range.sfgd] to ensure no pitch values are generated outside the 0-127 range of MIDI note numbers.

11. After passing through a [gswitch2] object the transformed pitch value is finally sent out from the patcher via one of two [outlet]s, depending on whether it is that of a note originally inside the clip at the time the transformation was triggered or a newly generated passing note. If a bang is received from [p joy-lam.compile] via [r ---next.passnote] it triggers a 1 from [1], thus sending the value entering into the right inlet of [gswitch2] out of its right outlet and out of the patcher via [outlet] 2. [r ---passnote.end] receives a 0 from [p joy-lam.compile] once all passing notes between the current and next note have been added to the clip and resets the state of [gswitch2] so that the transformed pitch of notes from within the clip are sent out of [outlet] 1.
12. Again, if [r ---next.passnote] receives a bang it triggers the calculation inside [p passnote.up-down], which works in exactly the same way as [p up-down] except that the generated pitch values can only move up or down in relation to the pitch value preceding them. There is no chance of there being no change in the original pitch of the current note seeing as all passing notes are newly generated and do not exist prior to the calculation process. All passing notes are generated in relation to the pitch of the previous note in the clip regardless of the number of passing notes to be added, which means that there is no requirement either to account for repeating the pitch of the previous passing note due to the fact that, for both "joy" and "lament", the restrictions on the probabilities of certain intervals being generated means that there is an inherent increased likelihood of repeated passing notes at both ends of the "joy-lament" scale.

    The pitch of the previous note is released inside [p joy-lam.compile] from [v ---pitch]/[v ---pitch.new] (depending on whether or not a transformation of the note preceding the passing notes has occurred) and stored in [v ---passnote.pitch.prev] at (4). The pitch value released from [v ---passnote.pitch.prev] in [p transpitch] is sent into [- 0] at (4), from which point on the transformation process is exactly the same as previously described. ***Refer to 9.4.26 [p joy-lam.compile].***

9.4.21 [p up-down]

Main patcher → [p joy-lam] → [p joy-lam.transform] → [p pitch] → [p trans.pitch] → [p up-down]

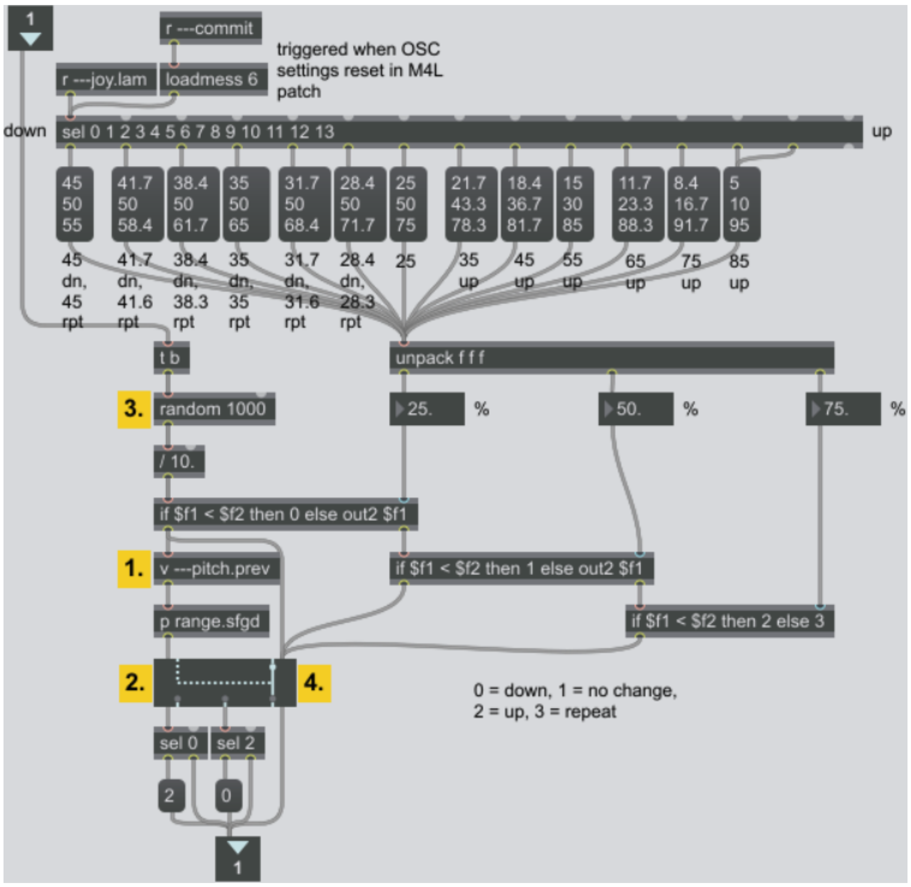


Figure 65. [p up-down].

1. When the original pitch of the current note arrives inside [p up-down] it causes a bang to be outputted from [t b], which first causes the output of the transformed pitch of the previous note from [v ---pitch.prev]. This integer then enters [p range.sfgd], which acts to ensure all pitch values generated by [p trans.pitch] stay within an octave either side of the lowest and highest notes originally present in the clip when the transformation was first triggered by forcing the next note to be generated to move up or down in pitch from the previous note if the previous note is too low/high respectively. ***Refer to 9.4.22 [p range.sfgd].***
2. The values outputted by [p range.sfgd] are used to direct the output of the [if] objects at (3) through one of three outlets on a [gswitch2] object.
3. As with the other probability calculations, the bang from [t b] also causes the output of a random value, which is directed into a series of [if] objects with varying arguments

depending on the position of the "joy-lament" slider in order to determine the outcome of the calculation. Here, there is a 45% chance at "lament" that the pitch of the current note will move down in register from the previous note, a 45% chance that the pitch of the previous note will be repeated and a 5% chance each that there will be no change from the original pitch of the note or that it will move up in pitch from the previous note. At "joy" there is an 85% chance that the current note will move up in pitch from the previous note and a 5% chance that it will move down, there will be no change, or the previous pitch will be repeated. There is an equal chance (25%) of an upward movement, downward movement, no change or repeated pitch in the middle of the two extremes.

4. Unlike the other probability calculations the value outputted by the [if] objects (0 = pitch down, 1 = no change, 2 = pitch up, 3 = repeat pitch) then enters the right inlet of the [gswitch2] at (2). If the value outputted by [p range.sfgd] is a 0 - pitch of previous note too low for next note to move down in pitch - the value outputted by the [if] objects is directed through the left outlet of [gswitch2] and into [sel 0]. If the value arriving at [sel 0] from [if] is 0 (i.e. pitch to go down) a "2" is released, thus forcing the pitch instead to go up in register from the previous note given that it cannot possibly go any lower. Otherwise, if the value from [if] is 1, 2 or 3 (i.e. no change in pitch, pitch up or repeat pitch), it passes straight through the [select] object and out of its right outlet with the original result of the probability calculation being retained and released from the patcher. If the value outputted by [p range.sfgd] is 1 - pitch of previous note too high for next note to move up in pitch - the value outputted from the [if] objects is directed through the second outlet of [gswitch2] and into [sel 2]. If the value from [if] is 2 (i.e. pitch to go up) [sel 2] triggers a "0", forcing the pitch instead to go down in register from the previous note given that it cannot go any higher. If the value from [if] is 0, 1 or 3 (i.e. pitch down, no change or repeat pitch) it passes straight through the [select] object and out of its right outlet with the original result of the probability calculation being retained and released from the patcher. If the value outputted by [p range.sfgd] is 2 the value resulting from the probability calculation outputted by the [if] objects passes through the right outlet of [gswitch2] and straight out of the patcher unchanged.

It should be noted that, as the calculation process for pitch transformation is based upon the intervals between the current and previous notes as opposed to simply treating each note in isolation and moving it either up or down in pitch, it is possible for the pitch of the current note to move down in register despite the result of the calculation inside [p up-down] determining that it should move up in register from the previous pitch or vice versa. ***Refer to 9.4.20 [p trans.pitch], (3).***

9.4.22 [p range.sfgd]

Main patcher → [p joy-lam] → [p joy-lam.transform] → [p pitch] → [p trans.pitch] → [p up-down] → [p range.sfgd]

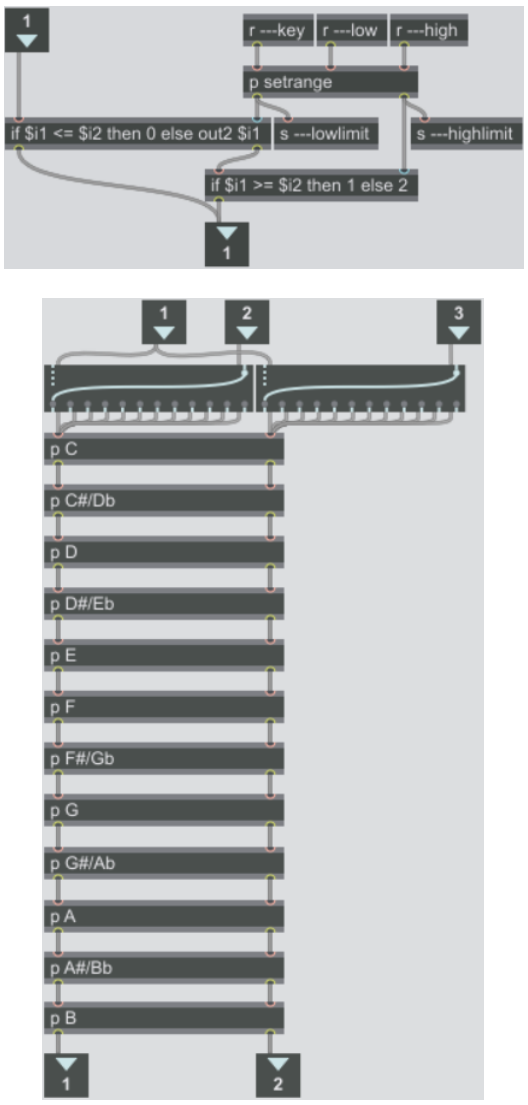


Figure 66. [p range.sfgd] and [p setrange].

When the transformed pitch of the previous note first arrives inside the patcher it enters the left inlet of an [if] object and is compared against the value entering the right inlet of the object from [p setrange], which establishes the upper and lower limits of the range within which note transformations are able to take place for any given clip by outputting from its right and left outlets respectively the root notes of the chosen key signature/scale from the octaves above and below the highest and lowest notes present in the clip when the transformation is first triggered. The highest and lowest pitch values arrive inside [p setrange] from [r ---high] and [r ---low] and each enters its own [gwtich2] object, the outlets of which the two values pass through are defined by the value arriving at [r ---key] from [p joy-lam]. Connected to the twelve outlets of each of the [gswitch2] objects via their left and right inlets are twelve individual patchers for each key. Each patcher contains a series of [if] objects which ascertain within which octave from the 0-127 MIDI note number range the highest and lowest pitch values from the clip fall and then output the root note of the key from the octave above.

As well as entering the right inlets of the two [if] objects inside [p range.sfgd] upon being released from [p setrange] the upper and lower limits of the transformation range are also sent via [s ---highlimit] and [s ---lowlimit] to (10) inside [p trans.pitch]. The first of the two [if] objects is responsible for calculating if the incoming pitch value is too low for the following note to move down in pitch from that of the note preceding it without the possibility of falling outside the lower limit of the range defined by [p setrange]. If the transformed pitch of the previous note arriving at the left inlet of [if] is less than or equal to the value at the right inlet the object outputs a 0 from its left outlet, otherwise the transformed pitch of the previous note passes out of the right outlet unaffected and into the second [if] object in the sequence.

The second [if] object calculates whether or not the incoming pitch value of the previous note is too high for the following note to move up in pitch from its own value. If the value arriving at the left inlet of the object is greater than or equal to the value set at its right inlet a 1 is released, otherwise a 2 is released. ***Refer to 9.4.21 [p up-down], (2).***

9.4.23 [p scaleswitch]

Main patcher → [p joy-lam] → [p joy-lam.transform] → [p pitch] → [p trans.pitch] → [p scaleswitch]

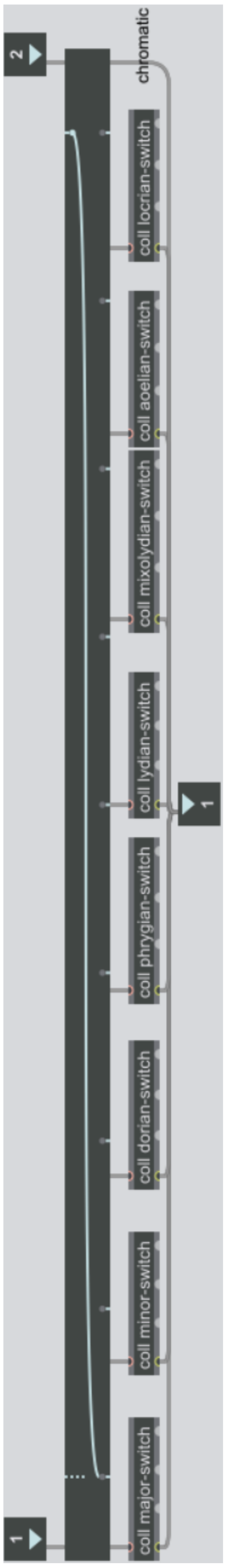


Figure 67. [p scaleswitch].

[p scaleswitch] serves to normalise the intervals between all the notes of any scale made available by *ScreenPlay* in order to simplify the process of transposing notes inside the clip whilst ensuring they remain within the chosen key/signature scale. Because the notes of a chromatic scale remain the same regardless of the root note, no normalisation process is necessary and, if the [gswitch2] object receives the integer value that denotes a chromatic scale at its left inlet via the left inlet of [p scaleswitch] to which [r ---scale] is attached, the incoming normalised pitch values arriving from the right inlet of [p scaleswitch] and sent into the right inlet of [gswitch2] pass out of [gswitch2] via its rightmost outlet and straight back out of the patcher. For all the other seven-tone diatonic scales available to users of *ScreenPlay*, the incoming pitch value enters the appropriate [coll] object after passing through [gswitch2], inside which the index values correspond to every MIDI note number of the given scale with a root note of C, with each having assigned to it a single element. The value of these elements increases from 0 for the lowest note of the scale up to 74 for the highest note - given that the seven-tone scales all contain a total of 75 notes. When the pitch arrives at the inlet of a [coll] it causes the element integer associated with that specific pitch index to be outputted from the left outlet of the object and then out of the patcher. Converting all possible notes within each scale to the same sequential list of integers means that, regardless of the scale, a jump from the root to the seventh, for example, can be achieved simply by adding a 6 to the value corresponding to the pitch of the root note, rather than, for example, having to add eleven semitones for a major scale and only ten for a minor scale. ***Refer to 9.4.20 [p trans.pitch], (6).***

9.4.24 [p trans-diatonic]/[p trans-chromatic]

Main patcher → [p joy-lam] → [p joy-lam.transform] → [p pitch] → [p trans.pitch] → [p trans-diatonic]/[p trans-chromatic]

The probability calculations taking place in both these patchers follow the same blueprint as all other probability calculations in the transformative patch, albeit in an extended form so as to include seven and twelve possible outcomes respectively for [p trans-diatonic] and [p trans-chromatic], given the number of intervals between notes within the diatonic and chromatic scales afforded by *ScreenPlay*.

At the "lament" extreme of the scale inside [p trans-diatonic], there is a 10% chance of the calculation resulting in an interval of a seventh above/below the previous note and a 45%

chance for both a second and a third. At "joy" there is a 10% chance of an octave and a 45% chance of both a fourth and a fifth. With the "joy-lament" slider on the GUI positioned centrally there is an equal chance of all available intervals being generated.

With [p trans-chromatic], at "lament" there is a 5% chance of both a major sixth and a minor seventh, while a minor second, major second, minor third and major third each have a 22.5% chance. At "joy" there is a 5% chance of both an octave and major seventh, while each of minor sixth, perfect fifth, diminished fifth/augmented fourth and perfect fourth have a 22.5% chance. Again, with the slider positioned centrally, there is an equal chance (8.3%) of any interval within an octave being generated. ***Refer to 9.4.20 [p trans.pitch], (7).***

9.4.25 [p +/-8ve.calc]

Main patcher → [p joy-lam] → [p joy-lam.transform] → [p pitch] → [p trans.pitch] → [p +/-8ve.calc]

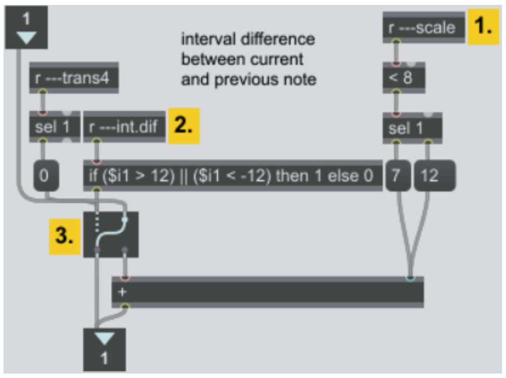

1
interval difference between current and previous note
r ---scale
1.
r ---trans4
< 8
sel 1
r ---int.dif
2.
sel 1
0
if ($i1 > 12) || ($i1 < -12) then 1 else 0
7
12
3.
+
1

Figure 68. [p +/-8ve.calc]

1. An integer corresponding to the currently selected scale on the TouchOSC GUI is received by [r ---scale] (0-7 = Major, Minor, Dorian, Phrygian, Lydian, Mixolydian, Aeolian, Locrian; 8 = chromatic) and is directed into [< 8]. If the value of the integer is less than 8 (i.e. 7-tone scale), the object outputs a 1, which, in turn, triggers the "7" message connected to the left outlet of [sel 1] and sends it into the right inlet of [+]. If a chromatic scale is selected [< 8] outputs a 0, triggering instead the "12" message connected to the right outlet of [sel 1].
2. The interval difference between the original pitch of the current and previous notes ascertained at (1) in [p trans.pitch] is received by [r ---int.dif] and directed into an [if] object, which determines whether or not the difference is greater than an octave and sends a 1 into the left inlet of a [gswitch2] if it is and a 0 if it is not.

3. The interval integer to be added/subtracted to the transformed pitch of the previous note, which arrives at the patcher via [inlet] 1, is directed into the the right inlet of [gswitch2] and then either passes straight back out of the patcher if it has been determined that the interval difference between the original pitch of the previous and current notes is less than or equal to an octave, or, if the interval difference between the original pitch of the previous and current notes is greater than an octave, into the left inlet of [+] to be added to either 7 or 12 - depending on whether the currently selected scale is diatonic or chromatic - before being outputted from the patcher. ***Refer to 9.4.20 [p trans.pitch], (8).***

9.4.26 [p joy-lam.compile]

Main patcher → [p joy-lam] → [p joy-lam.compile]

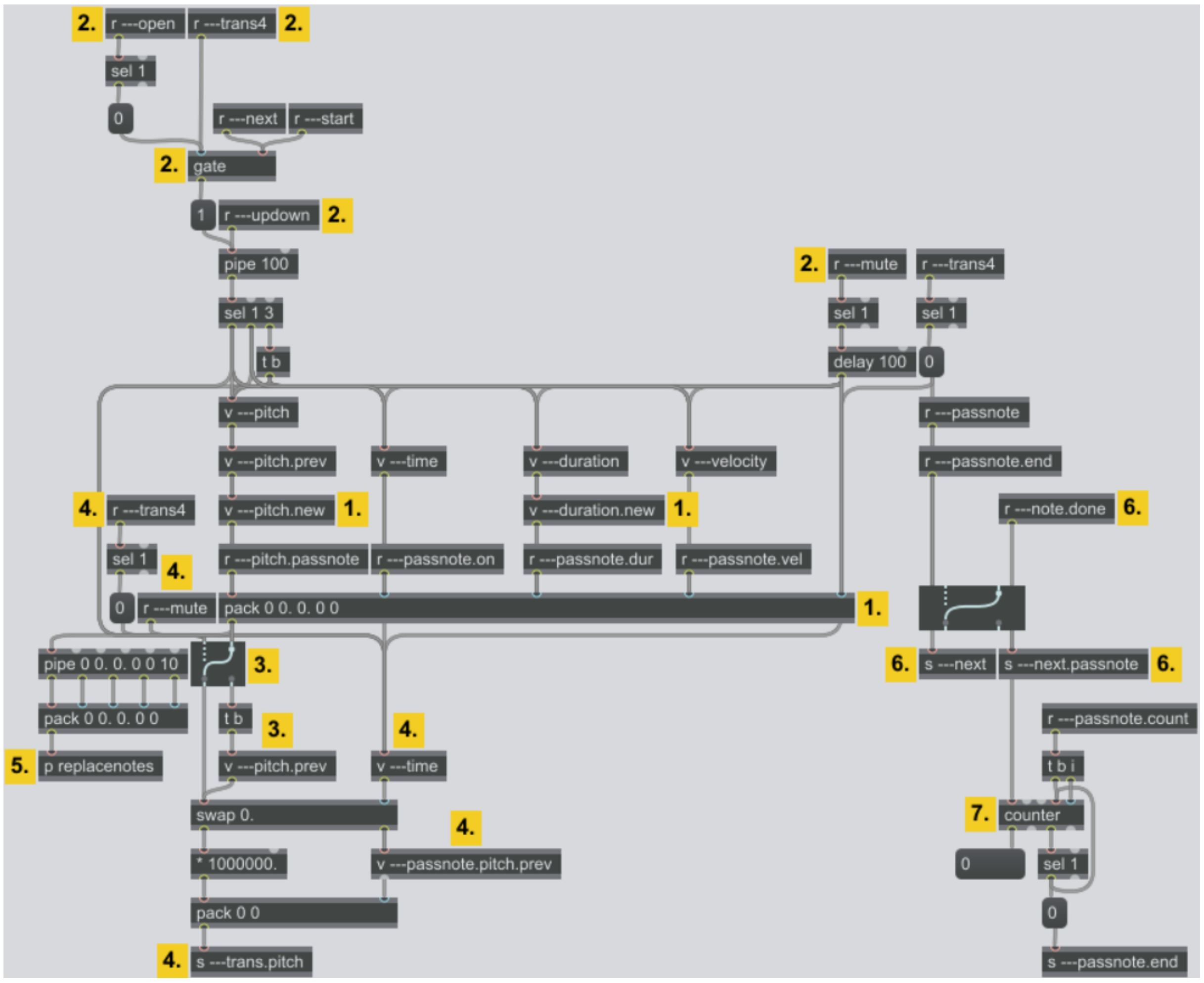


Figure 69. [p joy-lam.compile].

It is inside [p joy-lam.compile] that the velocity, duration, pitch and note-on time of every note to be added to the clip in Ableton Live is compiled into a list of that order (with the addition at the end of mute value), before being sent to [p replacenotes] where the newly formed list for each note is inserted into the clip one at a time.

1. Once the transformation process of duration and pitch is complete the new values are stored inside [v ---duration.new] and [v ---pitch.new] after being outputted from [p trans.dur] and [p trans.pitch] respectively. As the velocity and note-on times of the notes originally in the clip are not affected by the transformation process these values are recalled from [v ---velocity] and [v ---time], whilst the original pitch and duration values are also available to be recalled from [v ---pitch] and [v ---duration] respectively in the event that it is calculated that there should be no change to either of these parameters of any particular note during the transformation process. Likewise, the transformed pitch of the previous note can be recalled from [v ---pitch.prev] in the event that it is calculated that the pitch should be repeated. Because passing notes are newly generated during the transformation the velocity, duration, note-on time and pitch arrive inside the patcher instead via [send]/[receive] objects and are automatically packed into a list of "pitch; note-on; duration; velocity; mute value".
2. With the exception of the first note(s - if there is more than one with the same note-on time) in the clip the decision to use the original pitch/duration values of the notes inside the clip when compiling the final list of note values, or to repeat the pitch of the previous note, is subject to the output of both [r ---updown] and [r ---mute]. After being delayed by 100 ms by [pipe 100], the value arriving at [r ---updown] enters a [sel 1 3] object. The incoming value ranges from 0-3, with 0 and 2 indicating that a transformation (either up or down in pitch from the previous note) of the original note has occurred, 1 indicating that no change has been made to the pitch of the current note and 3 indicating that the pitch of the previous note should be repeated. If the value is either a 0 or a 2 the transformed pitch inside [v ---pitch.new] is outputted by a bang arriving from [t b] connected to the right outlet of [sel 1 3]. A value of 1 causes a bang to be released from the left outlet of [sel 1 3] which triggers the output of the original pitch value stored inside [v ---pitch]. A value of 3 causes a bang to be released from the middle outlet, which triggers the transformed pitch of the previous note stored inside [v ---pitch.prev].

   In the case of duration, the value stored inside [v ---duration.new] is triggered regardless of the value outputted by [r ---updown]. This is because, if the calculation inside [p trans.dur] determines that the duration of the current note should remain unchanged, the original duration value simply passes through unaltered and into [v ---duration.new] as opposed to the calculation process being halted as is the case inside [p trans.pitch]. Likewise, the original velocity and note-on time for each note are also outputted from [v ---velocity] and [v ---time] regardless of the value from [r ---updown].

The presence of [v ---duration] is still required in the case that the current note is muted as the original duration value of the current note is blocked off inside [p duration] before being able to enter [p trans.dur] and thus populate [v ---duration.new]. If it has been calculated inside [p nextnote.calc] that the current note should be muted a value of 1 arrives at [r ---mute] (as opposed to 0 if the note is to remain unmuted). The 1 first enters the right inlet of the [pack 0 0. 0. 0 0] object at (1) before causing a bang to be released from [sel 1], which, after being delayed by 100 ms, triggers the original values for velocity, duration, note-on time and pitch from their respective [value] objects.

3. Before the final note list is outputted from the [pack] object the pitch value alone - whether from [v ---pitch], [v ---pitch.prev] or [v ---pitch.new], and not including those outputted by [r ---pitch.passnote] - enters the right inlet of a [gswitch2] object without passing through the [pack] object. The outlet through which the pitch value leaves [gswitch2] is determined by the mute value of the current note received by [r ---mute], while its routing is also reset to the left outlet whenever a new transformation is triggered by the [sel 1]-"0" connected to [r ---trans4]. If the current note is to be muted, [r ---mute] sends a 1 into the left inlet of [gswitch2] and forces the incoming pitch value to be outputted through its right outlet, after which a bang is released from [t b] that triggers the release of the previous pitch from [v ---pitch.prev], which is then sent into the left inlet of [swap 0.]. If the current note is to remain unmuted a 0 is released from [r ---mute] and causes the incoming pitch value to be outputted through the left outlet of [gswitch2] and pass straight into the left inlet of [swap 0.]
4. From here the note-on time that arrives at the right inlet of [swap 0.] after being released from [v ---time] when it is triggered by a bang received either from the [delay 100] connected to [r ---mute]-[sel 1] at (2) or any of the left, centre or right outlets of [sel 1 3] connected to [r ---updown]-[pipe 100] at (2) is swapped with the pitch value arriving at the left inlet; with the pitch value leaving [swap 0.] via the right outlet and the note-on time via the left outlet before being multiplied by 1000000.

The pitch value first enters [v ---passnote.pitch.prev], from where it is accessed inside [p trans.pitch] and [p passnote.up-down] during the process of calculating the pitch of the next passing note to be generated. The two values are then packed into a list of "(note-on time x 1000000); pitch" and sent via [s ---trans.pitch] into [p pitch.store] to be used in the pitch transfromation calculation for the next note.

5. Finally, the list outputted from [pack 0 0. 0. 0 0] at (1) enters [p replacenotes], after being delayed by 10 ms, where it inserted into the MIDI clip inside Ableton Live. ***Refer to 9.4.27 [p replacenotes].***
6. When the bang sent out of [p replacenotes] via [s ---note.done] is received at its corresponding [receive] object, it enters the right inlet of a [gswitch2] and is outputted out of one of two possible outlets depending on whether or not the next note will be a passing note or a note originally found within the clip. For an original note, the bang is sent via [s ---next] to a variety of locations elsewhere in the patch, including [p velocity.store], [p duration.store] and [p pitch.store] in order to trigger the transformation of the next note in the clip. For a passing note, the bang is sent via [s ---next.passnote] into [p trans.pitch] and [p nextnote.calc] in order to generate the next passing note.
7. In the case that the next note is a passing note, the bang coming out of the right outlet of [gswitch2] also enters the leftmost inlet of a [counter], the maximum value of which is set by the value received via [r ---passnote.count] indicating the total number of passing notes to be inserted between the note-off time of the current note and the note-on time of the next. Once the maximum value has been reached [sel 1] releases a bang that resets the [counter] to 0 as well as triggers a "0" message that is sent via [s ---passnote.end] into the left inlet of [gswitch2] at (6), redirecting the bangs received by [r ---note.done] back out of the left outlet and into [s ---next], as well as into the left inlet of the [gswitch2] at (10) inside [p trans.pitch], thus directing the final pitch values out of the left outlet of the patcher instead of the right. ***Refer to 9.4.27 [p replacenotes], (5).***

9.4.27 [p replacenotes]

Main patcher → [p joy-lam] → [p joy-lam.compile] → [p replacenotes]

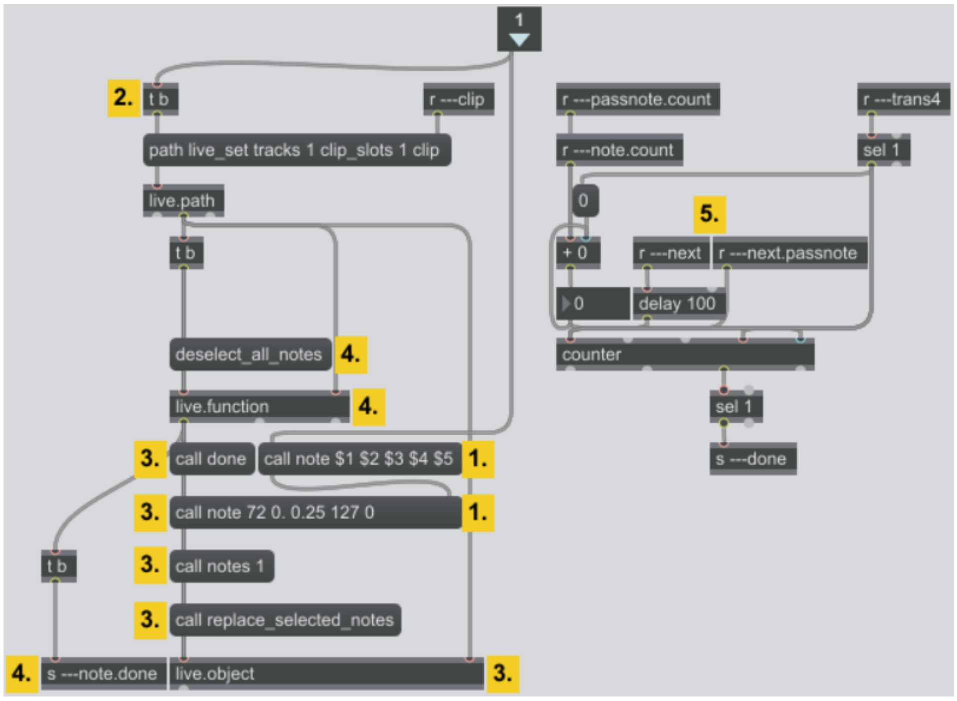


Figure 70. [p replacenotes].

1. Upon entering [p replacenotes] the final note list of the current note first passes through a "call note $1 $2 $3 $4 $5" message, populating the variable elements, before passing into the right inlet of a subsequent message box, therefore populating it without causing any output from the outlet.
2. The incoming note list then causes the release of a bang from [t b], which, in turn, causes a pathway message relating to the currently/last active clip in Ableton Live (received via [r ---clip]) to be released through the outlet of the message box in which it is contained and into the [live.path] object.
3. A number of command functions then follow with the "call replace_selected_notes", "call notes 1", "call note $1 $2 $3 $4 $5" (pitch; note-on time; duration; velocity; mute value), "call done" actions being triggered in that order.
4. The "deselect_all_notes" command function is then triggered, which, upon completion causes a bang to be released from the [t b] object connected to the output of [live.function] that is sent via [s ---note.done] into [p joy-lam.compile].

   The ordering of steps (3) and (4) ensures that, when the very first note in the clip is inputted back into the clip after undergoing the transformation process, all the original notes inside the clip will still be selected from the moment the transformation was initiated and so are deleted from the clip at the end of stage (3). The newly added note is automatically selected inside the clip after being inserted and so, because it is necessary to retain all the new note information that is inserted into the clip, must be sent a command in order to be de-selected and avoid being replaced by the following incoming note when the clip receives the "call replace_selected_notes" series of commands at (3). ***Refer to 9.4.26 [p joy-lam.compile], (6).***

5. The bangs sent out of [s ---next] and [s ---next.passnote] are also received inside [p replacenotes] and directed into the left inlet of a [counter], the maximum value of which this time has been set by the cumulative value of the original number of notes found in the clip at the time the transformation process was initiated - received from [p getnotes] via [r ---note.count] - and the total number of passing notes to be added to the clip during the transformation process - received from [p nextnote.calc] via [r ---passnote.count] every time a new set of passing notes is added between two notes in the clip. Each time a bang is received at the left inlet of the [counter] via [r ---next] or [r ---next.passnote] the current count inside the object increases until eventually catching up with the maximum value, at which point a bang is triggered from [sel 1] that is sent via [s ---done] into [p joy-lam] (main patcher → [p joy-lam]) where it is used to turn the "joy-lament" toggle on the GUI back off, indicating to the user that the transformation process is complete. A second [r ---done] object in [p joy-lam] also receives the bang and opens up a [gate] object that stops any output from the "joy-lament" transformation toggle on the TouchOSC GUI from passing through and interrupting an ongoing transformation.

10. Appendix 4: Example User-Feedback Questionnaire

Which of the following best describes you?

Classical musician	Popular musician	Electronic musician
Non-musician	Gamer	Other (please state)

On a scale of 1-10, what would you rate your level of musical proficiency?

Within which age bracket do you fall?

18-30	30-45	45-60	60-75	75+

Have you any previous experience using ICMSs of any kind?

Yes No

What is your relationship with touchscreen technology?

Everyday/casual	Music creation	Gaming	App developer

On a scale of 1-10, with 1 being the most difficult and 10 being the easiest, how easy/difficult did you find the process of interacting with *ScreenPlay* for the first time?

Would *ScreenPlay* benefit from the inclusion of an instructions page?

Yes No

Please list on the reverse of the questionnaire any further improvements you think could be made to *ScreenPlay* and its GUI.

11. Appendix 5: Accompanying Data CD

Included on the accompanying data CD to the thesis are a number of items, which are listed below:

11.1 ScreenPlay Video Demonstration

This video serves to showcase in an improvisatory, performative manner the functionality of *ScreenPlay* when used in single mode as a studio compositional tool, and how the system can be integrated into the compositional process alongside other hardware such as synthesizers and MIDI controllers. In addition to *ScreenPlay*, which is used to control three individual instruments/parts in the musical arrangement, an Ableton Push is utilised to switch between the different instruments/parts and arm/disarm their corresponding tracks in order to play and record notes via *ScreenPlay*'s GUI, duplicate the clips in the track for *ScreenPlay* part 1, and control the drum track. The MIDI notes played into Ableton Live via *ScreenPlay*'s GUI, along with those generated by *ScreenPlay-GEN* and *ScreenPlay-TRNS4M*, are sent out of Ableton Live and into an Access Virus synthesizer and Korg Volca FM synthesizer for two of the parts, and played back via an electric bass guitar preset for Ableton Live's Sampler instrument for the other.

The level of performance exhibited in the video accurately demonstrates that which can reasonably be expected when interacting with *ScreenPlay* using a standard home wireless router. Improved performance may well be possible with a high speed connection/router, although this has not been tested. Note also that sixteen bars have been cut from the video edit of the demonstration at 5'56" in an attempt to reduce the overall duration of the video and due to the fact that little of consequence happened during this time. The only significant change to the status of the system during this time is the repositioning of the "joy-lament" slider for the bass guitar part.

11.2 ScreenPlay Max for Live MIDI Devices/Ableton Audio Effect Rack

The *ScreenPlay-CTRL*, *ScreenPlay-GEN* and *ScreenPlay-TRNS4M* Max for Live MIDI Devices and the *ScreenPlay-FX* Ableton Audio Effect Rack are all required to run *ScreenPlay*. Detailed setup instructions are provided in Appendix 1, which demonstrate how to use the devices in tandem with TouchOSC and Ableton Live to configure the system for use in both single and multi mode.

11.3 ScreenPlay GUI TouchOSC Layout Files

TouchOSC GUI layout files for *ScreenPlay* supporting MIDI channels 1-4 have been provided for both iPad and Android devices. To make more versions of the GUI layout using different MIDI channels simply make copies of the existing files and alter the assigned MIDI channel of all of the "pads" on the grid-based playing surface in the TouchOSC Editor software (available from http://hexler.net/). Note that the iPad and Android layout files differ only in terms of size/resolution (with the iPad version also providing an additional row of notes/"pads" on the grid-based playing surface), and not in terms of file type. As such, it is possible to use either with any device, regardless of operating system.

11.4 Skeleton Ableton Live Set Files

The provided Ableton Live Set files can be used as the basis from which to configure *ScreenPlay* in both single and multi mode, meaning that it is necessary only to follow the setup instructions provided in Appendix 1 from stage 4/5 onwards for multi/single mode respectively.