Empirical Studies in End-User Computer-Generated Music Composition Systems

Samuel James Hunt

Director of Studies: Prof. Larry Bull

Supervisors: Dr Chris Nash and Dr Tom Mitchell

A thesis submitted in partial fulfilment of the requirements of the University of the West of England for the degree of Doctor of Philosophy

Department of Computer Science and Creative Technologies

Faculty of Environment and Technology

University of the West of England

April 2022

Abstract

Computer music researchers dream of the perfect algorithm, in which the music generated is indistinguishable from, or even superior to, that composed by the world's most talented composers. However, the fulfilment of this aim remains ambitious. This thesis pursues a different direction, proposing instead that computer-generated music techniques can be used as tools to support human composers, acting as a catalyst for human creativity, rather than a replacement.

Computer-generated music remains a challenge. Techniques and systems are abundant, yet there has been little exploration of how these might be useful for end-users looking to compose with generative and algorithmic music techniques. User interfaces for computer-generated music systems are often inaccessible to non-programmers as they frequently neglect established composition workflow and design paradigms that are familiar to composers in the digital age.

For this research, the Interactive Generative Music Environment (IGME) was developed for studying interaction and composition; building on the foundations established in modern music sequencing software, whilst integrating various computer-generated music techniques.

Three original studies are presented, based on participatory design principles, and evaluated with a mix-methods approach that involved studying end-users engaged with the IGME software. Two studies were group sessions where 54 participants spent an hour with IGME, in either a controlled (lab) environment or remotely as part of a conference workshop. The third study provided users more time with the software, with interactions studied and analysed with the use of screen recording technologies. In total, over 80 hours of interaction data was captured.

It was discovered that users need to understand several threshold concepts before engaging with computer-generated music, and have the necessary skills to *debug* musical problems within the generative output. The ability to do this requires pre-existing knowledge of music theory. The studies support the conclusion that computer-generated music is used more as a catalyst for composition than as a replacement for it.

A range of recommendations and requirements for building computer-generated music systems are presented, and summarise the contributions to knowledge, along with signposts for future work.

Acknowledgements

At first, a PhD appears a solo endeavour, when in fact few things in life truly are, and there are many for whom without their companionship, insight, support, and just generally *'being there'*, this PhD would not have been possible.

Firstly I must thank my supervisory team, Dr Chris Nash and Dr Tom Mitchell for believing in my ability to become something greater than I thought possible, and for supporting me through the many years of discovery and chaos. A special mention to my director of studies, professor Larry Bull for the rightfully needed *'grilling'*, and for always being available for coffee. To my examiners Professor Eduardo Miranda, Dr Andrew McPherson, and Professor Jim Smith for reviewing my work, and to Professor Aniko Varadi for acting as Independent Chair.

A special thank you to Dr Tom Mitchell, Phil Phelps, Dr David Creasey, and Martin Robinson for kindling my interest in programming and setting the wheels in motion that led to this moment. To Mr Paddy and Miss Morrison for giving me a stage to showcase my music, and for helping launch an *'interest'* into a career. And to the rest of the music technology team (Stephen, Max, Liz, Zak, Adrian, Luke, Nat, Martyn, Lukas, and Rich) for their support over the years.

To Dr Dom Brown for the camaraderie, this LaTeX template, and for being a brilliant IGME guinea pig. To Cameron Rye for being an excellent friend and motivating me to finish, when seemingly no one else could. To my late Grandpa, William John Hillier, for always taking an interest in whatever endeavour I chose, and sharing an interest in the art of *'tinkering'*. To Sally and Peter Little, for all the support along the way and for reading and critiquing this thesis. To Jean Bowden for reminding me that a complex problem is simply a sum of much simpler parts, and for giving me some time to work on them. To my Dad for teaching me honest principles, hard work, and what it means to be humble. I hope this work is a reflection of his saying *"If a job is worth doing, then it is worth doing properly - or not at all"*. Moreover, to Josh, Lilly and Mum, for all the little life lessons.

A special mention to the many students and musically minded people, that generously devoted their time to be a participant in one of the many user studies, that were crucial to the success of this thesis. Every piece of praise or constructive criticism was invaluable to IGME's success. Although you remain nameless your input shall ever be immortalised in this work.

I would like to thank the many people whom I have met along this journey, from places all over the world, whose experiences of this world are different to mine, who has taught me to have a

different outlook on life - and in troubled times there is more that binds us together than drives us apart.

To the rest of the graduate tutor team, for sharing a car on this roller-coaster ride, and a special mention to Virginia Power for looking after us all. I would also like to thank Dr Timo and the rest of the team at Aflorithmic for setting up the next adventure in my story.

Finally to the most important people: *Sophia Maryanne* and *Everly Amelia Rose* for teaching me patience, courage, and empathy. I hope one day you will *stumble upon* a copy of this work in some dusty box somewhere, and feel as proud of this work as I do. Although neither of you were here at the start of this journey you found a way to hitchhike along with it. And to Claire Bowden, for putting up with me spending all the long days (and nights) tucked away in my *'little-box-room'* office. Thank you for all the food, proofreading, and just generally being there; and adventures down roads less travelled.

For Sophia and Everly.

Declarations

This section details the publication created as part of completing this PhD. The author Samuel James Hunt is the sole author of the work listed, the additional authors Dr Chris Nash and Dr Tom Mitchell supervised the work.

The following publications are amalgamated as part of this thesis and appear in the following chapters:

Chapter 5 and 7.

Hunt, S., Nash, C., and Mitchell, T. (2017) Thoughts on interactive generative music composition. Laney, R. In: *2nd Conference on Computer Simulation of Musical Creativity*. Milton Keynes, Uk, 11-13 September 2017. Open University.

Chapter 6.

Hunt, S., Mitchell, T., and Nash, C. (2018) A cognitive dimensions approach for the design of an interactive generative score editor. Bhagwati, S. and Bresson, J. In: *Proceedings of the International Conference on Technologies for Music Notation and Representation – TENOR'18*. Montreal, Canada, 24-26 May 2018. Concordia University. pp. 119–127.

Chapter 9.

Hunt, S., Mitchell, T., and Nash, C. (2020) Composing computer-generated music, an observational study using IGME the Interactive Generative Music Environment. Michon, R. and Schroeder, F. In: *NIME 2020*. Royal Birmingham Conservatoire, UK, 21-25 July 2020. Birmingham City University. pp. 61–66.

The following publications were produced as part of completing this PhD, influencing elements of the research, or taking the research in a tangential direction, but are not included in this thesis. These are however inserted as appendix items:

Hunt, S., Nash, C., and Mitchell, T. (2017) How can music visualization techniques reveal different perspectives on musical structure. Lopez Helena, P., Solomon, M., Tucci, E., and Lage, C. In: *Third International Conference on Technologies for Music Notation and Representation*. A Coruña, Spain, 25-26 May 2017. Universidade A Coruña. pp. 69–78.

Hunt, S., Mitchell, T., and Nash, C. (2019) Automating algorithmic representations of musical structure using IGME: The Interactive Generative Music Environment. Paterson, J. In: *Innovation In Music 2019*. London, Uk, 5-7 December 2019. University of West London.

Hunt, S.J. (2020) Exploring Polyrhythms, Polymeters, and Polytempi with the Universal Grid Sequencer Framework. anon In: *Proceedings of the 15th International Conference on Audio Mostly.* Graz, Austria, 15th-17th September. Association for Computing Machinery, 101-06.

Hunt, S.J. (2020) An analysis of repetition in video game music. Strum, B. In: *Proceedings of the 2020 Joint Conference on AI Music Creativity.* Royal Institute of Technology (KTH), Stockholm, Sweden, 19th-23rd October. Royal Institute of Technology (KTH).

Contents

IV. EVALUATION **111**

Glossary

Term	Definition
Computer-generated music	Music generated by a digital computer based processes.
Generative music	Music generated by a stochastic processes.
Algorithmic music	Music generated by a deterministic processes.
IGME	The Interactive Generative Music Environment.
IGMESynth	A sub-application for synthesising sound sent from IGME.
Seed	The music sequence edited by the user.
Result	The music auditioned by the user once any computer-generated processes have been applied.
Parameters	Collection of computer-generated processes.
SPR	Collectively contains a seed, an arbitrary number of parameters, and a result. Seed, parameter, and result (SPR) model.
VCS	Version control system.
Iteration	A single output created by a computer-generated processes.
Part	Collective name for a container holding a SPR model and a list of edits (VCS). *Similar to a clip in most existing music software*.
Track	A collection of parts placed on a timeline.
Render	The processes of sequencing together multiple parts and tracks to produce a single output sequence.
Plug-in	A process that takes an input sequence and produces an output.
Note properties	A specific process that applies per-note stochastic processes.
Human part	A part with only human created music.
Human-computer part	A part with human created music and computer-generated processes.
Computer part	A part with only computer-generated processes.
Reference part	A part whose seed is taken from the seed or result of another part.
Iterator part	A part whose result is computed from iterating another part.
Repeater part	A part whose result is copied from a given range of bars.
Seed generator	A specific process that generates a seed dynamically.
Two-stage editing	IGME's editing paradigm whereby the musical output is created in two stages: an editor and a set of computer-generated processes.

Part I.

OVERVIEW

1. Introduction

The potential for machines to compose novel, interesting and *'human-esque'* music is an important and recurring theme in the history of computer music. Despite the abundance of flexible computer-based music sequencers, possible ways of incorporating computer-generated music techniques into such systems and subsequent composer workflow are seldom explored. This thesis explores and develops both theoretical and practical models for studying interaction inside an end-user computer-generated music system.

As detailed in Chapters 3 and 4, few tools exist for examining computer-generated music (CGM) inside established composition workflows using common digital design metaphors. Most tools for CGM exist as either programming base interfaces (textual or visual) or through novel graphical interfaces, using unfamiliar compositional workflows.

CGM has a rich history and remains an exciting field for researchers, but little work has been undertaken on how CGM can be adopted, appropriated, and authored by a human operator. This research seeks to answer the following two questions:

- *"What are the interfaces, tools and, workflows needed for engendering interactive computer-generated music composition?"*
- *"What role can computer-generated music take within composition?"*

1.1. Computer-Generated Music and Technological Context

Computer music, defined as music that cannot be created without the use of computers (Cope and Mayer, 1996), is an umbrella term for all activities that combine music and computers. Generative music, automated composition, and algorithmic music are often interchangeable terms referring to a formal process whereby music is composed with minimal human intervention (Alpern, 1995).

Wooller *et al.* (2005) define generative music as a process where the output of an operation generally has more musical predisposition than the input data (e.g. parameters) and the size of such data (e.g. notes) increases. Generative music is used primarily in one of two contexts:

- In a live context, where each recital is different from the next, e.g. Terry Riley's in C (Potter, 2002).
- For a fixed composition, where generative processes are used to create a static composition, performed by musicians, or synthesised by a computer, that is the same for each recital, e.g. Melomics (Quintana *et al.*, 2013).

This research focuses on the latter.

Although automated composition, generative music, and algorithmic music are interchangeable terms within the existing literature, generative music is defined in this research as music created and sequenced by an automated stochastic process, whereas algorithmic music is a method for generating deterministic music using a set routine.

For example, a simple generative process is one where a computer program generates 8 random notes in the C4-C5 range. This process is considered non-deterministic as each completed iteration can produce a different output. The number of unique sequences that can be produced is extensive but finite (in this instance). By contrast, an algorithmic process might take all notes in an existing sequence and reverses their order. This process is deterministic because, for each iteration, the output is predictable and will always be the same.

Historically both processes described above have been utilised in regular music practice. Mozart's dice game (Ruttkay, 1997) is an example of a generative process, while techniques such as inversion and retrograde have been used for algorithmic composition for centuries (Todd, 1978). Furthermore, McAlpine, Miranda, and Hoggar (1999) observe that, even in the earliest history of composition, there is evidence of many composers using Fibonacci sequences to create algorithmic music.

While this research focuses on music composed with digital computers, generative and algorithmic music techniques have also been realised mechanically, and extensively detailed in (Cope and Mayer, 1996; Ariza, 2005; Miranda, 2001). *Computer-assisted composition* (Assayag *et al.*, 1999) is used to describe systems that facilitate composition using a computer. Many existing tools can be grouped like this, including: digital audio workstations e.g. Logic (Apple, 2020b); score-editing programs e.g. Sibelius (Avid, 2020a); programmable environments, e.g. Max

(Manzo, 2016); and live coding interfaces, e.g. Sonic Pi (Aaron, 2016). *Computer-aided composition* (Buxton, 1977; Bouche *et al.*, 2017) is a similar term used to describe software that allows composers to design computer processes for generating musical structures and data. This label can be applied to many music programming systems.

Computer-generated music produces two different types of output: sequenced music events and digital audio. Research into both applications is extensive. This research focuses on the output of a computer-generated process that produces a sequence of music events, such as MIDI. A framework for categorising different techniques for computer-composed music is explored in Chapter 3.

Existing computer-generated music systems and languages often require the expression of musical structure using programming language syntax. This is a skill that must be developed by composers in addition to learning digital music sequencing. For example, Sonic Pi (Aaron and Blackwell, 2013), Impromptu (Sorensen and Gardner, 2010) and Supercollider (Wilson, Cottle, and Collins, 2011) provide tools supporting computer-generated music but require expert knowledge of the system and a transition into a different style of composition workflow. Although these systems have obvious merits, such as micro-detailed timing and pitch, they are often alien to formally-trained musicians and composers. Many other systems, such as neural networks and evolutionary techniques (Chapter 3), require a composer to be competent with the underlying mathematical models before being able to utilise those models as tools for music composition. In essence, such systems are often designed primarily for computer scientists, not musicians.

1.2. Project Aims and Contribution to Knowledge

This thesis explores and aims to answer the following research questions:

- What tools do music practitioners require for working with computer-generated music?
- How do these tools differ from existing systems and interfaces?
- How can computer-generated music processes be appropriated in a co-collaborative creative process?

The contributions to the field of computer-generated music are:

- A set of design heuristics for building end-user computer-generated music systems.
- How are computer-generated music techniques used in real-world practice.
- The types of users who will engage with computer-generated music and the knowledge they require to do so.
- The role of the computer in co-collaborative music practice.

Before designing end-user computer-generated music systems, four overarching problems must be discussed and explored through the literature. These are:

- How can automated techniques for generating music be categorised, and which of these are most suited for composition?
- What design metaphors can be borrowed from existing music sequencing software?
- How can a workflow for using interactive generative music be modelled?
- How can the interaction between composer and computer be studied and evaluated?

1.3. Methodologies

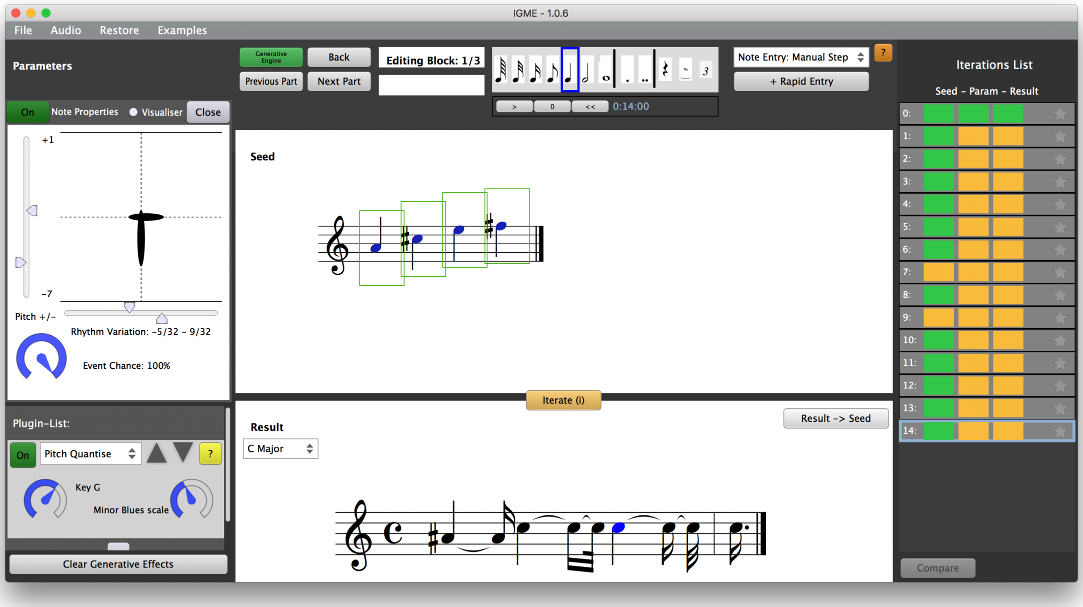
This thesis takes a mixed-methods approach for studying interaction, notably, the end-user tools are underpinned by participatory design (Muller and Kuhn, 1993). Qualitative methods, including descriptive video analysis and user surveys, are the primary methods for evaluating end-user experiences in this thesis, supported by quantitative interaction logging techniques (detailed in Chapter 8). All studies received approval from the university's ethics committee and were GDPR compliant.

The Interactive Generative Music Environment (IGME) software (Figure 1.1) was specifically developed using iterative design and evaluation, as a platform for studying user interaction. Through the research (Chapter 4), the features of existing music sequencers were identified and used as the foundations to build IGME. Studying existing computer-generated music techniques (Chapter 3) revealed those that were most suitable for end-user interaction, and through developing various models (Chapter 5), allowed these to be integrated into IGME.

The IGME software was designed to support composition in a range of genres and styles within the scope of *"Western music"* (within the affordances of common practice notation) and permitted the enactment of computer-generated processes upon them.

IGME was built purely to test research theories and is a means to an end, rather than a research contribution, and demonstrates one way in which end-user computer-generated music composition can be made more accessible (discussed at length in Chapter 7). Some technical details of the implementation and functionality are given in the development section (Chapters 5 - 7), and these are intended to give a vocabulary for discussing specific features in context during the evaluation section.

Fundamentally, IGME was a probe for gathering broad knowledge about how music practitioners use computer-generated music software. Its extended iterative development process created an opportunity for reflecting on both the design and use of related tools, leading to a set of empirically evaluated design heuristics (part of this work's contribution to knowledge - see Chapter 11) for future end-user computer systems. To this end, several chapters focus on the evaluation of IGME interaction, to develop and test specific aspects of the proposed design heuristics.



Editing Block: 1/3

Note Entry: Manual Step

0:14:00

Rhythm Variation: -5/32 - 9/32

Event Chance: 100%

Key G

Minor Blues scale

C Major

Figure 1.1.: IGME, the software built to study end-user interaction with computer-generated music.

1.4. Thesis Structure

The remaining Chapters of this thesis are divided into three main parts: background (Chapters 2-4), development (Chapters 5-7), and evaluation (Chapters 8-10).

Part II: Background

The project draws upon many fields that inform the overall research. Chapter 2 investigates the psychology of creativity in order to understand how a music composition workflow can be computerised or automated. Understanding this generates an appreciation of which elements of creativity a computer can realistically replicate. The Chapter also examines fundamental concepts in human-computer interaction (HCI).

Chapter 3 reviews the broad history of computer-generated music techniques, with Chapter 4 presenting a survey of existing popular music sequencing software, programming packages, and specific generative systems. Together, these two Chapters highlight the lack of support for CGM techniques inside existing music software, and the disparity of workflows found between mainstream music software, generative music systems, and music programming environments.

Part III:Development

Chapter 5 draws together the fields of computer-generated music, HCI, and creativity; defining a set of design requirements for supporting computer-generated music composition. This work creates the foundations to develop IGME. Chapter 6 uses the Cognitive Dimensions of Notations framework to compare existing interfaces for CGM and uses this to help define a set of design guidelines for building end-user CGM systems. Through this process, the development and rationale behind IGME's features are detailed. A full technical description of IGME is then presented in Chapter 7.

Part IV: Evaluation

An overview of the studies and methodologies employed is detailed in Chapter 8, along with two informal pilot studies conducted using a participatory design-based approach. IGME is further evaluated empirically with formal user studies spread across both controlled and uncontrolled conditions. Chapter 9 presents the findings of two workshops where 54 participants used IGME to compose music. Chapter 10 discusses a longitudinal study in which four participants used IGME for 4 hours each. The session was captured with a screen recorder and analysed to depict themes.

Chapter 11 (part V) reviews the project's aims and reveals the contributions to knowledge.

Part II.

BACKGROUND

2. Computational Creativity and Human Computer Interaction

Music composition supported by digital technologies combines two disciplines: creativity and human-computer interaction (HCI), each forming a section in this chapter. Music is the expression of creativity, and HCI encompasses many aspects of a musician's interactions with a computer.

2.1. Computational Creativity

Creativity is difficult to define, evaluate and model; and is surrounded by subjectivity and mystery. The purpose of its study is to determine whether generative music can simulate established aspects of creativity, and the roles it can take within existing compositional workflows. Papadopoulos and Wiggins (1999) propose that artificial intelligence (AI) researchers should aim to answer: *"Do we want to simulate human creativity itself or the result of it?"*. Both concepts produce different types of systems; the first is more difficult to develop than the latter.

It can be argued that for a computer to be an artist of creative media, it should follow existing models and theories of creativity. In addition, there are parallels between music composed by computer, music composed with the assistance of a computer, and music composed purely using traditional methods. Jacob (1996) makes similar notes, relating the incremental revisions of the creative process to those of the iterative tasks undertaken by computer algorithms, further arguing that this element is realisable by the computer. Gartland-Jones (2002) puts forward a generative algorithm that is able to model part of the creative process of composition. Fernández and Vico (2013) considers automated composition systems as ones whereby the user is not the main source of creativity, but whose purpose is to set model parameters, encode musical knowledge, and provide human-composed examples from which the system can learn.

Boden (2004) studies and identifies three core types of human creativity: unfamiliar combinations of familiar ideas; exploration of an existing conceptual structural space; and transforming this conceptual space. For example, combining a known harmonic progression in an unfamiliar way with a familiar melodic progression relates to the first type of creativity. Some composers explore the limits of a genre or structured form of music (second type), whereas others attempt to fragment the conceptual space altogether (third type). All three concepts can be simulated with automated music processes.

For the first type, a program could blend a randomly generated sequence with an existing user-defined sequence. The second type could be modelled by a neural network trained on 'familiar-pieces', which explores the limits of the created conceptual space. The third type can be modelled by many computer music techniques, for example even a simple stochastic processes can break all the *'rules'* of a genre.

Sternberg (2003) states that creativity is the ability to produce novel, high quality, and task-appropriate work. Current generative music techniques easily meet the originality criteria but struggle to meet quality requirements. Automatically measuring *'appropriateness'* is both difficult and subjective. The work in this thesis focuses on an interactive context, wherein the user ultimately chooses whether or not the created content is an appropriate addition to their composition. If however, the composer is writing music for a third party rather than themselves, this requirement may differ. Many researchers state that evaluating the output of generative music serves little purpose. Sorensen and Brown (2008) note *"objective measurement of music composition is impossible; it is dangerous as it can lead to rational explanations that do not bear experiential scrutiny."*

Sternberg (2003), in reference to work by Finke, Ward, and Smith (1992), discusses two main phases in cognitive creative thoughts: generative and exploratory. The generative phase creates structures with properties for promoting creative discoveries, while the exploratory phase manipulates these parameters, producing creative ideas. Relating this to a computer-generated music application, the generative phase creates the structures (for example an interactive Markov model) and then the user becomes the explorer.

Csikszentmihalyi (1996) argues that creativity is as much a cultural and social event as it is a psychological event. The author states that *"Original thought does not exist in a vacuum. It must operate on a set of already existing objects, rules, representations, or notations... Without rules, there cannot be exceptions, and without tradition, there cannot be novelty."* Such an observation may explain why simple stochastic music lacks widespread audience appeal. Therefore, CGM techniques should be *'educated'* on the nuances of the genre or type of music they are trying to simulate.

2.1.1. Knowledge, Motivation and Flow

Sternberg (2003) suggests that creativity is limited in that there may be a threshold beyond which creativity cannot happen without prior knowledge of the field. McIntyre (2008) supports this idea, stating that songwriters must acquire specific knowledge for creative practice, referred to as *domain acquisition*.

The ability to create, is at the highest level of the revised Blooms Taxonomy (Krathwohl, 2002), implying the need to fully comprehend the subject before being able to synthesise new material within that domain. Hanna (2007) specifies that creativity is the most complex cognitive process, requiring the need to remember, understand, apply, analyse, and evaluate musical knowledge before engaging with creative cognitive processes. On the contrary several styles and genres of music (and more wider creative artefacts) have appeared without necessarily following the aforementioned steps, as if stumbled upon by accident. This still however poses the question, can music practitioners be expected to use computer-generated music techniques effectively if they do not fully understand the domain? While it is unclear as to what degree any software envisioned by this thesis should educate users on techniques, it could be argued that generative techniques are most suited to experienced music practitioners, because to use them and evaluate their output requires knowledge of music theory[1].

There is considerable anecdotal and empirical evidence that creative production requires a high level of motivation (Collins and Amabile, 1999) often categorised as 'extrinsic' and 'intrinsic'. An extrinsically motivated individual is motivated by external factors, such as money or recognition. With intrinsic motivation, the output of the task, e.g. writing a song, is its own reward. As stated by Crutchfield (1962) *"the person is motivated by the intrinsic value in the attaining of the creative solution itself"*. The author believed greater creativity would result when a person was primarily intrinsically motivated to do a task. In some cases, extreme levels of extrinsic motivation and tight deadlines – for example, in life-saving situations – provoke creative solutions (Amabile, Hadley, and Kramer, 2002). However, the presence of extrinsic motivators does not preclude that of intrinsic motivation, and the relative valence of each or synergy of both is rarely clear for a given

[1] This is discussed again in the conclusion, and found to be mostly true.

moment or agent – in the previous example, saving lives is as profound a personal experience as it is social. However, in summarising the field's findings, Collins and Amabile (1999) note that, in general, extrinsic motivation and external deadlines are detrimental to creativity.

Echoing similar thoughts, Csikszentmihalyi (1990b) suggests that high levels of intrinsic motivation, accompanied by relatively low levels of extrinsic motivation, may help creative individuals be more independent of their field because they are unlikely to conform. Alternatively, creating noteworthy and stand-out creative artefacts in the pursuit of large extrinsic (fame and money) rewards, are also possible avenues in which creativity may flourish.

A number of factors influence intrinsic motivation, which is a crucial element in creative endeavours, including *'flow'*; which can be simply defined as a state where by a person is fully immersed in an activity to the exclusion of external factors. Notable work in defining the theory was proposed by Csikszentmihalyi (1996) and there is evidence to suggest that composers often find themselves in a flow state (Nash, 2011). Csikszentmihalyi (1990a) argues that people involved in creative pursuits actively seek flow experiences and that creativity is more likely to result from such experiences.

When it comes to flow, as researched by Nash (2011), a balance has to be struck between the challenge facing a user and their skill in the area. Too much challenge combined with low skill can result in anxiety, whereas the opposite can induce boredom. When the challenge matches a user's skill, a high level of intrinsic motivation is achieved, resulting in a state of flow, creating both opportunities and difficulties for computer-generated music. For example, a computer generating complex material for novice users is unlikely to be rewarding; the same would be true of generating simplistic material for adept composers. Using certain techniques, such as neural networks, which requires knowledge of maths, computer science, and music, might be overly complex. Moreover, in an ideal world in which a computer generates music better than any composer would have no challenge. In summary, to harmonise a user's skill with challenge, a compromise between easy and complex techniques is required, potentially leading to higher occurrences of flow states.

2.1.2. Modelling Composition and Creative Problem Solving

While current literature includes little exploration of formal models for defining a generative composition workflow, notable work has been done on modelling ordinary human-led music composition.

Alty (1995) presents work in which the composition process is modelled as a journey through a series of state spaces, where each state represents a possible direction in which the music can go. Constraints are often applied to these states, limiting the choice of direction and morphing the task into a problem-solving exercise. Many existing systems aim to reduce the search space by cutting the number of options available to the composer, creating an abstraction of the musical domain (Nash, 2015). For example, Sibelius (Avid, 2017) supports all the actions that can be performed with pen and paper, but the scale mapper (discussed in Chapter 4) prevents the user from entering certain notes, creating tension between ease of use and unconstrained composition. In general, systems emphasising usability create abstractions of the musical domain (Nash, 2015), reducing the problem-solving search space. For example, a step sequencer can be filtered to allow notes only from a certain scale or key. Existing devices exhibiting these types of abstractions include Ableton Push (Ableton, 2020), Numerology (Five12, 2020), and the Novation Launchpad Pro (Ampify-Music, 2020). Creating abstractions of musical data is commonly used in machine learning applications within the context of music, to reduce the search space (Choi, Fazekas, and Sandler, 2016; Todd, 1989; Papadopoulos and Wiggins, 1999), for example using MIDI instead of RAW audio.

A framework for modelling open-ended creative activities, such as music, is presented by Holland (2000). This is based on choosing a goal, selecting constraints, and then iterating through solutions. A result is generated throughout each step and the constraints and goals are adjusted until some *acceptance* criteria are met. The model is built on a framework that presents sets of modular but interacting components: a module for applying and planning constraints; an interactive interface; an existing corpus of reference material; and a series of plans for generative prototype musical material.

Holland (2000) defines music composition as ripe with *'problem seeking'*. Unlike other fields, there is often no generic goal or problem to be solved; rather, the problems composers set themselves to solve are ill-defined and open-ended, with few (if any) criteria for completion. Such concepts are at odds with computer music algorithms which, by definition, are often *'problem-solvers'*. Music composition is also full of *'wicked problems'* with no clear goals, no criteria for testing correct answers, and no comprehensive set of well-defined methods (Rittel and Webber, 1973). Nash (2011), in reference to Rittel and Webber (1973), highlights the parallels between artistic expression and "wicked problems".

2.1.3. Unison of Human-Computer Creativity

Human-computer co-creation can be defined as creativity in which both a human and a computer take responsibility for the generation of a creative artefact (Kantosalo *et al.*, 2014). Adapting creative software for supporting human-computer co-creation often requires redesigning major aspects of the software. Lubart (2005) puts the computer's role when supporting creativity, into four categories:

- As a management aid (nanny).
- As a communication enabler (pen-pal).
- As a creativity enhancer (coach).
- As a co-creator in the creative act (colleague).

Most of the categories above can be used as a label, in music composition activities. For example, digital audio workstations can be used to collate and manage fragments of musical ideas *(nanny)*. Ohm-studio (2020) enables users to collaborate on musical projects *(pen-pal)*. Few music tools can be considered as *coaches*, especially with automated music techniques that assume domain specific knowledge, whereas many generative music tools are co-creators *(colleagues)*, generating music automatically (but often not collaboratively). Kantosalo *et al.* (2014) notes how co-creative systems can transform the lives of professionals and lay people alike by increasing their creative potential. Focusing on user-centred design principles can transform computer creativity software into tools that enable human-creativity co-creation.

2.2. Human Computer Interaction and Usability

Human-computer interaction (HCI) focuses on the design and usability of interaction systems and employs a range of disciplines (Card, 2018). The term *'interaction design'* is concerned with the theory, research and practice of designing user experiences, for technologies, systems and products (Preece, Sharp, and Rogers, 2015), and a subset of the wider HCI field. Historically, before graphical user interfaces (GUI) became common, most computing was done on terminal-based consoles, with commands executed in text. As GUIs became more widespread, so did research interest in HCI (Card, 2018).

Usability is about the effectiveness, efficiency and satisfaction of a task, whereas user experience (UX) encompasses all aspects of the experience when interacting with the system. Different types of software generally emphasise one over the other, however considering both is important. Notably, *effectiveness* and *efficiency* are less of a focus in software built for creativity than one purely for productivity.

Usability, as defined by Nielsen (1994), generally has five components: *learnability, efficiency, memorability, errors, and satisfaction*. All systems have to be learned (*learnability*), but there is a trade-off between the skills and motivation of envisioned users, the time needed to learn the software, and the overall complexity of the software. A low-entry threshold (easy to get started) and high ceiling (difficult to master) are desirable characteristics for music software (Wessel and Wright, 2002). *Efficiency* is not a label meaningfully applied to creative tasks, such as music composition, given its open-ended, problem-seeking domain. However, certain aspects, such as transcribing a score from paper using notation software, can be measured in terms of *efficiency*. Most music software shares common interfaces and design metaphors, such as the sequencing and arranging of clips upon a timeline. The ubiquitous copy-and-paste is an ideal illustration of *memorability* that is almost universally implemented in software. In general, *errors* encountered when interacting with software should be avoided, although music is sometimes an exception, as errors can result in *happy accidents*, in which a mistake is accepted, leading to surprising musical directions. *Satisfaction* with music software is measurable in two ways: how satisfying the software is to use, and the satisfaction derived from the quality of the work produced. The first is relatively easy to assess, but evaluating perceived satisfaction with musical output remains challenging.

2.2.1. Design Methods

Formal methods for designing software are extensive in current literature (Budgen, 2003). The focus of this research is on the *'borrowing'* of existing music sequencing workflows, paradigms, and design metaphors. Therefore, much of the groundwork has already been done.

This thesis primarily deploys participatory design (discussed in Chapters 7 and 8), which involves engaging users as agents in designing the software, and continuous evaluation (Muller and Kuhn, 1993). The aim is to produce an artefact that fits the needs and desires of the users more effectively than other methods. However, it is unlikely that notable technological breakthroughs, such as touchscreen devices, would have emerged through a participatory approach since the technology would have been too alien to users. Irrespectively, participatory design has become increasingly prevalent and accepted, even though, as Nielsen (1994) notes, users do not always know what they really want[2]. In summary, revolutionary designs need not always consult an existing group of users.

User-centered design differs from participatory design, in that the user is not a member of the design team but their needs are researched and evaluated by the designers. The shift from user-centered to participatory design, reflects a change in attitude from one of designing for users to one of designing with users (Sanders, 2002). Participatory design has been used for music-based research, including developing piano practice tools (Sanders, 2002), and for music interfaces (Geiger *et al.*, 2008). Originally IGME was developed using a user-centred approach, before transitioning to participatory design, which was necessitated through the use of several pilot studies (Chapter 7).

2.2.2. User Requirements

Different users can have vastly different requirements when it comes to software, so it is paramount to consider who the users are. This is often the first step in designing for usability (Nielsen, 1994). Software such as ProTools (Avid, 2020a) is targeted at experts familiar with studio workflow, emphasising shortcuts and time-saving features, and consequently has a reasonably high entry threshold. By comparison, GarageBand (Apple, 2020a), a simplified version of Logic, (Apple, 2020b) with minimal features represents a *'first-sequencer'* for novices.

Who the users of computer-generated music might be is a difficult question to answer, given the lack of research on the subject. As such, a cursory survey was undertaken to find out who a user of generative music might be, with the results discussed in Chapter 5.

2.2.3. Cognitive Dimensions of Notations Framework

A key aim of this project is to find out how computer-generated musical processes can be integrated into established composition workflows[3]. Several notable frameworks (Carroll, 2003) exist for designing interfaces for task-oriented traditional computer software. Carroll (2003) describes a task in terms of goals, operators, methods, and selection rules, in relation to the user's knowledge of how to complete tasks. This particular model is used for tutoring purposes and, for

[2] As automotive pioneer Henry Ford once stated, "If I had asked people what they wanted, they would have said faster horses" (Vlaskovits, 2011).

[3] methods for understanding current composition workflow are explored in more detail in section 4

many, is the go-to model for designing software. However, Liikkanen *et al.* (2012) notes that music interaction research has become marginal in HCI.

Green and Petre (1996) and Nash (2015), observe that creative tasks, such as music, are not well-suited towards conventional, goal-oriented HCI design. For example, Nash (2011) presents work that shifts the design focus from usability frameworks to *virtuosity*[4]. Nash's research observes over 1000 users interacting with music composition software, finding correlations between user action and evidence of *flow* experiences. In doing so, Nash (2011) suggests that more holistic HCI frameworks, for example, the Cognitive Dimensions of Notations (CDN) framework, are needed to design tools to support creativity, rather than more formal HCI frameworks.

Green and Petre (1996) proposed the CDN framework as an evaluation technique for visual programming environments, interactive devices and non-interactive notations. Nash (2015) has adapted this framework for use in designing and analysing music notations and user interfaces for digital and traditional music practice and study. Bellingham, Holland, and Mulholland (2014a) present similar work, using the dimensions' approach for analysing a representative selection of user interfaces for algorithmic composition software. Finally, the cognitive dimensions can also be thought of as discussion tools for designers (Bellingham, Holland, and Mulholland, 2014a).

Each dimension and its description is set out in Table 2.1. The descriptions are adapted from Nash's (2015) work. Chapter 6 discusses the CDN in detail, focusing on both existing systems and research and on evaluating the end-user software developed for this thesis. Furthermore, evaluation methods for the user studies later in this thesis (Chapter 8) are also built upon the CDN.

2.2.4. Methodologies and Evaluation

Aside from the CDN framework, numerous methods for evaluating user-facing software exist in the literature. Many prioritise traditional usability goals where there are defined objectives, for example, Preece, Sharp, and Rogers (2015) DECIDE framework. Computer-generated music systems are often evaluated through listening tests, asking users to evaluate objectively the audible quality of a generated artefact(Hsu and Sosnick, 2009). Few studies evaluate the software interface itself, as evidenced in Chapters 3 and 4. The goal of computer-generated music systems remains

[4] The user masters the software, instrument or notation.

Dimension	Description
1. Visibility	How easy is it to view and find elements of the music during editing?
2. Juxtaposabillity	How easy is it to compare elements within the music?
3. Hidden Dependencies	How explicit are the relationships between related elements in the notation?
4. Hard Mental Operations	How difficult is the task to work out in your head?
5. Progressive Evaluation	How easy is it to stop and check your progress during editing?
6. Conciseness	How concise is the notation?
7. Provisionality	How easy is it to experiment with ideas?
8. Secondary Notation	How easy is it to make informal notes to capture ideas outside the formal rules of the notation?
9. Consistency	Where aspects of the notation mean similar things, is the similarity clear in the way they appear?
10. Viscosity	Is it easy to go back and make changes?
11. Role Expressiveness	Is it easy to see what each part of the notation means?
12. Premature Commitment	Do edits have to be performed in a prescribed order, requiring you to plan or think ahead?
13. Error Proneness	How easy is it to make annoying mistakes?
14. Closeness of Mapping	Does the notation match how you describe the music yourself?
15. Abstraction Management	How can the notation be customised, adapted, or used beyond its intended use?

Table 2.1.: Terms of the Cognitive Dimensions of Music Notations framework used in this research taken from Nash (2015).

focused on creating music indistinguishable from traditional human-led composition, attempting to pass the famous Turing Test (Ariza, 2009).

Brown, Nash, and Mitchell (2017) summarise methodologies used within NIME[5] and ICMC[6] conference proceedings and found that asking users to complete a *'specific task'* is the most common method followed by *'open exploration'*. In addition, questionnaires are the most popular way to collect data as they permit quantitatively analysing otherwise qualitative elements of interaction. The authors observe a move from traditional user experience to subjective evaluation, such as measuring engagement, enjoyability and reward. Frustration is rarely assessed.

[5] New interfaces for musical expression.

[6] International computer music conference.

2.3. Conclusion

This chapter reviewed computational models of creativity and methods for designing and evaluating tools to support creative activities. The chapter highlights the factors that catalyse creative endeavours, focusing on intrinsically rewarding tasks, ensuring that users have enough knowledge of the domain, and promoting a state of flow in which the challenge encountered by the user is in balance with their skill. Users' knowledge of the domain will influence the degree of challenge enabled by the software, ideally resulting in a *low barrier to entry* but a high ceiling. This promotes the idea that any end-user system proposed by this research should be familiar enough to get started easily, but different enough to make mastering it challenging, emphasising that there is no linear way of working through it. Crucially, this research does not consider computer processes as a replacement for human creativity, but rather as means for enhancing human creativity.

There is no single solution for supporting creative tasks, nor a magical algorithm that can act as a replacement. The definitions of creativity explored in this chapter have demonstrated applicable routes in which computer-generated music can mimic certain behavioural characteristics found in creative activities. Such generative processes have not yet been discussed, but the knowledge needed to relate such processes to definitions of creativity has been defined by this chapter.

3. Computer-Generated Music Techniques

There is extensive literature on generative and algorithmic music systems covering everything from 10th Century mechanical composition methods (Ariza, 2005) to supercomputer systems (Quintana *et al.*, 2013) capable of album-length composition. Modern approaches utilise breakthrough deep-learning techniques (Sturm *et al.*, 2019). Only systems using a *'digital computer'* for music are explored in this research; for more in-depth discussions of mechanical systems see Cope (1991) and Ariza (2005). This thesis focuses on the noteworthy, novel, popular, and most appropriate computer-composed techniques[1]. The work in this chapter was heavily influenced by Wooller *et al.* (2005), Cope (1991), Miranda (2001), Papadopoulos and Wiggins (1999), Fernández and Vico (2013), and Liu and Ting (2017). A recent synopsis of the field summarising the strengths and weaknesses of different techniques is given by Siphocly, El-Horbaty, and Salem (2021), and a in depth discussion of the *state-of-the-art* for music and AI is summarised in Miranda (2021).

Brown (2004) notes that, historically, artificial intelligence had the incredible potential for improving generative music, but the techniques were limited by poor computational power. Musical history is heavily aligned to technological breakthroughs such as those for the piano, electric guitar, turntable, and MIDI sequencer, with each leading to the development of new music genres (Katz, 2010). Just as instruments have brought monumental shifts in popular music, advances in computing power have led to the development of new computer music techniques. For example, deep learning technologies (supported by near limitless computing power) permit the development of exemplary tools such as Aiva (Zulić, 2019).

Most techniques discussed in this section use probabilistic methods[2]. Temperley (2007) emphasises how interwoven music and probability are, with the author demonstrating and modelling many musical concepts. For example, a musical key defines a likely distribution of pitches within a piece of music, such as pitch C is more likely than C# in a piece conforming strictly

[1] A complete analysis and description of the vast history of generative music would merit a thesis on its own.

[2] This section assumes the reader is familiar with elementary statistics and probability, although an overview of statistic and probabilist models (including advance concepts) in music examples is given by (Lorrain, 1980).

to C-Major. Moreover, Antoine and Miranda (2016) note the strong links between maths and music in their summary of algorithmic composition.

3.1. Categorising Computer-Generated Music Techniques

Technique	**Type**	**Feature-space {Music, Parameters}**	**Reproducible**	**Appropriate hierarchal level**
Random number generation	Generative	{None, None}	No	Note - Phrase
Deterministic Rules	Transformational	{None, Simple}	Yes	Note - Global
Probabilistic rules	Transformational	{None, Simple}	No	Phrase - Section
N-gram models	Analytical	{Dataset, None}	Yes	Phrase - Section
Markov chains	Analytical	{Dataset, None}	Yes	Phrase - Section
Simple grammars (type 3)	Transformational	{Fractional, Simple}	Yes	Note - Phrase
Complex grammars (type 0)	Transformational or Analytical	{Dataset, Complex}	No	Phrase - Section
L-Systems	Transformational	{Dataset, Complex}	No	Section
Neural networks	Analytical	{Dataset, Simple}	Yes	Section
RNN and LTSM	Analytical	{Dataset, Simple}	Yes	Global
Deep learning	Analytical	{Dataset, Complex}	Yes	Global
Evolutionary and genetic algorithms	Generative	{Fractional, Complex}	No	Section - Global
Cellular automata	Generative	{None, Simple}	No	Section - Global
Arpeggiators	Transformational	{Fractional, Simple}	Yes	Phrase
Harmonisers	Transformational	{Fractional, Simple}	Yes	Note

Table 3.1.: Various computer-generated techniques categorised using the framework discussed in this section.

Although there are many techniques for CGM, they are difficult to group and categorise. Wooller *et al.* (2005) defines a framework for comparing CGM processes that allow systems to be related and contrasted. Techniques are grouped into three categories:

- Analytic: reduce the size of input (or training) data to extract specific features by, for example, obtaining a set of notes from a database of sequences.
- Transformational: transform a structural element of the music by, for example, modifying pitches without altering rhythm.
- Generative: produces music from input data, or rules, resulting in an increase in data size, for example, a random note generator.

These elements are also considered in terms of their contextual breadth, which is defined by Wooller *et al.* (2005) as the size of the surrounding data that can influence the computation of the algorithm. For example, a CGM process that is controlled by 8 parameters has a broader *'contextual breadth'* than a simple process controller by a single parameter. The authors further note that future research might focus on how *'contextual breadth'* might be measured.

Developing on Wooller *et al.* (2005) work, this research establishes a *'feature space'* category, that defines for a given computer-generated process what external inputs are required. For example, applying retrograde, a transformational algorithm that reverses an input sequence, requires no parameters but needs the entire music sequence. In contrast a machine-learning model requires a vast dataset and complex parameters. This research defines a two-dimensional *feature space*: with *music* and *parameters* as separate dimensions. For music, the input is considered to be either *none*, *fractional* (a small music sequence), *full* (the entire sequence), or *data set*. For parameters, the input is either *none*, *simple*, or *complex*. Examples of these applied to various CGM techniques are given in Table 3.1. Some techniques, for example a Markov model, could be fed a continuum of music data from *fractional to dataset*, however the model becomes more *'generalisable'* with a *dataset*.

The level of reproducibility (deterministic versus stochastic) is a further consideration for creating a framework to define generative applications. Simple transformations, such as retrograde, are both reproducible and exact for the same input data, and therefore deterministic. Simple neural network systems (Todd, 1989) and seed-based random models have similar properties, whereas stochastic and evolutionary systems, which use random seeds (Myhill, 1979; Biles, 2002), produce a different output for the same input, making it impossible to perfectly reconstruct the results from the model.

Finally, each technique works on a different level of musical hierarchy or structure. Influenced by Jackendoff and Lerdahl (1996) (who approach the analysis of music in terms of hierarchical levels also) this thesis defines 5 levels: single note; phrase (a small sequence); bar; section (several bars); and song. To maximize each algorithm's efficiency, it is paramount to consider its hierarchical level. For example, simple probabilistic models such as Markov chains, are more suited to the note level than to the section level. This research hypothesises that algorithmic music systems are less suited to be used at a global level (the entire composition) than to be used at the lower levels (note and phrase), and is evaluated through user studies in later chapters.

The framework in this thesis used to categorise music algorithms, extended from Wooller *et al.* (2005), consists of four dimensions:

- Type: *analytic, generative or transformation.*
- Feature space (two-dimensional):
  - Music: *none, fractional, full, or dataset.*
  - Parameters: *none, simple, or complex.*
- Reproducibility: *reproducible or non-reproducible.*
- Appropriate hierarchical level: *note level, phrase level, bar level, section level, or song level.*

This framework is discussed again in Chapter 5.

3.2. Early Work in Computer Music

The following outlines early experiments in probability, and music programming systems. Many of the experiments used large mainframe computer systems, inaccessible to the average composer. During the 1950s and 1960s, computers remained mainly the property of large businesses, the military, and universities. The operators of such machines were computer scientists rather than musicians. Such systems permitted only rudimentary text-based interfaces (as GUI had not been invented), requiring competency with primitive programming languages. The growth of personal computers during the 1980s democratised access to computers for the average person, supported further by the invention of graphical user interfaces (Webster, 2002).

Early work in computer music was influenced by Shannon's *information theory* (Shannon, 1948; Shannon, 1953) and Chomsky's work on linguistics and formal grammars (Chomsky, 1957). Hiller and Isaacson (1959) created rule-based programs that could generate various forms of music (producing the Illiac Suite for String Quartet), which is often cited as the earliest example of automated computer composition [3].

[3] A comprehensive history of early computer music systems is detailed by Cope (1991).

The Iliac Suite consisted of four experiments (Hiller and Isaacson, 1979), with each focusing on a different musical style: first species counterpoint; counterpoint rules to random white notes; rhythmic elements with chromaticism; and Markov models (discussed shortly). In summary, the authors concluded that *"Digital computers are readily programmed to perform deductive reasoning, but their ability to draw generalizations from special cases is extremely limited"* (Brooks Jr *et al.*, 1957).

3.2.1. N-Gram Models

An N-gram model is used to describe the splitting of a continuous sequence into smaller sub-sequences. For example, the word *'music'* contains the bigrams (2nd order N-gram) *mu, us, si, and ic* (Doraisamy, 2004). Such processes can be applied to a simple monophonic sequence, whereby the next note is computed based on the previous N notes. Using a bigram model, the probability of an event is given only by its previous event - how likely is C# when the last event was C? The earliest example of this, as applied to computer-generated music, is the work of Brooks Jr *et al.* (1957), who created varying length N-gram models of sequenced notes. Using a table to store the data, each series from 1 through to 8 was collated, sorted numerically and tallied. Some constraints, including limiting music to incorporate only monophonic lines, and music with the same rhythmic measure, governed which data was used for building the N-gram models. Music was then re-synthesised by sampling randomly from the pre-collected data. Therefore, the sequences with high occurrence in the training data would more likely be chosen for output. Certain rules were applied to ensure the processes operated uniformly. General observations by the author suggest that varying the N-gram's size produced different results. With an order of 1, the program produced music that was chromatic and not in keeping with training data, whereas larger orders produced sequences that could be easily heard in the training data.

3.2.2. Markov Models

A Markov model is a stochastic model in which the probability of an event depends on the state of previous events. In music, this could be illustrated as the probability that a sequence will produce a pitch G following a previous pitch C, notated as $P(G|C)$. When states are combined, they are referred to as a chain of states. For example, the probability of the pitch sequence $[A, C, D]$ would be given by $P(D|AC)$ using a 2nd-order Markov, which defines the probability of pitch D given the previous two pitches $[A, C]$. The probability of such transitions have been both *artificially*

	Output note		
Input Note	**C**	**E**	**G**
C	0.7	0.1	0.2
E	0.4	0.6	0
G	0.5	0.25	0.25

Table 3.2.: A transition table (single order Markov model) that lists the probability of transitions between two notes.

composed and obtained through analysis of existing work (Ames, 1989). Table 3.2 illustrates a simple Markov model, in which there are three possible states (or three notes). For example, if our current note is a C, then there is a 70% chance that the next note in our sequence will be a C again, or a 10% chance to go to E, or a 20% chance to go to G. Markov models are often referred to as transition tables. A zero-order Markov model would be the same as an arbitrary random number generator. A drawback of Markov models is the inherent exponential complexity of working with larger order models (Mochihashi and Sumita, 2008), and with longer chains require considerable memory requirements. Despite their age, Markov models are still explored in modern literature (Van Der Merwe and Schulze, 2010; Antoine and Miranda, 2016).

3.2.3. Other Early Work

In the mid-1950s, Xenakis (1971) infused mathematical techniques in music composition using computerised techniques from Markov chains, game theory and stochastic music composition. Myhill (1979) later extended Xenakis' stochastic music language (SML), as well as proposing that generative music should be balanced in terms of stochastic and deterministic music, to avoid the loss of stylistic coherence.

Brooks Jr *et al.* (1957), created a tool that re-synthesised music through the analysis of existing musical structure. A novel idea presented in this work was for programs to determine rules using conclusions formed through analysis of music, to synthesise new music. Contrary to other work, the data model used here for generative music is constructed through a real-world dataset. Such ideas form the basis of many more modern techniques, such as those based on machine learning.

Olson and Belar (1961)'s work in computer music focused on analysing and creating distributions of musical elements from a database of existing songs. A limitation of computer hardware meant that pitch and rhythm combinations were kept separately. An unforeseen side effect was that musical fragments that were not in the original works could be output. For example, the output may contain a (C4, 1/8) → (C5, 1/16)[4] transition, a combination not present in the original dataset. This is permissible, as transitions from C4 → C5 and 1/8 → 1/16 are found in both independent sets. Although this work is fairly primitive, its application is both noteworthy and useful[5].

After creating several compositions with their music programming language MUSICOMP, Hiller and Baker (1964) acknowledged that the computer is more *compiler* than *composer*, with most of the music structure predefined by the user. The system employs ordinary music terminology, so is accessible to composers, but still requires domain-specific knowledge of mainframe programming. The basis of this work led to the development of the Digital-Alternate Representation of Musical Scores System (Cope, 1991).

Other work from this era included the automation of serialist style composition (Gill, 1963; Koenig, 1970; Koenig, 1971). Serialism itself is a genre that is fundamentally built on rules and procedures. This is one genre of music that can be realised either manually by a human or through an automated computer process.

[4] This notation refers to a C4 (MIDI note 60) quaver followed by a C5 (MIDI note 72) semi-quaver.

[5] This process is used in IGME's distribution sample plugin, see appendix item D.

Grammar	Language	Computer Type
Type-0	Unrestricted grammar	Turing machine
Type-1	Context sensitive grammar	Linear bound Automata
Type-2	Context free grammar	Push down Automata
Type-3	Regular grammar	Finite state machine

Table 3.3.: Types of Chomsky Grammars.

3.3. Grammars

Grammars are the structural rules of language. Prominent work by linguistics expert Noam Chomsky (1957) formalised grammar into 4 hierarchical levels. Grammar is used widely in music, both for modelling existing musical structure (Jackendoff and Lerdahl, 1996) and in systems for generating it (Holtzman, 1981). The components of grammar are formally described as:

V = A set of non-terminal nodes, written as upper case letters: (A, B).

T = A set of terminal nodes, written as lower case letters: (a, b).

S = A starting symbol (S).

P = And a set of rules for transforming non-terminal nodes into both terminal and non-terminal nodes: ($S \rightarrow aA$), ($A \rightarrow Bb$), ($B \rightarrow b$).

A production rule has the form $\alpha \rightarrow \beta$ where α and β are strings in either the set V or T, and at least one symbol of α belongs to V. The arrow symbol ($\rightarrow$) denotes an instruction to replace the string of symbols on the left side of the arrow with those on the right (Holtzman, 1981).

Using the grammar defined above, each iteration (rewrite) of this process gives the following output:

0 : S

1 : aA

2 : aBb

3 : abb

After the 3rd iteration, no further transforms can be computed as the sequence contains all terminal nodes. To create music from this, each character could be mapped to a musical event, for example, $a =$ C4 and $b =$ C5. This kind of grammar is defined as regular or type-3. More complex mappings and examples of grammars mapped to music are discussed by Miranda (2001).

A type-2 grammar (context-free grammar) introduces variables in production rules meaning that, for example, the rules $A \rightarrow aA$, $A \rightarrow AB$, are both applicable. Therefore, a probabilistic decision is made over which rules should be followed. Unlike type 3 grammar, this often produces stochastic output.

Type-1 grammars introduce context in their production rules. For example, in a sequence of symbols aBc, the production rule $a < B > c \rightarrow D$ applies only to symbol B when the previous symbol is a and the proceeding symbol is c. Type-0 grammars have no limit on permissible production rules, making them the most complex and powerful grammar. Table 3.3 shows a summary of grammar types.

Although grammars were utilised in early computer music work, more formal utilisation did not start until the late 1970s with the work of Lidov and Gabura (1973). More prominent work by Holtzman (1981) utilised the Generative Grammar Definition Language (GGDL) for exploring grammars applied to automatic music composition.

Holtzman (1981) analysed various pieces of music to produce both mappings and production rules, creating serialist style compositions. (Kohonen, 1989) used a context-sensitive grammar that learns its production rules from analysing examples of existing music. These grammar types are referred to as Kohonen grammars and have been used in other related fields of generative music, both as benchmarks for new techniques (Mozer, 1994) and fitness evaluation tests (discussed in evolutionary algorithms). More recent work using grammars is detailed by Gillick, Tang, and Keller (2010).

3.3.1. L-Systems

Lindenmayer systems (L-Systems) are a type of formal grammar originally designed to model the growth of plants and other biological organisms (Prusinkiewicz and Lindenmayer, 2012), but have been used to generate music. Worth and Stepney (2005) use such techniques to search for *'pleasing'* graphical and musical renderings output by L-systems. By default, an L-system produces tree-like structures as an image, mapping must take place for the musical output to occur; there are several ways of achieving this (Worth and Stepney, 2005). Prusinkiewicz (1986) uses a lookup table to map y-coordinates to notes and line lengths to note durations. Soddell and Soddell (2000) maps branch angles to changes in pitch and the distance between lines to a note's duration. McCormack (1996) maps L-system grammars directly to pitch, duration and timbre (terminal symbol to music event).

Standard grammars are processed in a linear fashion, so the resulting string can be re-written as the grammar is processed; whereas L-systems employ parallel rewriting, with all the rewriting done at once, and is a defining feature (Togelius, Shaker, and Dormans, 2016). L-Systems have been implemented as formal, context-sensitive, and stochastic grammars. Examples of these are given by Togelius, Shaker, and Dormans (2016).

Lim *et al.* (2017) present an L-system framework for generating music scores, designed for non-experts. The authors extended the system so production rules could be altered with genetic operators (see next section). The research presents an overview of different applications and their ability to work as interactive music generators, indicating potential applications for exploring L-systems in existing music composition software.

Fractals are mathematical models that produce interesting mathematical shapes and share similarities with L-Systems. Sukumaran and Thiyagarajan (2009) define fractals as *"an irregular and fragmented geometric shape that can be subdivided into parts, where each part appears to be the same in all ranges of scale"*. Fractals have been used as tools to generate music, normally achieved through mapping each x/y position (pixel) to a MIDI event (as fractals are normally images). A comprehensive overview is given by Madden (1999).

3.4. Neural Networks

A Neural Network (NN) is a type of computer model inspired by the biological neural connections of the brain, which are successful in language modelling, pattern recognition, and predicting time series data (Liu and Ramakrishnan, 2014). Early work in applying NNs to music generation was conducted by Todd (1989), but it was not until computing power developed sufficiently that the field was explored more thoroughly. As noted by Mozer (1994) such techniques offer the potential to overcome the various limitations of transition table approaches and musical grammars.

The main task of an NN is to model an existing dataset or problem space given labelled training data. In a musical context, this could result in a system that predicts the next note in a sequence, having been trained on a series of existing melodies. As the complexity of the musical structure increases, so does the overall size of the network and type. Simple NN implementations have been superseded several times by more complex models, including recurrent networks (Mozer, 1994), long short-term memory networks (Eck and Schmidhuber, 2002; Liu and Ramakrishnan, 2014), and generative adversarial networks (Kolokolova *et al.*, 2020). A more in-depth explanation of such networks is beyond the scope of this research. Many frameworks, for example, TensorFlow (Abadi *et al.*, 2016), abstract the complex implementation details for doing machine learning, and in general provide a low entry threshold for users.

Deep learning networks are large-scale neural networks suited to elaborate problems, such as natural language processing. Deep learning has been applied in many domains as well as for generating musical sequences (Sturm *et al.*, 2016; Liang, 2016). An overview of this is given by Briot, Hadjeres, and Pachet (2020).

There are several reasons for not pursuing deep learning (and other NN based techniques) more in this thesis, as discussed further in the next chapter, but in summary, such systems are designed to replace the composer entirely, require vast datasets, and long training times, so present many technical headaches. Furthermore, using the computer to generate entire pieces of music conflicts with the purpose of this research, which is to look at how humans and computers can create music collaboratively.

3.5. Evolutionary Algorithms

The application of evolutionary computing to music started in the early 1990s (Loughran and O'Neill, 2020) with the work of Horner and Goldberg (1991). However, the field itself began development in the 50s (Eiben and Schoenauer, 2002). Evolutionary computing attempts to solve computing problems by modelling them on the theory of *evolution*, which in summary: given a population of individuals and environmental constraints, which of these individuals are the strongest and survive through natural selection (Eiben and Schoenauer, 2002). The best candidates seed the next generation, with mutations optionally applied to each candidate.

Generally, evolutionary processes can be defined by four characteristics (Brabazon, O'Neill, and McGarraghy, 2015):

1. A population of entities.
2. A mechanism for selection (fitness).
3. A method for creating offspring (crossover).
4. A method for inserting variety (mutation).

Within a musical context, an evolutionary algorithm (EA) can be applied to generate a simple monophonic sequence. To illustrate this consider the following example. First, a population of 100

1-bar sequences, each containing 8 evenly spaced 1/8 (quaver) notes, could be randomly created. The selection process could simply correlate the pitch distribution, with that of an ideal *C-Major* key[6]. The top 50 1-bar phrases, with the highest fitness value (i.e. the ones most likely to be in C-major) are then selected as parents to create a new population. Two randomly selected examples are then combined by taking a note from each sequence in an alternating order to produce a child (crossover). From this, a *mutation* process is applied that randomly increments a pitch in each sequence by ± a semitone[7]. Finally, these 50 new children replace the lowest-scoring 50 examples from the population. This process could be repeated until a single example has a high enough fitness value. Although this example is simple, it illustrates the foundation concepts of an EA.

The two main methods for creating the initial population are either randomly or through rule-based processes. Brown (2004) defines rules for melody generation by analysing text designed for teaching novice composers. Those rules were then evaluated against existing melodies in Western music to determine their merit. The research applied the rules to the initial generation in an evolutionary process, noting that the quality of music was generally improved in comparison with a random population initialisation. A downside of this rule-based approach is that it limits the search space of the evolutionary process and guides it in a predefined direction.

Three problems are specifically applicable to music and evolutionary computing: the problem domain, individual representation, and fitness measure (Loughran and O'Neill, 2020). Music has many individual components that could be considered the *problem domain*. For example, the algorithm might be trying to create melodies. The music itself could be *represented* as MIDI, binary or raw audio. In general, music is a difficult subject to evaluate (*fitness measure*) objectively. The difficulty is exacerbated when giving that role to a computer, although the issue might be as simple as determining how well a generated melody fits within a given key profile, using a statistical test (Temperley, 2007). Loughran and O'Neill (2020) in summary explicitly state that *"no single fitness measure can autonomously, objectively and reliably determine what is good music"*.

GenJam (Biles, 1994; Biles, 2002) is a prominent generative system for generating jazz solos over predefined chord progressions. Despite using a genetic (binary) representation, the system makes use of musical mutation operators for reversing, rotating, and sorting the note order. The output is mapped from binary into one of 14 MIDI pitches. The evaluation stage (fitness) requires the

[6] For example using Temperley's (2007) key detection method.

[7] Other simple mutations in the literature (Sheikholharam and Teshnehlab, 2008) include: swapping adjacent notes, reversing a group of notes, and transposing a note's pitch by an octave

user to rate iterations, which are then fed back into the evolutionary process. Biles' notes that it is difficult to judge each instance because the piece of music must be auditioned from start to finish, making such a process time-consuming compared with an automated fitness evaluation. In conclusion, Bile's notes that a different technique (with different fitness, mutation and crossover operators) could be used to create a specialisation for the task at hand by, for example, creating a melody or chord progression.

Instead of evolving a musical output, Sulyok, McPherson, and Harte (2016) evolve the musical composition process itself. The fitness test evaluated statistical similarities between output and a corpus of *Bach* keyboard exercises. In reviewing music produced by the system, they praised repetition and variation but criticised other musical properties such as harmony and melody. In similar work by Sheikholharam and Teshnehlab (2008), the fitness function is established with deterministic rules obtained by analysing existing music and modelled using a Kohonen musical grammar.

Unehara and Onisawa (2003) present an interactive music system that generates music by allowing the user to rate the output of a Genetic Algorithm (GA). In turn, this feeds back into the computation of future musical output, operating as a human-in-the-loop system. The example demonstrates users listening to each 4-bar musical part and evaluating it on a 4-point scale. The time taken just to listen to all the generations is a minimum of 24 minutes (12 phrases, 8 seconds long, 15 iterations), assuming one full listen of each example. In reality, users of the systems spent an average of 64 minutes on the task and listened to 576 iterations, just to create 4 bars of music, which is unlikely to be practical or useful in the real world.

A recent and noteworthy development for EAs is the Melomics program (Diaz-Jerez, 2011). The system does not replicate any musical genre, but rather generates its own style based on thousands of generalised musical rules. Output from the machine was realised in an album-length composition called *Iamus*. It is unclear how many outputs the system created and how these were selected, so it is difficult to assess how adaptable the system is musically. Comments on performance videos of the piece suggested people admired what had been achieved, although they also cited a lack of both artistic value and musical progression. The system is difficult to reconstruct due to its proprietary and commercial nature.

In summary, evolutionary computing can produce musical fragments that make sense at the phrase level but is perhaps less suitable for producing larger musical structures. Unlike other generative techniques, evolutionary techniques allow a user to remain a *co-author*, through human-in-the-loop concepts, something that is considered crucial for this research. A conclusive overview of evolutionary music is given by Loughran and O'Neill (2020).

3.6. Cellular Automata

Cellular automata (CA) are discrete dynamical systems (often expressed as a 2D grid of cells) that modulate their features over time (McAlpine, Miranda, and Hoggar, 1999). In general, they consist of two elements: a matrix of cells that are in some kind of state, and a rule defining how the state of a cell modulates over time (Reiners, 2004). John Conaway's *game of life* is a canonical example of cellular automation (Adamatzky, 2010), in which cells 'live' or 'die' as time progresses, illustrated as a culture moving across a 2D plane. Despite seemingly mostly unrelated to music, researchers have used such concepts for music composition (Miranda, 1993). In a simple implementation, Reiners (2004) uses a single dimension CA (array). At each iteration, the row becomes a binary representation (each cell has a binary state). This representation is then converted into a MIDI note number[8], therefore producing a sequence that modulates over time. CA implementations include 1D, 2D, and 3D methods (McAlpine, Miranda, and Hoggar, 1999).

Due to the limited scope of this research, CAs are not explored further in this research. For overviews of the field, as applied to music, see Burraston and Edmonds (2005) and for MIDI implementations see Burraston *et al.* (2004).

3.7. Evaluating Computer-Generated Music and Usability

In summarising findings from the field of computer-generated music research, there remain several *issues* with musical output. Todd (1989) notes that generative programs can easily get stuck in cyclic loops. Existing theories, as observed by Pearce (2007), focus on certain musical elements in isolation, whereas in reality music is an exotic combination of these elements. Pearce and Wiggins (2007) suggest that some computational models often fail to meet the intrinsic stylistic constraints of the genre. The computer can easily make music that hits the extremes of ranges, or group notes all in the same register (Sorensen and Brown, 2008), therefore becoming inappropriate for the instrument, or performer. Finally, many of the systems reviewed are

[8] The modulo 128 operator is used to ensure the notes remain in range.

isolated into scientific research projects (Quintana *et al.*, 2013) without conforming to existing compositional practice, requiring domain-specific knowledge (Nash, 2015) and competency with the notation (Sorensen and Gardner, 2010). The most prominent of all issues is the lack of high-level structure (Sorensen and Brown, 2008; Mozer, 1994; Wiggins *et al.*, 1998) in musical output. Repetition is one of the most important characteristics of structured music (Rahn, 1993), and much of computer-generated music is devoid of it. Original research, conducted alongside this thesis, on repetition in music and computer-generated systems[9] was produced in (Hunt, Mitchell, and Nash, 2019; Hunt, 2020). In summary of this work, computer-generated music lacks structured repetition and has an imbalance of novel ideas.

As a general observation, many complex generative music systems set themselves the goal of automating human-esque composition, and such systems are not designed to work in unison with composers. As set out by the thesis in Chapter 1, computer systems should seek to aid the composer (for example providing inspiration), not to replace or remove them from the process. Fernández and Vico (2013) note that any method that automates the generation of creative work can be used as a tool to aid composers.

Several researchers using evolutionary computation have reported promising results using human-in-the-loop processes (Jacob, 1995; Takagi, 2001; Bryden, 2006). Sorensen and Gardner (2010) emphasises the need to include humans as part of the system, as an active agent in a cyber-physical world. Biewald (2015) suggests that human-in-the-loop computing is the future of machine learning, with Zanzotto (2019) noting that such systems reward users. Creating a system that acknowledges the strengths of both the human composer and the generative process enables effective collaboration.

Techniques vary considerably in terms of difficulty of implementation, comprehension by users, and integration into composition workflow. McAlpine, Miranda, and Hoggar (1999) state that simple stochastic algorithms are some of the easiest techniques to implement and comprehend. Despite much of the early work making use of probabilistic models, research on the application of algorithmic models to music is still an area of exploration over 50 years later (Chapel, 2003), with simple models found in modern music sequencing software (discussed in Chapter 4).

[9] Included in appendix item G

3.8. Conclusion

In conclusion, many techniques exist for generating music using a digital computer. It is apparent that certain techniques are better than others for a given composition task, but it is unclear what the most suitable technique is. Loughran and O'Neill (2020) notes there is no single best evolutionary method with which to approach music composition, and such an observation can be applied to most generative techniques. After experimenting with GAs, Wiggins *et al.* (1998) suggests that harmonization is better suited to a rule-based (expert) system. Biles (1994) advocates the use of genetic algorithms for creating specific elements of music, such as chord progressions and voicing or bass lines, in collaboration with the composer. In addition, Sorensen and Brown (2008) note that trivial systems that make use of probability, linearity, periodicity, set theory and recursion can be superior to complex systems. Instead of trying to establish the best processes for various musical activities, this research aims to leave this role to the users of any proposed end-user music system.

From studying this field, it remains unclear what purpose computer-generated music takes within composing music. This can be attributed to the fact that much of the work in this section has not been evaluated with users, as it has focused instead on musical quality. However, notable observations made by Edwards (2011) state *"Formalisation of compositional technique in software can free the mind from musical and cultural clichés and lead to startlingly original results"*. Likewise, McAlpine, Miranda, and Hoggar (1999) comment that *"Considered thus, it is apparent that the computer is merely a tool for the realisation of abstract design constructs, and is best employed as a labour-saving device to free the composer from performing menial calculations by hand"*. Formal definitions of the uses of generative music are theorised in Chapter 5 and evaluated in the later chapters of this thesis.

4. Existing Music Composition Systems

While, the previous chapter detailed techniques for computer-generated music, this chapter explores, in three sections, different interfaces for end-users. The first section takes a broad look at popular sequencers, digital audio workstations, and score editors, identifying features that might aid computer-generated music. The second section considers generative music systems that are more informal and perhaps less well-known (than the first section), while the third section considers systems that function primarily as programming environments, both textual and graphical. The three systems are loosely grouped as popular, experimental and programmable. A computer-generated system differs from a specific technique, in that it is a holistic approach to music composition via an interface.

The purpose of this chapter is to identify features that might enable interaction with computer-generated music. Many purpose-built generative music systems and more general-purpose programming environments inherently allow this, but this section attempts to prove how different these are in comparison with popular music sequencing software. Additionally, this section provides an overview of the common features and terminology found in music software.

4.1. Popular Music Software

A recent survey Sethi (2018), asked over 30,000 musicians and producers what their favourite digital audio workstations were, with the results shown in Figure 4.1, giving a broad overview of the most commonly used software. A list of software analysed for this section is shown in Table 4.1. This research did not look at linear audio editing and sequencing tools, such as Pro Tools, Ardour and Acid Pro, that replicate the studio workflow of using *tape* and are arguably more suited to recording, sequencing and editing linear audio than to composing music. The purpose of this section is to assess the feasibility of generative music within existing software, and the features

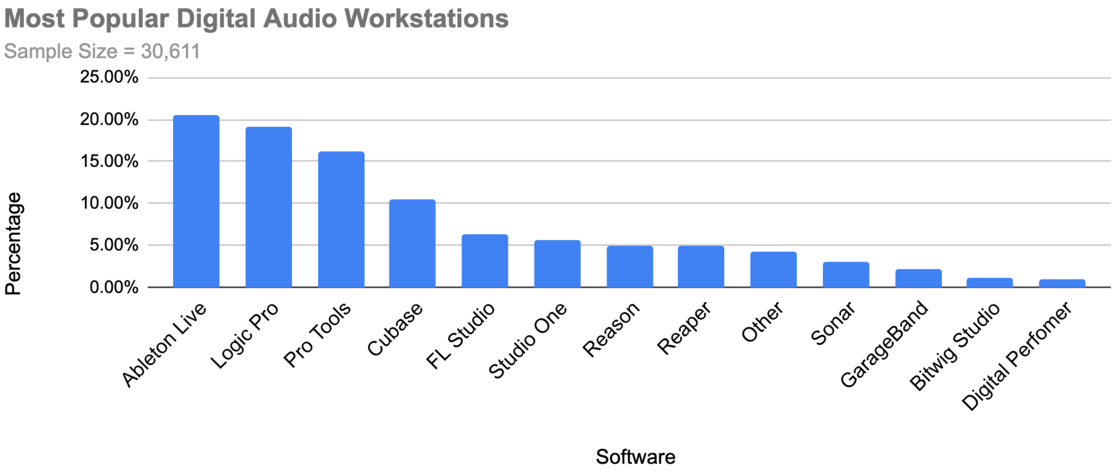


Figure 4.1.: A visualisation of the most popular digital audio workstations. Reproduced from (Sethi, 2018).

deemed necessary for supporting it. The software was analysed through lenses such as notation, workflows, plug-ins, and musicology. This chapter focuses on systems for composing and notating western music, in general, there is often limited support for working with non-western music and notations.

Name	Summarised	Reference	Version
Ableton Live	Yes	(Ableton, 2020)	9
ACID Pro	No	(MAGIX, 2020)	10
Ardour	No	(Davis, 2020)	5.12
Audiotool	Yes	(AudioTool, 2020)	2020
Bitwig Studio	Yes	(Bitwig, 2020)	3.1.2
Cubase	Yes	(Steinberg, 2020a)	10.5
Digital Performer	Yes	(MOTU, 2020)	DP10
Dorico	Yes	(Steinberg, 2020b)	3.1
FL Studio	Yes	(Image-Line, 2020)	20
GarageBand	Yes	(Apple, 2020a)	10.3.4
Guitar Pro	Yes	(Arobas-Music, 2020)	6/7
Logic Pro X	Yes	(Apple, 2020b)	10
Numerolgy	Yes	(Five12, 2020)	3
Pro Tools	No	(Avid, 2020a)	2020
REAPER	Yes	(Cockos, 2020)	6.07
Reason	Yes	(Reason-Studios, 2020)	10.4
Renoise	Yes	(Renoise, 2020)	3.2
Sibelius	Yes	(Avid, 2020b)	2020
Studio One	Yes	(Electronics, 2020)	4.6
Tracktion	Yes	(Tracktion-Software-Corporation, 2020)	10

Table 4.1.: A list of music composition software and digital audio workstations that were analysed in the first half of this chapter. To aid with narrative when discussing the systems listed in the above Table throughout this section, the citations are omitted (please refer to the ones listed in the table), specific systems not listed are cited ordinarily.

4.1.1. Notations

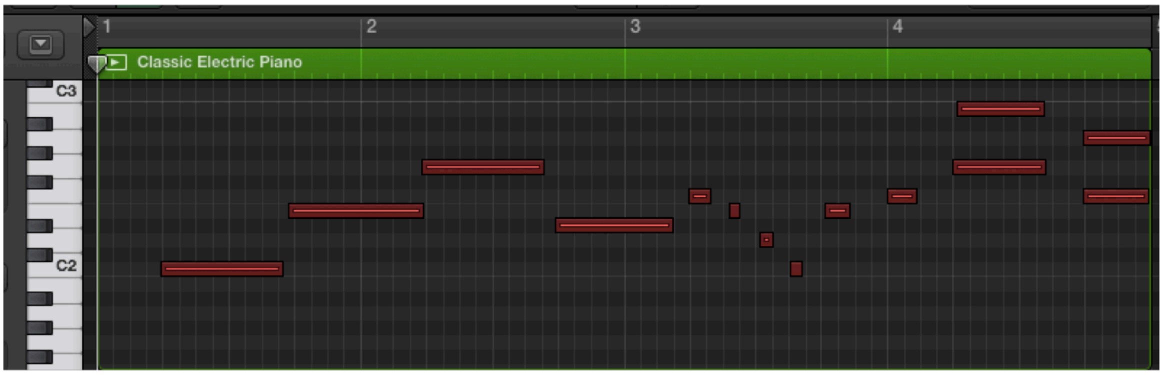


Figure 4.2.: The piano roll notation editor in Logic Pro.

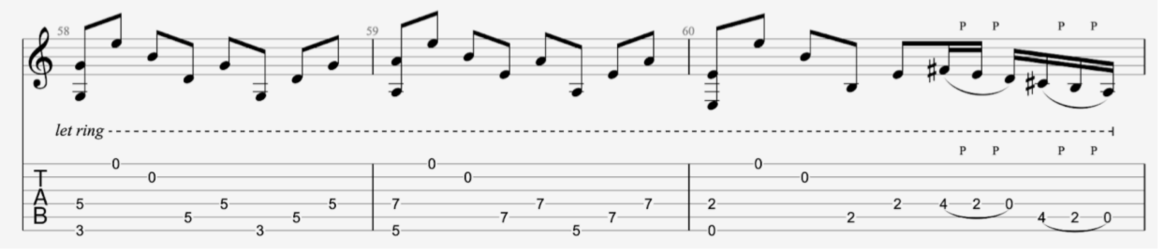
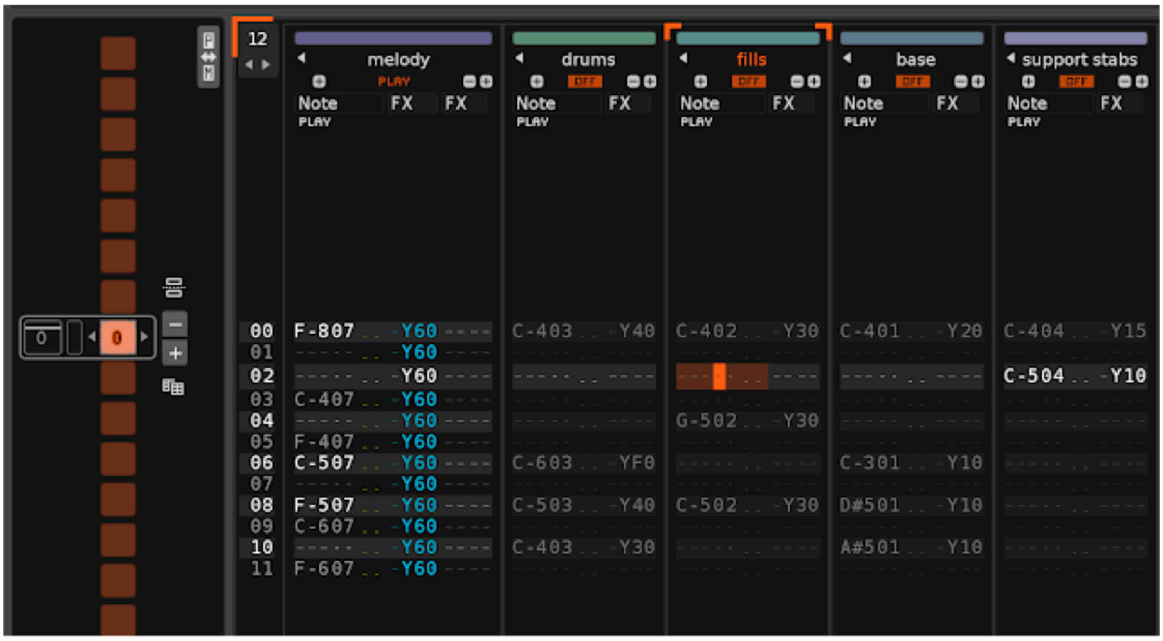


Figure 4.3.: The guitar pro tabulature notation.

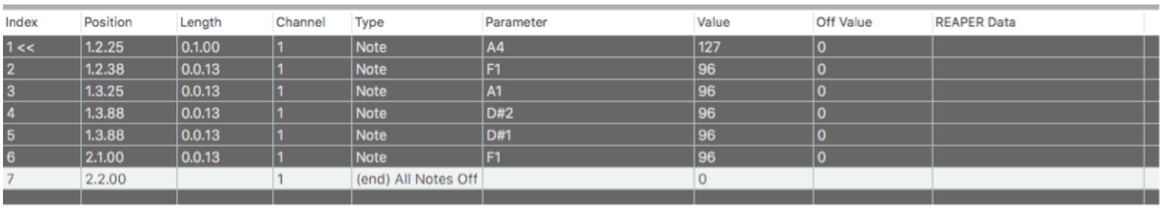
Music software has four main notations: piano roll, western score notation, step sequencer, and tracker. Piano roll (Figure 4.2) is dominant in most software. A note's pitch is represented by its on-screen y-position fixed incrementally to a discrete value, while its on-time is represented by the x-position. A note's length is represented by its on-screen width. On-time and length are generally quantised, because, in theory, the resolution is infinite.

Western score notation remains the most common notation for learning and performing western music. Score notation is supported in many sequencers, but specific notation programs (e.g Sibilies, Dorico) are required when notating more complex musical arrangements. Score notation can be more restrictive than other notations when editing and has a high literacy threshold (Nash, 2015). Tablature (Figure 4.3) based notation programs (e.g. Guitar Pro) extend traditional notation for stringed instruments such as guitar.



	melody	drums	fills	base	support stabs
00	F-807 .. -Y60 ----	C-403 .. -Y40	C-402 .. -Y30	C-401 .. -Y20	C-404 .. -Y15
01	----- .. -Y60 ----	----- .. ----	----- .. ----	----- .. ----	----- .. ----
02	----- .. -Y60 ----	----- .. ----	----- .. ----	----- .. ----	C-504 .. -Y10
03	C-407 .. -Y60 ----	----- .. ----	----- .. ----	----- .. ----	----- .. ----
04	----- .. -Y60 ----	----- .. ----	G-502 .. -Y30	----- .. ----	----- .. ----
05	F-407 .. -Y60 ----	----- .. ----	----- .. ----	----- .. ----	----- .. ----
06	C-507 .. -Y60 ----	C-603 .. -YF0	----- .. ----	C-301 .. -Y10	----- .. ----
07	----- .. -Y60 ----	----- .. ----	----- .. ----	----- .. ----	----- .. ----
08	F-507 .. -Y60 ----	C-503 .. -Y40	C-502 .. -Y30	D#501 .. -Y10	----- .. ----
09	C-607 .. -Y60 ----	----- .. ----	----- .. ----	----- .. ----	----- .. ----
10	----- .. -Y60 ----	C-403 .. -Y30	----- .. ----	A#501 .. -Y10	----- .. ----
11	F-607 .. -Y60 ----	----- .. ----	----- .. ----	----- .. ----	----- .. ----

Figure 4.4.: Renoise's tracker notation.



Index	Position	Length	Channel	Type	Parameter	Value	Off Value	REAPER Data
1 <<	1.2.25	0.1.00	1	Note	A4	127	0	
2	1.2.38	0.0.13	1	Note	F1	96	0	
3	1.3.25	0.0.13	1	Note	A1	96	0	
4	1.3.88	0.0.13	1	Note	D#2	96	0	
5	1.3.88	0.0.13	1	Note	D#1	96	0	
6	2.1.00	0.0.13	1	Note	F1	96	0	
7	2.2.00		1	(end) All Notes Off		0		

Figure 4.5.: Repear's event list.

Trackers (Figure 4.4) provide a special type of notation, with many similarities to a spreadsheet. The flow of time runs top to bottom and is evenly divided into rows, which can contain a range of information from simple notes to program changes. Other than Renoise, few such applications are popular. Logic Pro and Reaper offer a similar alternative notation, in which events can be edited in a rudimentary list (Figure 4.5).

The step sequencer (Figure 4.6) is a unique notational interface that is provided in a number of DAWs and took its influence from early digital electronic instruments. Steps sequencers are similar to piano roll editors, but partition the available notes into discrete rows and divide *time* into fixed steps - therefore resembling a grid. For each position (step) in the grid, the note is usually either on or off. The notation has a low barrier to entry, but a shallow ceiling compared to other types of notation.

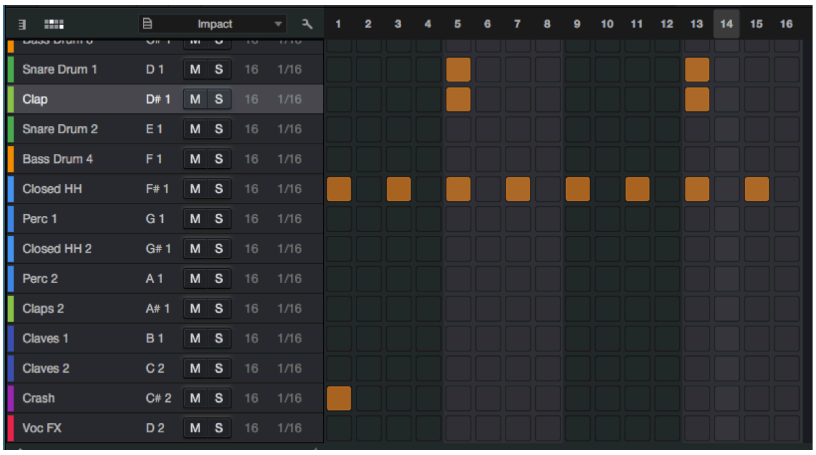


Figure 4.6.: Studio one's step sequencer.

4.1.2. Workflows

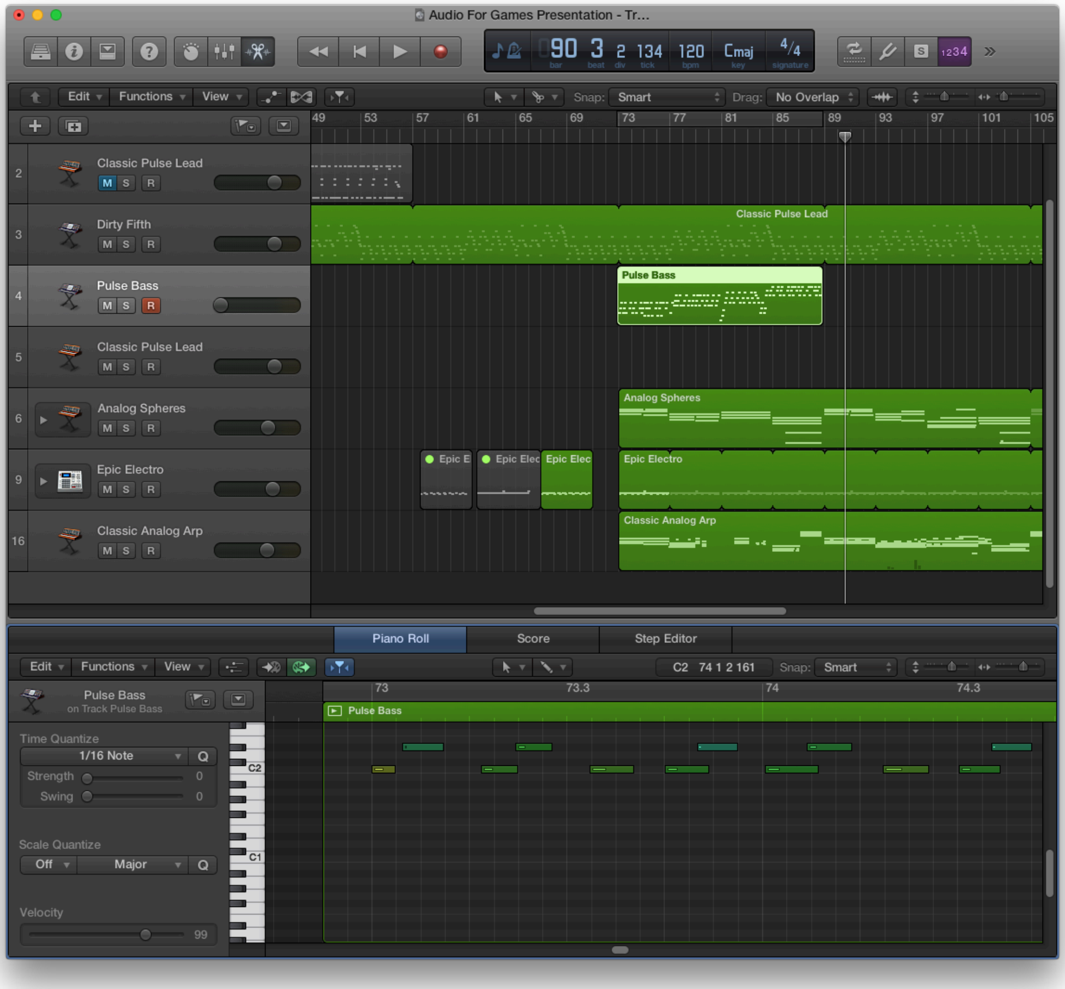
A workflow is generally defined as a series of interaction events that produces (or edits) a musical sequence when using software [1]. The way in which composers interact with different music software (*workflows*) are generally similar, supported though broadly homogeneous design metaphors (see Figure 4.7). Notably, clips of arbitrary length music are arranged on a timeline, in which time flows from left to right and tracks from instruments proceed vertically. Score editors are similar in this respect, although the spacing of musical markings is not linear. The clip's content is edited by tools and notations or recorded in real-time through a MIDI instrument. Most of the software discussed also supports clips of recorded audio, although the only clips we consider are those that process sequenced notes (i.e. MIDI). Specific score editing software (Sibelius) are mostly suited for engraving music (i.e. preparing for publication).

FL Studio differs slightly in that musical patterns are created primarily in a step-sequencer with the patterns then painted on a timeline (Figure 4.8). Editing the original pattern will propagate changes through all instances of it on the timeline, creating *hidden dependencies*, whereby changing one variable inadvertently updates another. Patterns can be created without fixing them to a timeline, encouraging *provisionality*[2].

Bitwig Studio and Ableton Live are similar and allow the development of clips independent of a timeline. These clips can be triggered and sequenced in real-time or placed back on the timeline. Studio One has a novel scratchpad feature for adding virtual timelines for experiments

[1] Workflows also exist in non-digitally mediated composition, i.e. on paper notation.

[2] *"Is it possible to sketch things out and play with ideas without being too precise about the exact result?"*, from Nash (2015).



Snap: Smart

Drag: No Overlap

Time Quantize 1/16 Note

Strength 0

Swing 0

Scale Quantize Off Major

Velocity 99

Figure 4.7.: Logic Pro X's timeline.

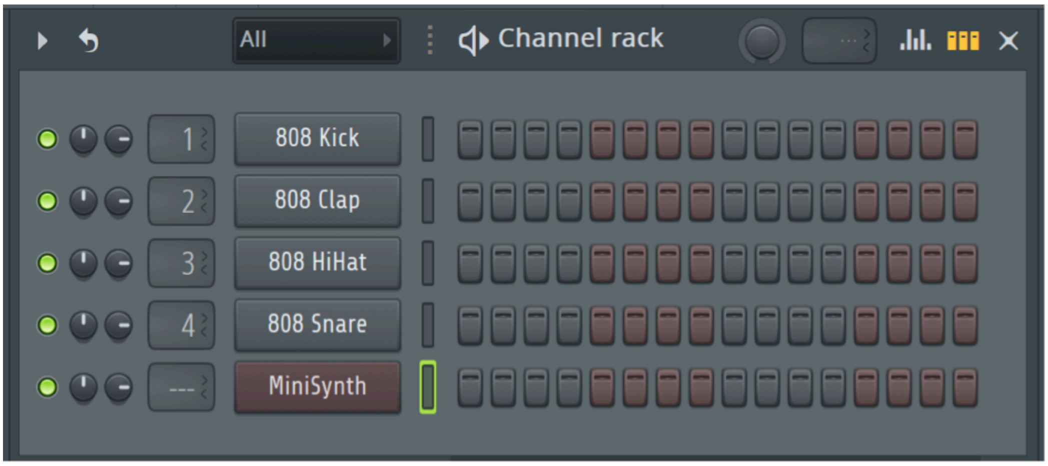


Figure 4.8.: FL Studio's step sequencer.

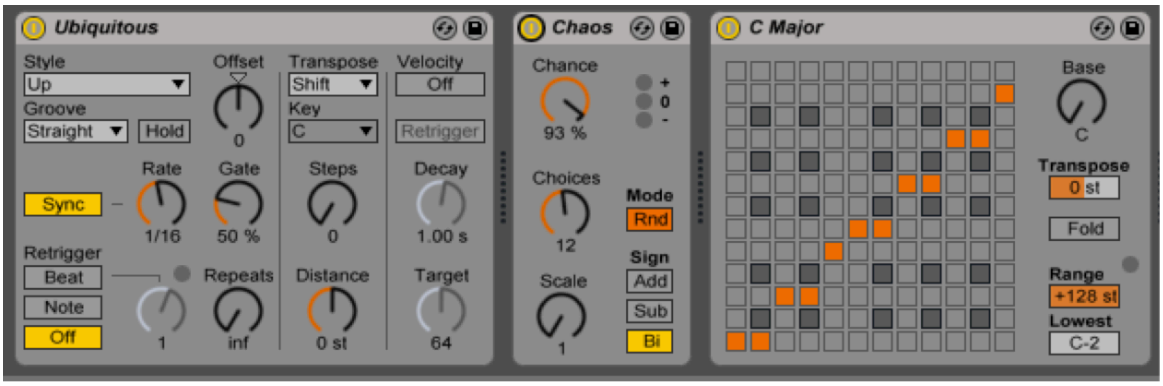
Name	Input	Summary	Is deterministic
Chords	Single note	For a single note, this process outputs a chord, creating a one-to-many mapping.	Yes
Transposer	Single note	Transposes all incoming notes by a given value.	Yes
Arpeggiator	Sequence	Sequences the input notes into a given pattern, with various parameters controlling the order and timing of events.	Yes/No
Randomiser	Sequence	Randomises the ordering of a sequence of notes, or applies randomisation to various elements (pitch and time).	No
Quantisers	Single note	Ensures the input note is within a given scale, transposing the value to the nearest scale division.	Yes
Repeater	Single note	Sequences an input note into a repeating pattern so that it echoes.	Yes/No

Table 4.2.: A summary of the various computer-generated effects found in common music software.

with different arrangements without affecting the master timeline. The user is free to copy clips between timelines.

4.1.3. Plug-ins

A range of MIDI effects are found within music sequencer applications, which generally perform algorithms on a sequence of input notes. The effects can be deterministic (e.g. transpose) or stochastic (e.g. randomiser). Table 4.2 summarises these common processes and Appendix C further discusses them individually with specific examples. The majority of these effects work in real-time synthesising their resultant effect during playback. However, FL studio's arpeggiator transfers the output back into the note editor. The effect is destructive once the user hits *'confirm'* and the initial notes and settings are lost. The majority of these effects are simplistic in nature, with more noteworthy processes summarised at the end of this section.



Ubiquitous
Style: Up
Groove: Straight
Offset: 0
Transpose: Shift
Key: C
Velocity: Off
Rate: 1/16
Gate: 50 %
Steps: 0
Decay: 1.00 s
Retrigger: Off
Repeats: inf
Distance: 0 st
Target: 64

Chaos
Chance: 93 %
Choices: 12
Scale: 1
Mode: Rnd
Sign: Bi

C Major
Base: C
Transpose: 0 st
Range: +128 st
Lowest: C-2

Figure 4.9.: Three effects (the arpeggiator, randomiser and scale quantiser) chained together in Live,

Chaining effects

Certain applications support the chaining together of effects. For example, in Figure 4.9, an arpeggiator, randomiser and scale quantiser in Live have been concatenated. The arpeggiator provides a repeating pattern that is then slightly randomised. To prevent unnecessary stochasticism, the scale quantiser is used to ensure the output stays in the major scale (and also in key).

4.1.4. Specific Effects

As well as general plug-ins/effects, some novel and noteworthy processes are also available. Ableton Live can be integrated with Max for Live, which gives access to a low-level API that can be customised to provide a plethora of extensions, including generative applications. However, Max for Live is a visual programming language (notation) that requires expert knowledge to use (Nash, 2015). Max patches can access low-level APIs for close integration with Live, thus allowing anybody to utilise the power of Live, permitting all sorts of generative and experimental applications. Max for Live is a domain-specific language for Ableton and is incompatible with other mainstream software. Max is further explored in section 4.3.2.

FL Studio contains a complex riff machine, with 8 steps for generating a riff from scratch (Figure 4.10). At each stage, the output is rendered directly back into the clip, so can be evaluated visually and audibly.

The script editor (Figure 4.11) in Logic Pro X offers a range of options for complex editing of note events in real-time, but the user needs to be familiar with programming notation, thus alienating traditional composers without this domain-specific knowledge. Furthermore, building

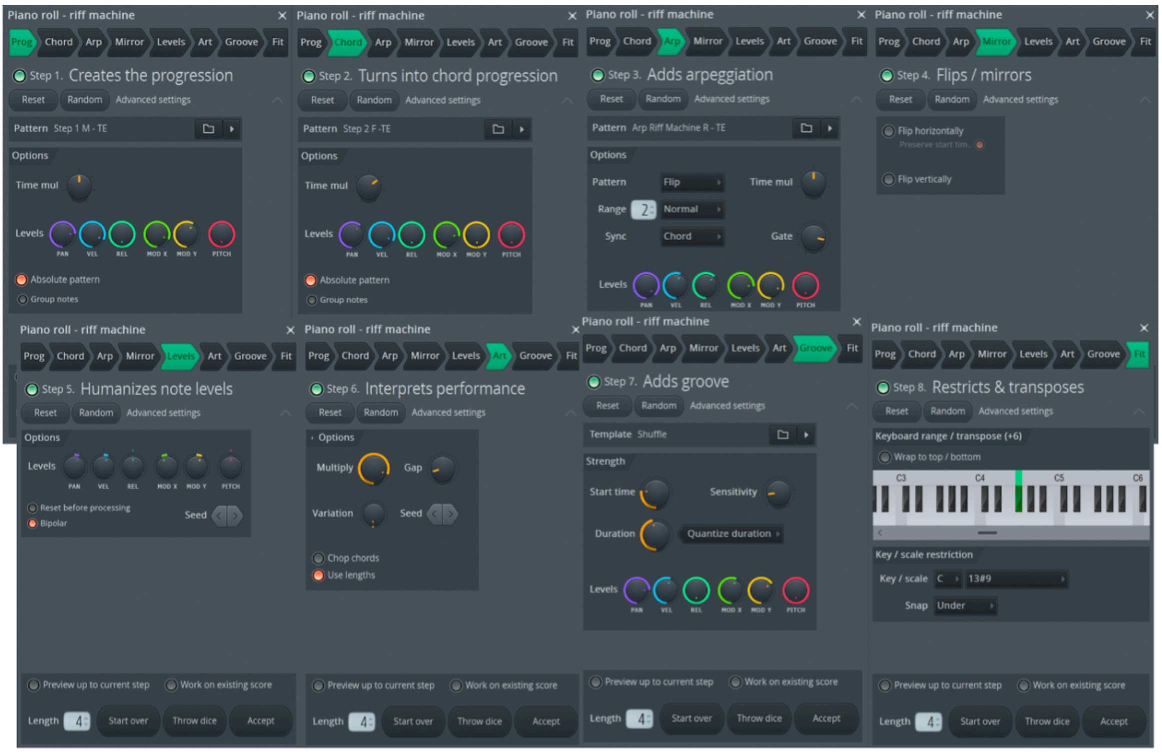


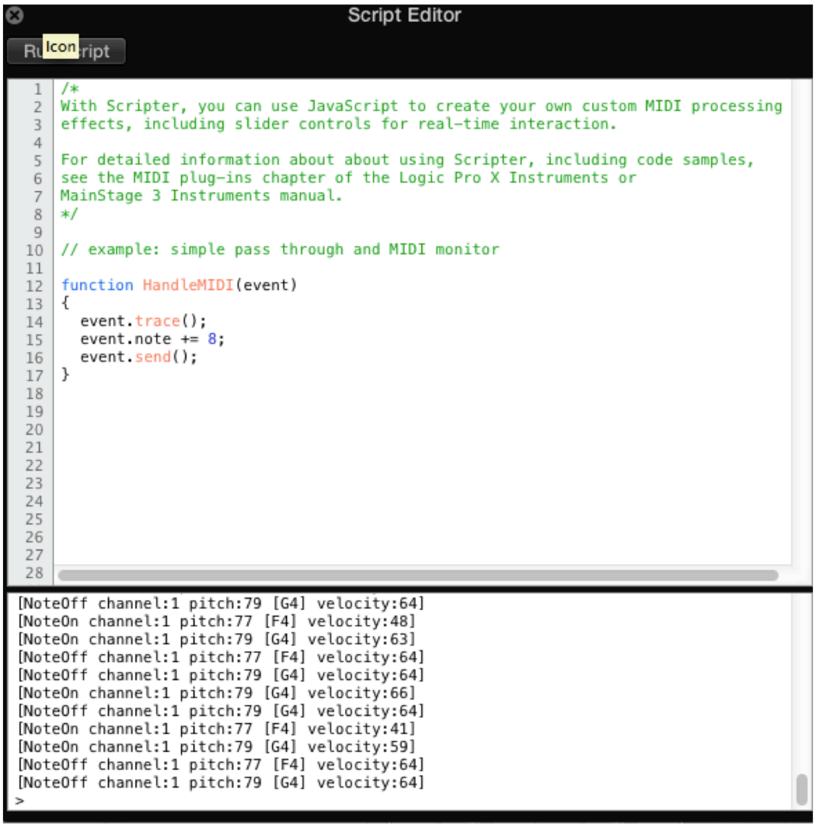
Figure 4.10.: FL Studio's complex riff generator.

more complex effects, such as the inbuilt 4-step algorithmic drum sequencer example, creates an unwieldy interface with 64 sliders. The main limitation is the inability to generate and store sequences, as events are computed in real-time while auditioning the piece. In theory, the user could set up complex MIDI routing, to route outgoing messages to another track for recording, however, this is needlessly complicated.

Numerology has inbuilt generative effects and a simple interface for *'evolving'* variants of a user-defined sequence, such as swapping two pitches, transposing notes, or modifying gates/velocities (Figure 4.12). Several rules can be chained up with a discrete probability for each. These can be evolved at a given point, such as at every beat or every bar.

Reason includes the *BeatMap* effect, which constructs an algorithmic drum machine by synthesising the contents of a metaphorical map, with each drum part relating to a certain geological feature (Figure 4.13). The user is encouraged to *'scroll'* the map, searching for drum patterns.

When exploring computer-generated music, it is important to consider how the interactive processes and workflow are modelled. The closest example of a computer-generated technique being integrated inside an existing workflow is Logic Pro X's smart drummer, which creates

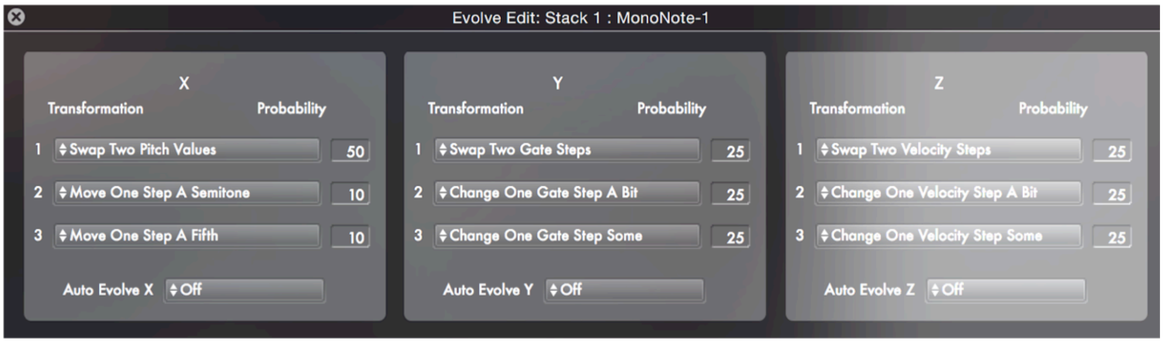


```
/*
With Scripter, you can use JavaScript to create your own custom MIDI processing
effects, including slider controls for real-time interaction.

For detailed information about about using Scripter, including code samples,
see the MIDI plug-ins chapter of the Logic Pro X Instruments or
MainStage 3 Instruments manual.
*/

// example: simple pass through and MIDI monitor

function HandleMIDI(event)
{
  event.trace();
  event.note += 8;
  event.send();
}
```

```
[NoteOff channel:1 pitch:79 [G4] velocity:64]
[NoteOn channel:1 pitch:77 [F4] velocity:48]
[NoteOn channel:1 pitch:79 [G4] velocity:63]
[NoteOff channel:1 pitch:77 [F4] velocity:64]
[NoteOff channel:1 pitch:79 [G4] velocity:64]
[NoteOn channel:1 pitch:79 [G4] velocity:66]
[NoteOff channel:1 pitch:79 [G4] velocity:64]
[NoteOn channel:1 pitch:77 [F4] velocity:41]
[NoteOn channel:1 pitch:79 [G4] velocity:59]
[NoteOff channel:1 pitch:77 [F4] velocity:64]
[NoteOff channel:1 pitch:79 [G4] velocity:64]
>
```

Figure 4.11.: Logic Pro's script plug-in.

'computer-generated' clips. Each clip has parameters, which are evaluated when moved, to create music content. The content will remain the same until a parameter changes, so the piece can be repeatedly auditioned with identical and replicable results. The notes themselves are not specified, but instead, the parameters and techniques used to create them are (Figure 4.14). For example, the amount and complexity of the hi-hat pattern in each section is controlled by a single slider. The overall pattern for each part (kick, snare, hats, toms, cymbals, and percussion) is controlled by two contrasting parameters on a 2D grid, with simple-complex on one axis and



X

	Transformation	Probability
1	Swap Two Pitch Values	50
2	Move One Step A Semitone	10
3	Move One Step A Fifth	10

Auto Evolve X: Off

Y

	Transformation	Probability
1	Swap Two Gate Steps	25
2	Change One Gate Step A Bit	25
3	Change One Gate Step Some	25

Auto Evolve Y: Off

Z

	Transformation	Probability
1	Swap Two Velocity Steps	25
2	Change One Velocity Step A Bit	25
3	Change One Velocity Step Some	25

Auto Evolve Z: Off

Figure 4.12.: Numerolgy's evolver settings window.

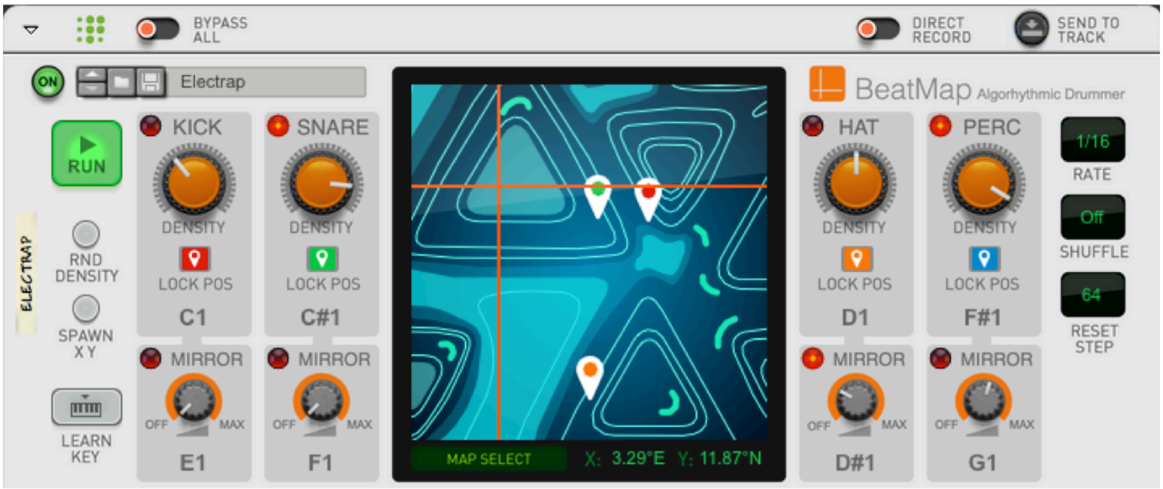


Figure 4.13.: Reason's beat map.

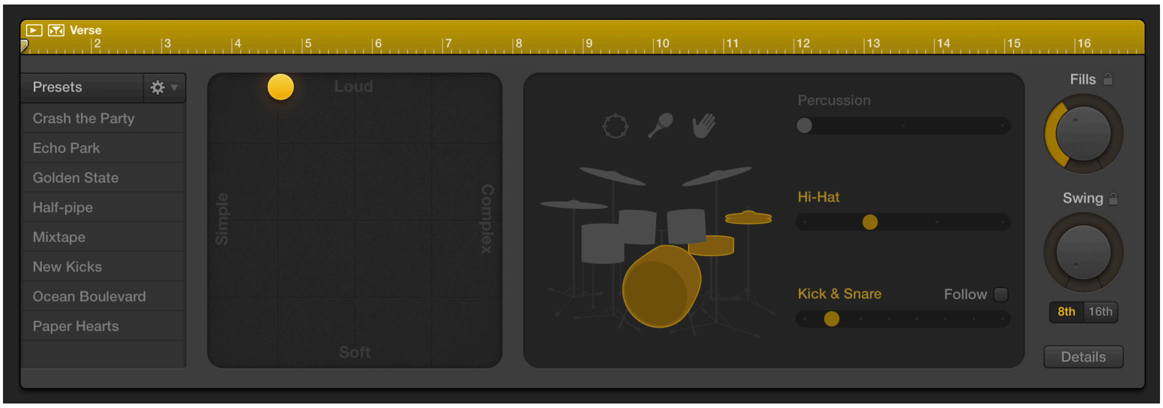


Figure 4.14.: Logic Pro X's Smart Drummer plug-in

loud-soft on the other (left in Figure 4.14). Upon loading the session from disk, the previous pattern will be recalled, but only moving a parameter results in iteration. This can produce errors, as moving a slider causes the previous content to be lost, although the 'undo' command would still apply. A smart drummer section is defined as a phrase N-number bar(s) in length with a series of parameters. An entire piece can therefore be made up of sections with differing parameters.

Importantly, the system is controlled with a design metaphor similar to that of a plug-in, which would already be familiar to the user. Secondly, the parameters themselves are labelled with words that closely map to the features of the music. Visualisation is used to model the various parts of the kits, showing exactly what the user is controlling or whether or not the part is active. Finally, this approach distinguishes between deterministic drum parts and those with stochastic effects applied to them (Figure 4.15).

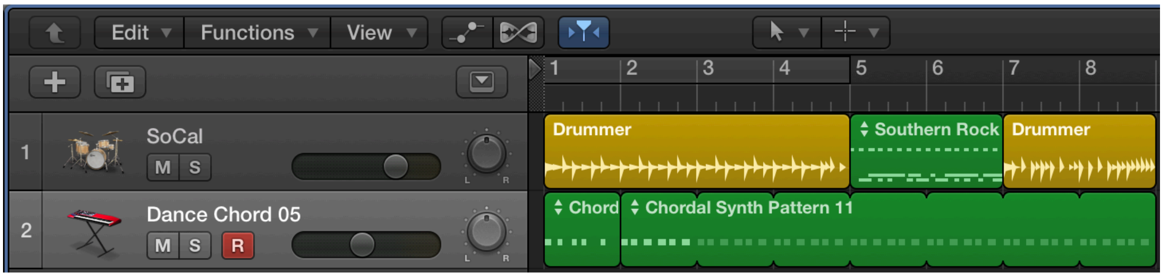


Figure 4.15.: Logic Pro X's timeline showing clearly the distinction between computer-generated and normal parts.

4.1.5. Computer Assisted Musicology, Editing, and Analysis

Sequencing software contains techniques that use the computer as an editing assistant. The computer can play several roles, including increasing editing efficiency, applying composition routines, and analysing the user's work for structural elements and to suggest scales, chords or loops. This suggests that the computer is already an active participant in some elements of music, digital creativity, and composition, but these techniques fundamentally remain optional - which is an important consideration for the design of future end-user CGM systems.

Editing

Computer-based techniques for editing are loosely categorised as tools for efficiency or creativity. Generally, tools that increase efficiency, simply automate tasks that could be done manually, albeit more slowly, or with an increased chance of error. Many software pieces have a transpose tool (different from the transpose plug-in discussed previously) that will transpose a selection of notes (Figure 4.16). Dragging notes up or down to transpose manually is error-prone and can be physically difficult to align to a specific division. Note duration tools modify selected notes by either a fixed amount (i.e. double their length) or through increasing by a percentage, highlighting how tedious such a task would be manually. More advanced tools, such as Digital Performer's Humanizer (Figure 4.17), can apply a large range of processes for modifying onset, velocity, duration, tempo, and pitch mapping.

Similarly, tools for creativity can also be seen as tools for efficiency, for example, adding a note on every beat, or applying retrograde inversions and note rotations. Whereas many randomisation techniques, previously discussed, can be applied to notes in the editor which have the potential to elicit creativity in the user (through novelty).

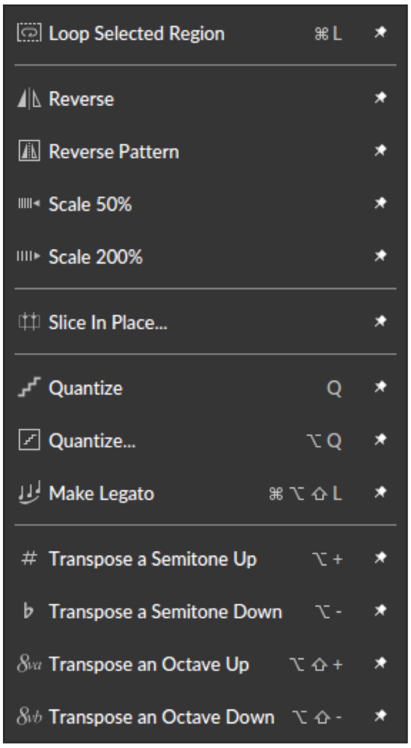
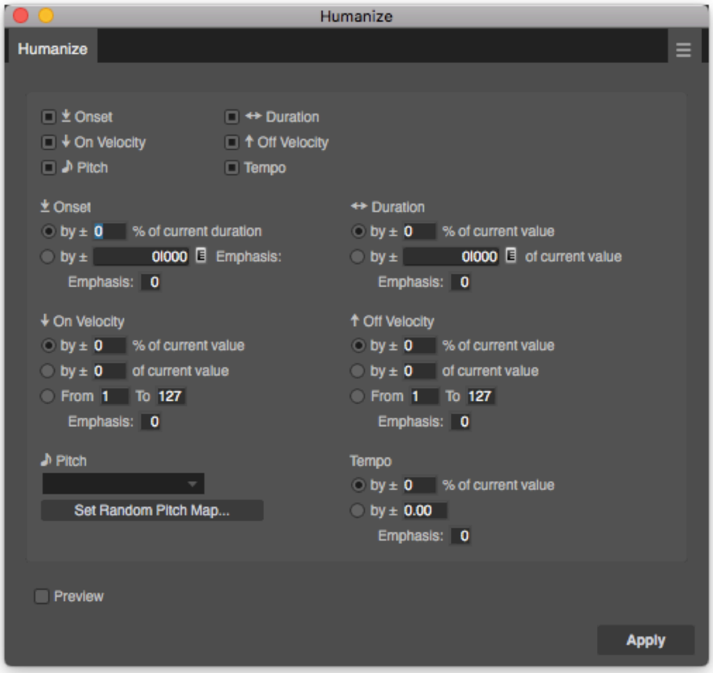


Figure 4.16.: Bitwig's note editing menu.


Humanize

Humanize

☒ Onset ☒ Duration
☒ On Velocity ☒ Off Velocity
☒ Pitch ☒ Tempo

Onset
● by ± 0 % of current duration
○ by ± 01000 Emphasis:
Emphasis: 0

Duration
● by ± 0 % of current value
○ by ± 01000 of current value
Emphasis: 0

On Velocity
● by ± 0 % of current value
○ by ± 0 of current value
○ From 1 To 127
Emphasis: 0

Off Velocity
● by ± 0 % of current value
○ by ± 0 of current value
○ From 1 To 127
Emphasis: 0

Pitch
Set Random Pitch Map...

Tempo
● by ± 0 % of current value
○ by ± 0.00
Emphasis: 0

□ Preview

Apply

Figure 4.17.: Digital Performer's humanizer.

Music Theory

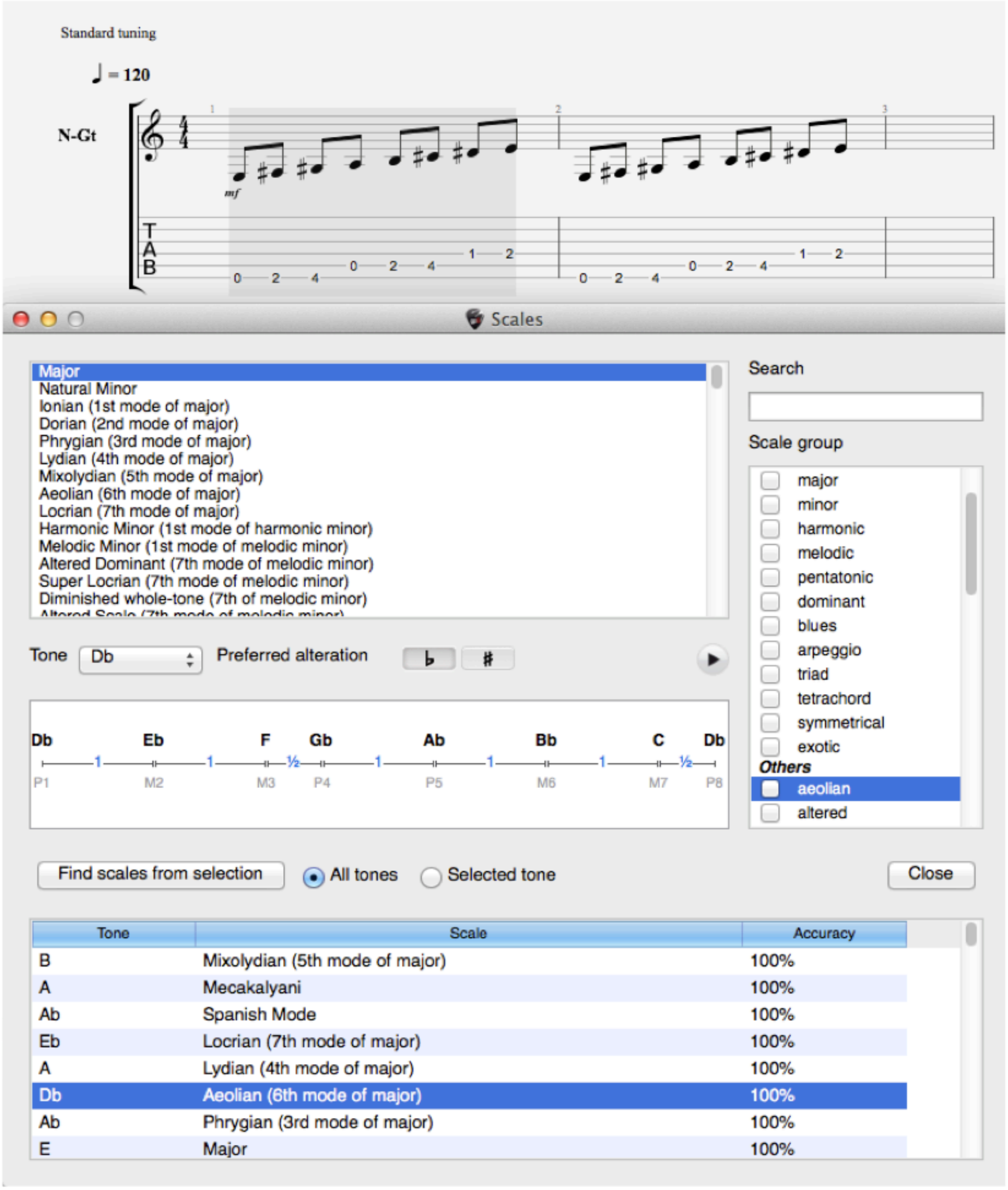
Some sequencers contain tools that aid musical theory by, for example, analysing notes to compute the likely chord or scale (Guitar Pro and Logic Pro). Other tools allow chords to be inserted from a predefined library. It is unclear whether this is used by novices experimenting with unusual or complex chords, or by experienced practitioners as workflow improvements. It can be argued that such features promote the use of chords based on how a user perceives their sound rather than on their logical relationship, or based upon the rules of harmony and key. Studio One can automatically detect the key of a MIDI sequence, helping those with limited music theory knowledge.

Cubase has a feature for automatically adding chord patterns to a piece. A user can place chords on a timeline or map them to keyboard keys. When triggered in real-time, these features enable rapid experimentation with chord types.

Guitar Pro has a library of built-in scales that can be browsed, with a feature that suggests the best scale for a given selection of notes (Figure 4.18), although it remains unclear how the best scale is determined. Once selected, a scale can be applied to a virtual fretboard viewer (Figure 4.19), allowing the user to more easily select notes in that scale. FL Studio allows a scale pattern to be added to the piano editor, so this can be visualised and auditioned.

Logic Pro X contains a vast metadata tagged loop library that provides compatibility scores, rating a sample with those already in the piece, allowing appropriately-composed material to be modified, arranged and experimented with in a narrow search space. Many systems and third party sample pack libraries have curated content, i.e. content that sounds *good* together. The user is free to arrange the content and order their playback. Novation's Launchpad app (Ampify-Music, 2020) encourages the user to work within a predefined *soundpack*. Although perhaps limited in scope, this offers some protection against making music with jarring musical clashes, whilst remaining sensitive to the style/genre in which the content pack is intended (i.e. rock or pop).

The chord track feature in Studio One (Figure 4.20) keeps track of harmonic structure and can be populated either by analysing existing tracks or manually. Other tracks can be set to follow the chord track. Updating the chord track will propagate changes to any tracks following this, therefore offloading detailed compositional elements and editing to the computer. Similarly, the online composition platform, HookTheory (HookPad, 2020), encourages composition through the use of chord blocks, with colour coding helping users to stay in key (Figure 4.21).



Major
Natural Minor
Ionian (1st mode of major)
Dorian (2nd mode of major)
Phrygian (3rd mode of major)
Lydian (4th mode of major)
Mixolydian (5th mode of major)
Aeolian (6th mode of major)
Locrian (7th mode of major)
Harmonic Minor (1st mode of harmonic minor)
Melodic Minor (1st mode of melodic minor)
Altered Dominant (7th mode of melodic minor)
Super Locrian (7th mode of melodic minor)
Diminished whole-tone (7th of melodic minor)
Altered Scale (7th mode of melodic minor)

Search

Scale group

□ major
□ minor
□ harmonic
□ melodic
□ pentatonic
□ dominant
□ blues
□ arpeggio
□ triad
□ tetrachord
□ symmetrical
□ exotic

Others

□ aeolian
□ altered

Tone: Db

Preferred alteration: ♭ ♯

Find scales from selection ● All tones ○ Selected tone Close

Tone	Scale	Accuracy
B	Mixolydian (5th mode of major)	100%
A	Mecakalyani	100%
Ab	Spanish Mode	100%
Eb	Locrian (7th mode of major)	100%
A	Lydian (4th mode of major)	100%
Db	Aeolian (6th mode of major)	100%
Ab	Phrygian (3rd mode of major)	100%
E	Major	100%

Figure 4.18.: Guitar Pro's scale analyser.

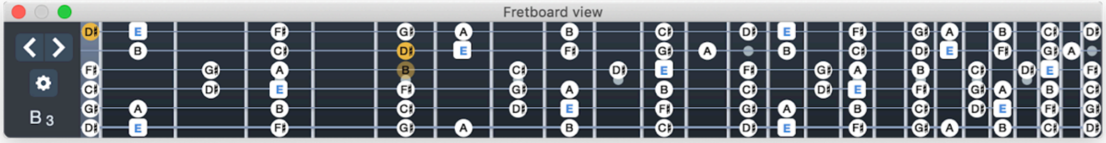
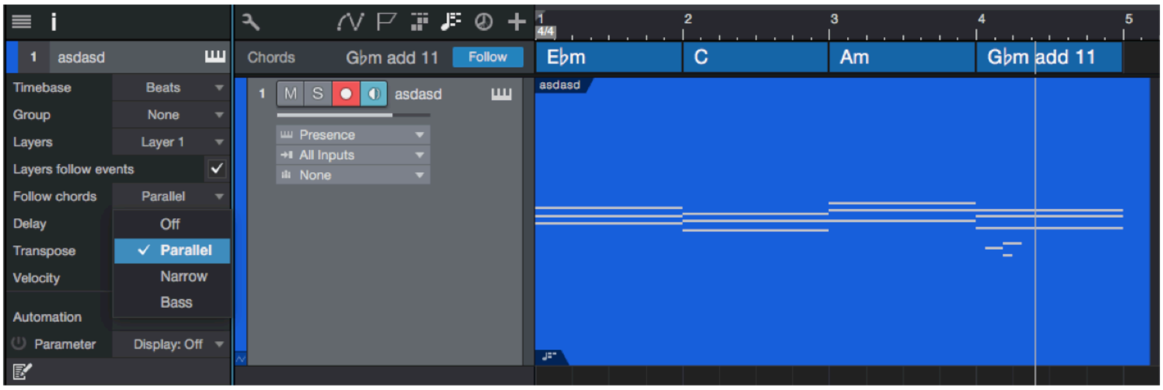
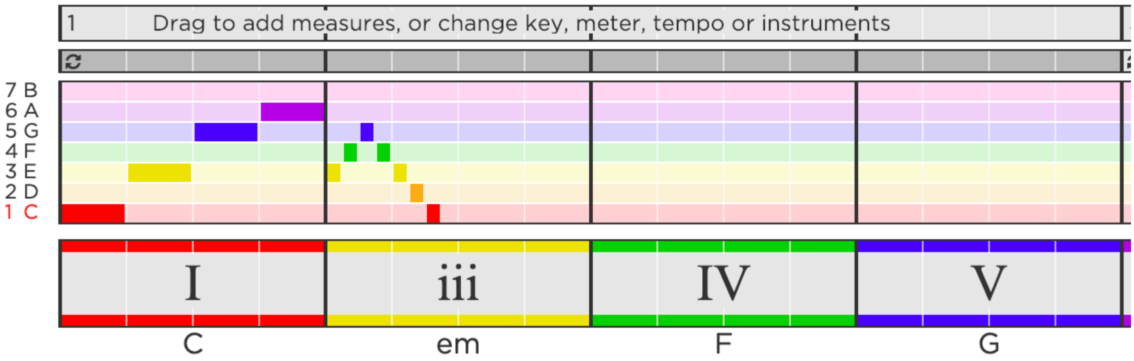


Figure 4.19.: Guitar Pro's fretboard viewer.



Timebase: Beats
Group: None
Layers: Layer 1
Layers follow events ✓
Follow chords: Parallel

Figure 4.20.: Studio One's chord track.



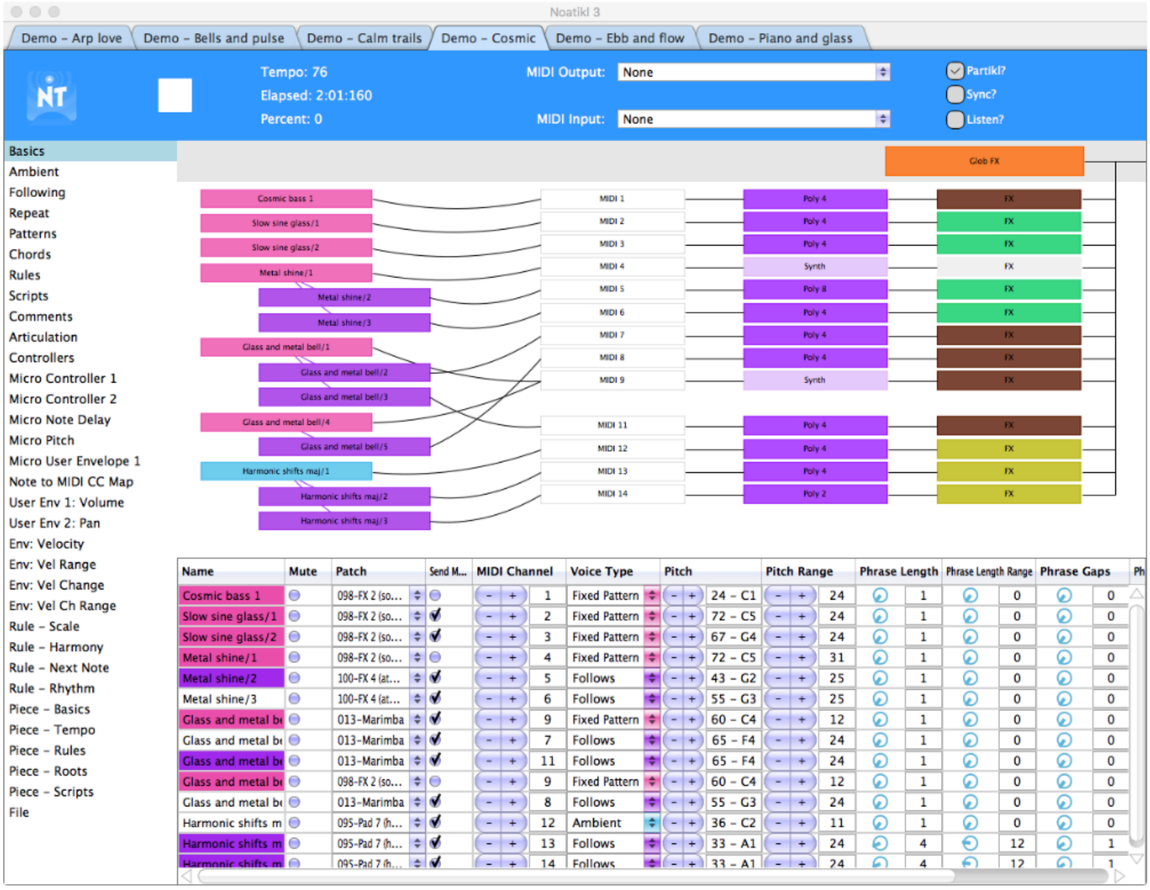
Drag to add measures, or change key, meter, tempo or instruments

Figure 4.21.: Hook Theory's colour coded interface.

4.2. Existing Computer-Generated Music Systems

This section looks at generative music systems that are informal music sequencers. Unlike general computer music systems, the systems discussed here employ more than one technique.

An early generative music system, famously used by avant-garde composer Brian Eno, is Koan (Intermorphic, 2018b; SSEYO, 1995) by SSEYO. Although the original program has been discontinued, and little technical or academic information regarding it is available, the software has been absorbed by later systems[3]. Koan, used by Eno for several installations (Föllmer, 2005), included a playback feature enabling composers to prepare a generative music system for distribution to listeners who could choose from numerous permutations. Albums released in this way give listeners a unique experience Intermorphic (2018a). In collaboration with Chilvers, Eno has created a range of generative music systems and applications (Eno and Chilvers, 2017).



Tempo: 76
Elapsed: 2:01:160
Percent: 0

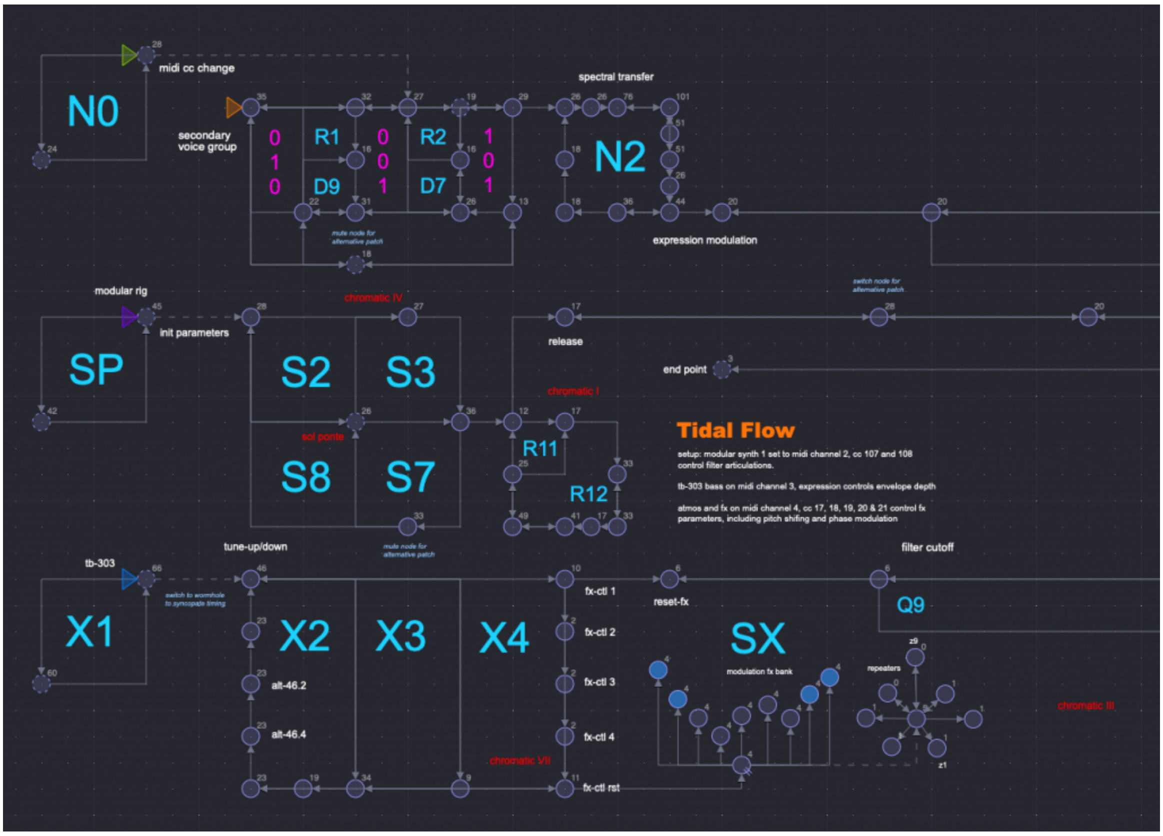
MIDI Output: None
MIDI Input: None

☑ Partikl?
○ Sync?
○ Listen?

Name	Mute	Patch	Send M...	MIDI Channel	Voice Type	Pitch	Pitch Range	Phrase Length	Phrase Length Range	Phrase Gaps	Ph
Cosmic bass 1	○	098-FX 2 (so...	○	1	Fixed Pattern	24 – C1	24	1	0	0	
Slow sine glass/1	○	098-FX 2 (so...	✓	2	Fixed Pattern	72 – C5	24	1	0	0	
Slow sine glass/2	○	098-FX 2 (so...	✓	3	Fixed Pattern	67 – G4	24	1	0	0	
Metal shine/1	○	098-FX 2 (so...	○	4	Fixed Pattern	72 – C5	31	1	0	0	
Metal shine/2	○	100-FX 4 (at...	✓	5	Follows	43 – G2	25	1	0	0	
Metal shine/3	○	100-FX 4 (at...	✓	6	Follows	55 – G3	25	1	0	0	
Glass and metal be	○	013-Marimba	✓	9	Fixed Pattern	60 – C4	12	1	0	0	
Glass and metal be	○	013-Marimba	✓	7	Follows	65 – F4	24	1	0	0	
Glass and metal be	○	013-Marimba	✓	11	Follows	65 – F4	24	1	0	0	
Glass and metal be	○	098-FX 2 (so...	○	9	Fixed Pattern	60 – C4	12	1	0	0	
Glass and metal be	○	013-Marimba	✓	8	Follows	55 – G3	24	1	0	0	
Harmonic shifts m	○	095-Pad 7 (h...	✓	12	Ambient	36 – C2	11	1	0	0	
Harmonic shifts m	○	095-Pad 7 (h...	✓	13	Follows	33 – A1	24	4	12	1	
Harmonic shifts m	○	095-Pad 7 (h...	✓	14	Follows	33 – A1	24	4	12	1	

Figure 4.22.: Noatikl 3 software. *Image used with permission of Intermorphic (Intermorphic, 2021).*

[3] A timeline of the specifics of Koan and its successors is given by Intermorphic (2018a).



Tidal Flow

setup: modular synth 1 set to midi channel 2, cc 107 and 108 control filter articulations.

tb-303 bass on midi channel 3, expression controls envelope depth

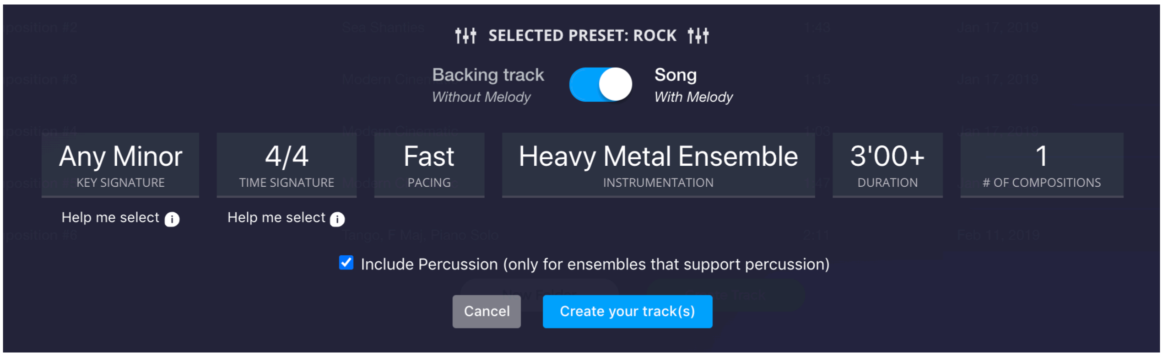
atmos and fx on midi channel 4, cc 17, 18, 19, 20 & 21 control fx parameters, including pitch shifing and phase modulation

Figure 4.23.: Nodal software. *Image used with permission of Nodal music (Nodal, 2021).*

Noatikl (Brown and Kerr, 2009) is considered a development of Koan (Figure 4.22). The environment is made up of objects that are patched into sound outputs. Parameters control the synthesis of MIDI events, including pitch range, phrase length, and the note/rest ratio. Noatikl can be used as a plug-in, so its output can be piped directly into existing music sequencers, however it has limited capacity for creating more structured forms of composition and is perhaps best-suited to creating ambient styles.

Intermorphic (2021) continue to develop accessible generative music systems for both desktop and mobile through Wotja. The systems are primarily patch-based and offer little control over the musical structure. Minimal academic work has been published on these systems.

FractMus (Diaz-Jerez, 2011) provides a graphical interface for exploring fractals as tools for creating musical structure. The author of the software, Peters (2010), notes that *"you are the composer, FractMus will create no masterpiece for you, nor it was designed for that. Think of it as a tool which gives you raw material that you can later use in your compositions".* Peters (2010) also notes that output from FractMus should be placed into a score-editing program for playback, visualization and fine editing. Such a system provides a simple method for exploring the power



SELECTED PRESET: ROCK

Backing track *Without Melody* / Song *With Melody*

Any Minor KEY SIGNATURE

4/4 TIME SIGNATURE

Fast PACING

Heavy Metal Ensemble INSTRUMENTATION

3'00+ DURATION

1 # OF COMPOSITIONS

☑ Include Percussion (only for ensembles that support percussion)

Figure 4.24.: Aiva's song configuration panel.

of generative music but has a paucity of meaningful ways to create a fully-realized composition within the system. Similarly, Tunesmithy (Bellingham, Holland, and Mulholland, 2014a; Walker, 2008) provides a graphical interface for the exploration of fractals.

Nodal (McCormack *et al.*, 2007; Nodal, 2021) is a generative system (Figure 4.23) that creates music through a user-defined network of objects, consisting of nodes (musical events) and edges (connections between events). Nodal can be integrated with existing sequencers through a plug-in interface. Signal flow in Nodal allows both sequential and parallel paths through the network. No direct order is enforced, so signals can flow left/right or up/down without restriction, unlike Max, which has a strict right-left, up-down execution order. Control flow is difficult to predict, as many conditions are only revealed once the user inspects the individual elements. Unlike other systems, Nodal offers finite control over events and flow, and also provides an effective notation for a generative system, which other stochastic systems often fundamentally fail to achieve.

The Continuator, developed by Pachet (2003), is a prominent interactive generative music system that works in unison with a composer. It uses Markov models to learn musical structure either through existing material, or in real-time from a performer, and reproduces music in either a standalone mode or in collaboration with the user. Intended for performance rather than traditional composition, the system shows that complex Markov models for composition can be made accessible to musicians who lack domain-specific knowledge.

4.2.1. Deep Learning

Cutting-edge machine learning technologies are often presented as tools capable of composing human-esque music autonomously with minimal direction from humans. While overcoming some of the limitations of computer-generated music, they mostly focus on replacing the

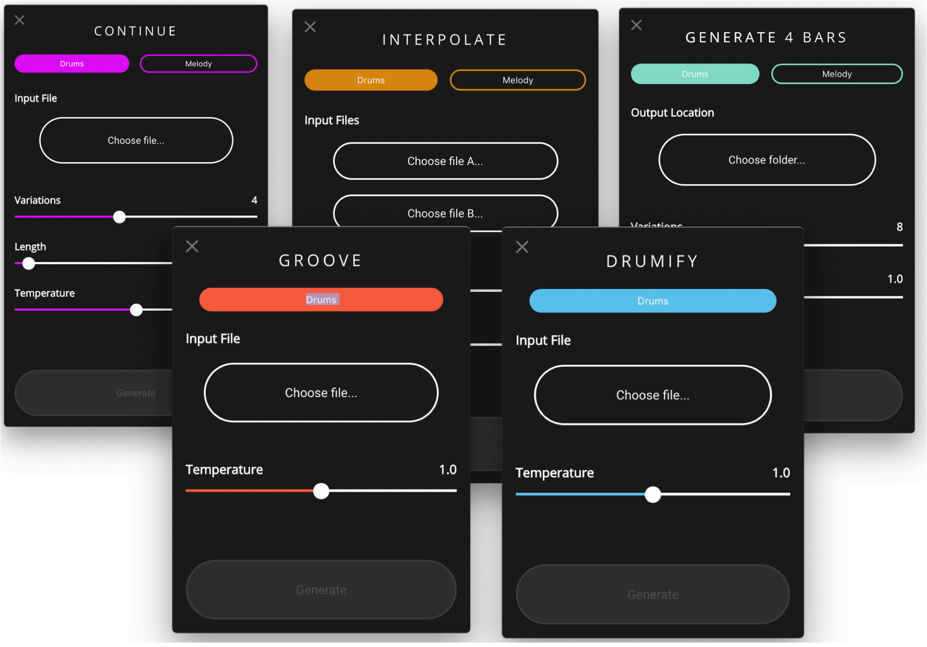


Figure 4.25.: Magneta Studio's plug-in collection.

composer altogether. Most of the systems discussed here are commercial, therefore little, if any, technical information about them is publicly available.

Jukedeck (Langkjær-Bain, 2018) is an AI system that can generate music in styles that sync up to film. It requires little or no input from the user other than configuring a few parameters, such as genre and tempo. Similar systems include AmperMusic (2020), in which tracks can be created and features and sound assets can be customised; and Melodrive (Collins, 2018), an adaptive music generation platform, specifically built to run inside a video game engine, generating music in real-time using deep learning models.

Aiva (Zulić, 2019) is another deep learning system and has a wider range of customisation options (Figure 4.24), including the ability to upload an influence track that can create music in the style of the upload. Such systems are undoubtedly useful for those with little or no experience in composition, but perhaps not for experienced practitioners. Recent updates to the software permit the user to regenerate specific sections of the pieces and apply manual edits.

The Magneta Studio (Roberts *et al.*, 2019) project is a collection of music generation tools that run either as standalone applications or are integrated within Ableton Live (Figure 4.25). Significantly, the software can read existing MIDI clips on the timeline and generate new ones, as enabled through a complex set of tools built around the Max for Live API. Although the authors assert that it would not be possible to achieve similar results in other music software through standard plug-in architecture, it does however demonstrate that it is possible to integrate complex computer-generated processes in an existing music composition workflow. The authors note that little deep learning technology has made its way into the hands of musicians, attributing this to the fact that applied creativity and deep learning research speak different languages, both literally and figuratively. In evaluating the software, users did not find the generative plug-ins that useful for creating *'desirable'* music but reported that they made them feel more productive in their creative process.

4.3. Music Programming Systems

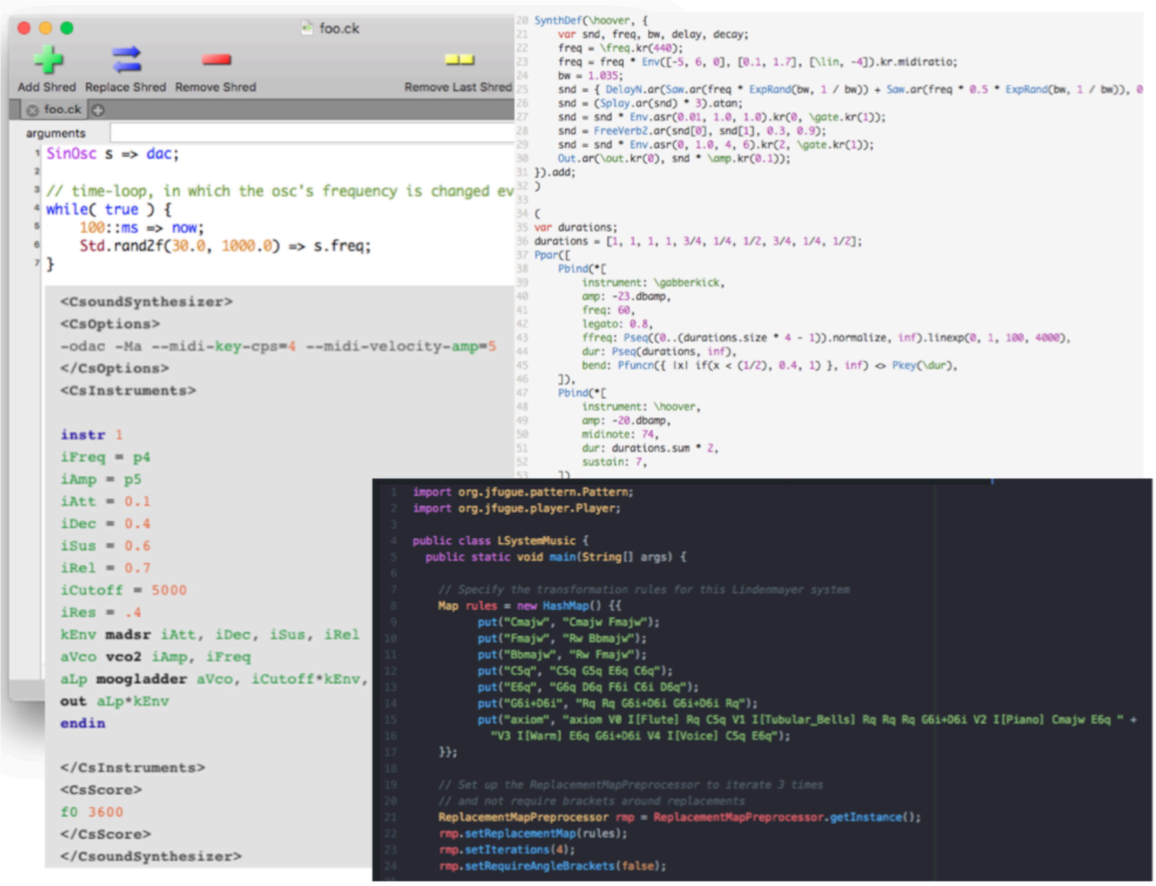
This research aims to avoid using generative systems that require the user to be a programmer. Nonetheless, they are briefly explored below.

4.3.1. Text Based

Sonic Pi (Aaron, 2016) is a live coding program for making music. It provides a way to write code similar to common programming languages and to generate audio output. The code can be modified in real-time and updated. Manipulation of lists provides similar generative capabilities to lisp programming (Cope and Mayer, 1996). Difficulties arise in writing more nuanced and complex structure with lots of musical lines. Although easy to pick up, the notation is alien to those familiar with common music sequencer workflow. Similar systems to Sonic Pi include Impromptu (Sorensen and Gardner, 2010) and Extempore (Sorensen, 2018). Live coding is geared more towards performance than composition. The process of live coding supports rapid content creation but not necessarily more complex, nuanced forms of composition.

ChucK (Wang, Cook, and Salazar, 2015) is a more generalised music programming language for real-time sound synthesis, music creation and real-time interactive music. Its syntax is similar to common high-level languages. Similar systems (Figure 4.26) include Supercollider (Wilson, Cottle, and Collins, 2011), CSound (Vercoe and Ellis, 1990), and JFugue (Kartika, 2010).

Nash (2014) has extended music tracker notation with a programming-like syntax to enable generative and algorithmic music applications. The system uses familiar spreadsheet-like cell programming, which is more familiar to a common user than that of a specific music programming language; and creates inroads to making generative music (and music programming in general)



```
SinOsc s => dac;

// time-loop, in which the osc's frequency is changed ev
while( true ) {
    100::ms => now;
    Std.rand2f(30.0, 1000.0) => s.freq;
}
```

```
<CsoundSynthesizer>
<CsOptions>
-odac -Ma --midi-key-cps=4 --midi-velocity-amp=5
</CsOptions>
<CsInstruments>

instr 1
iFreq = p4
iAmp = p5
iAtt = 0.1
iDec = 0.4
iSus = 0.6
iRel = 0.7
iCutoff = 5000
iRes = .4
kEnv madsr iAtt, iDec, iSus, iRel
aVco vco2 iAmp, iFreq
aLp moogladder aVco, iCutoff*kEnv,
out aLp*kEnv
endin

</CsInstruments>
<CsScore>
f0 3600
</CsScore>
</CsoundSynthesizer>
```

```
SynthDef(\hoover, {
    var snd, freq, bw, delay, decay;
    freq = \freq.kr(440);
    freq = freq * Env([-5, 6, 0], [0.1, 1.7], [\lin, -4]).kr.midiratio;
    bw = 1.035;
    snd = { DelayN.ar(Saw.ar(freq * ExpRand(bw, 1 / bw)) + Saw.ar(freq * 0.5 * ExpRand(bw, 1 / bw)), 0
    snd = (Splay.ar(snd) * 3).atan;
    snd = snd * Env.asr(0.01, 1.0, 1.0).kr(0, \gate.kr(1));
    snd = FreeVerb2.ar(snd[0], snd[1], 0.3, 0.9);
    snd = snd * Env.asr(0, 1.0, 4, 6).kr(2, \gate.kr(1));
    Out.ar(\out.kr(0), snd * \amp.kr(0.1));
}).add;
)

(
var durations;
durations = [1, 1, 1, 1, 3/4, 1/4, 1/2, 3/4, 1/4, 1/2];
Ppar([
    Pbind(*[
        instrument: \gabberkick,
        amp: -23.dbamp,
        freq: 60,
        legato: 0.8,
        ffreq: Pseq((0..(durations.size * 4 - 1)).normalize, inf).linexp(0, 1, 100, 4000),
        dur: Pseq(durations, inf),
        bend: Pfuncn({ |x| if(x < (1/2), 0.4, 1) }, inf) <> Pkey(\dur),
    ]),
    Pbind(*[
        instrument: \hoover,
        amp: -20.dbamp,
        midinote: 74,
        dur: durations.sum * 2,
        sustain: 7,
    ])
```

```
import org.jfugue.pattern.Pattern;
import org.jfugue.player.Player;

public class LSystemMusic {
  public static void main(String[] args) {

    // Specify the transformation rules for this Lindenmayer system
    Map rules = new HashMap() {{
          put("Cmajw", "Cmajw Fmajw");
          put("Fmajw", "Rw Bbmajw");
          put("Bbmajw", "Rw Fmajw");
          put("C5q", "C5q G5q E6q C6q");
          put("E6q", "G6q D6q F6i C6i D6q");
          put("G6i+D6i", "Rq Rq G6i+D6i G6i+D6i Rq");
          put("axiom", "axiom V0 I[Flute] Rq C5q V1 I[Tubular_Bells] Rq Rq Rq G6i+D6i V2 I[Piano] Cmajw E6q " +
            "V3 I[Warm] E6q G6i+D6i V4 I[Voice] C5q E6q");
    }};

    // Set up the ReplacementMapPreprocessor to iterate 3 times
    // and not require brackets around replacements
    ReplacementMapPreprocessor rmp = ReplacementMapPreprocessor.getInstance();
    rmp.setReplacementMap(rules);
    rmp.setIterations(4);
    rmp.setRequireAngleBrackets(false);
```

Figure 4.26.: A collection of different music programming systems, (Chuck, Superclolider, Csound, and JFugue).

integrated into an existing composition workflow. As discussed previously, the tracker interface on which the system is built is not as commonly used as other music interfaces/notations.

4.3.2. Visual Programming

Visual programming (Figure 4.27) is an alternative to textual programming, replacing text with GUI components. Notably, MaxMsp (Manzo, 2016) and Puredata (Pd-community, 2018) use objects and patch cables to create a plethora of interactive music systems. Max offers a reasonably low entry threshold, and has a high ceiling, but not well suited for creating both coarse and fine music structure given a lack of a timeline metaphor, making control flow difficult to predict. It is unsuited for managing large data sets (required for certain types of generative music such as machine learning). Max, like many other systems explored, offers entirely different workflows to that of common music sequencer applications. Although an unorthodox interface is often required for creating non-trivial musical interfaces and instruments, like those afforded by Max.

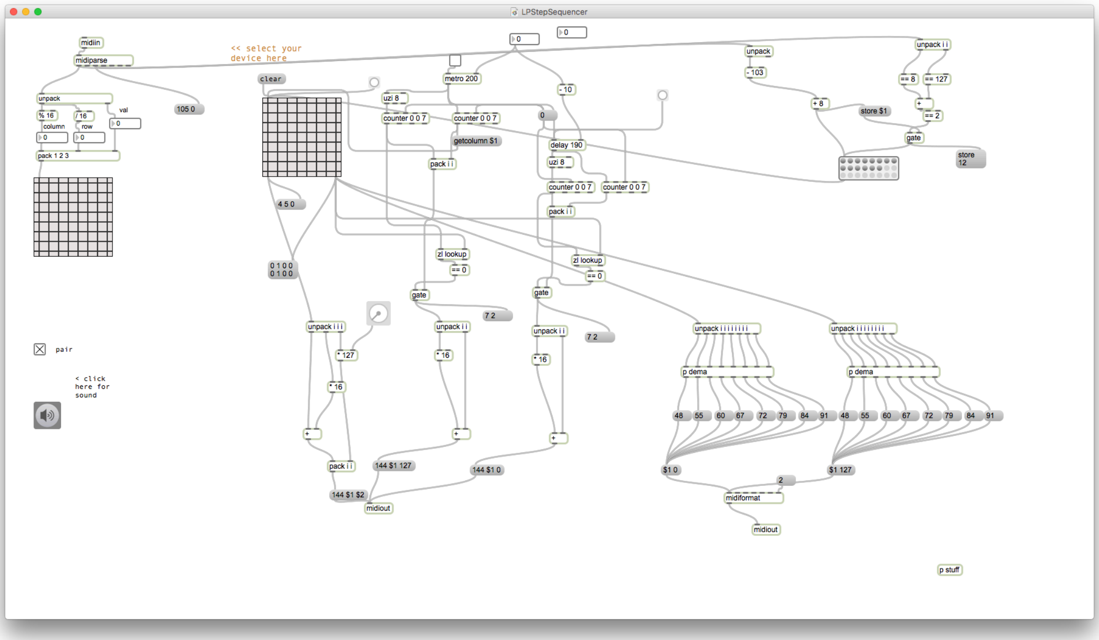
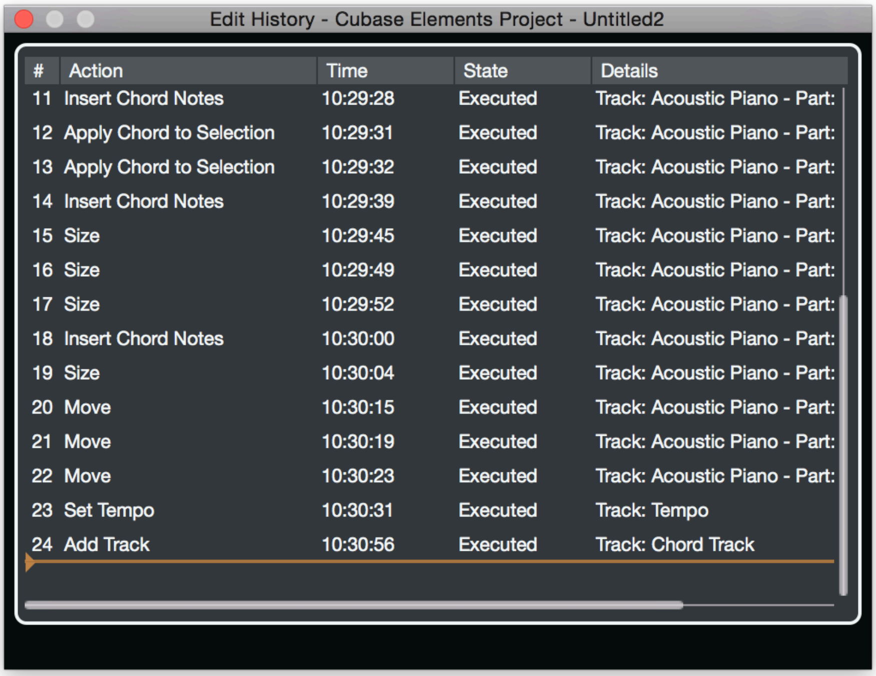


Figure 4.27.: Max's visual programming interface.

Open Music (Bresson, Agon, and Assayag, 2011) is another environment for music composition that uses visual programming languages for the generation or manipulation of musical material. This provides a lower-entry threshold than more traditional programming languages, and also allows familiar items such as scored elements to be retained.

4.4. Version Control

Version control is generally limited in most music software. Cubase has a primitive version control system that keeps track of edits to plug-in parameters, MIDI clips, and mixer configuration in a historic list (Figure 4.28). Unlike other version control systems used in software development, it does not provide any type of *commit*, i.e. a series of edits that contribute to a major change (or version), such as adding a drum track. The information leans towards being too detailed, rather than too succinct. In more general music software, the tools are limited. An undo list is a basic version control system, but the list is often lost on program exit. Although users are free to *save as*, and create different versions, it is very difficult to merge and manage these variations. Version control is easily applied to the programming code-based systems discussed previously, simply by wrapping them in an existing version control system, i.e. Git (Loeliger and McCullough, 2012), but this is not tailored for musical applications.



Edit History - Cubase Elements Project - Untitled2

#	Action	Time	State	Details
11	Insert Chord Notes	10:29:28	Executed	Track: Acoustic Piano - Part:
12	Apply Chord to Selection	10:29:31	Executed	Track: Acoustic Piano - Part:
13	Apply Chord to Selection	10:29:32	Executed	Track: Acoustic Piano - Part:
14	Insert Chord Notes	10:29:39	Executed	Track: Acoustic Piano - Part:
15	Size	10:29:45	Executed	Track: Acoustic Piano - Part:
16	Size	10:29:49	Executed	Track: Acoustic Piano - Part:
17	Size	10:29:52	Executed	Track: Acoustic Piano - Part:
18	Insert Chord Notes	10:30:00	Executed	Track: Acoustic Piano - Part:
19	Size	10:30:04	Executed	Track: Acoustic Piano - Part:
20	Move	10:30:15	Executed	Track: Acoustic Piano - Part:
21	Move	10:30:19	Executed	Track: Acoustic Piano - Part:
22	Move	10:30:23	Executed	Track: Acoustic Piano - Part:
23	Set Tempo	10:30:31	Executed	Track: Tempo
24	Add Track	10:30:56	Executed	Track: Chord Track

Figure 4.28.: Cubase's note edit list.

As most music software has little or no version control, its makes working with generative music difficult as iterations and parameters are easily overwritten or lost. Duignan (2010) notes that version control and snapshot tools are practically non-existent in existing music software and that it is necessary to work around such issues. This suggests music software systems could benefit from such tools. A recent development in this area is Splice studio (Splice.com, 2021) that offers a form of version control for multiple different music software packages via a third- party web-service, although this *"outside of the box"* solution necessitates the use of a reliable internet connection.

4.5. Conclusion

To conclude, inroads on integrating generative effects in existing music sequencers have been made, but remain mostly primitive. Notable exceptions include Logic Pro's smart drummer, FL Studio's riff machine and Numerolgy's Evolutionary Algorithms. Generative music systems are often too different from existing music sequencing workflows, using odd notations or avoiding design metaphors such as a timeline. Programming-based systems, unlike other systems, are considerably more complex and scalable but are often squarely aimed at programmers requiring code literacy skills.

Despite the safety net of knowing that model parameters and output are stored elsewhere and that edits are not destructive, the inability to rapidly generate music in isolation is detrimental to people working in the field, thus version control systems are paramount for CGM.

Music is evaluated visually through inspecting the notation but primarily from listening to it. The output of many computer-generated effects is not placed back on the timeline (as the timeline still shows the input material); rather, the output is in real-time. Users are not able to visualise the output of the effect but have to rely on audible evaluation[4]. Users could reroute the live output and record this on a secondary track to see the notation, but this is perhaps an unnecessary workaround and makes fine-grain editing of generative output difficult.

The output from code-based systems is evaluated almost exclusively audibly, as there is no timeline in which notes can be edited or inspected. The mapping between blocks of code or audible output can be difficult to comprehend with such systems.

To summarise, popular systems include little support for computer-generated music. Experimental systems provide inroads to generative music but do not conform well to existing composer workflows; while code-based systems have a high computer literacy threshold, but remain extremely powerful and feature complete.

[4] Some specific effects, notably FL Studio's arpeggiator does, in fact, do this.

Part III.

DEVELOPMENT

5. Design Elicitation

From researching existing systems (Chapters 3 and 4), it is clear that few, if any, support computer-generated music techniques using the provided music sequencing and editing workflows. This chapter proposes the design requirements for such a system. The plethora of existing computer-generated techniques is narrowed down and a set for implementation is chosen. An original piece of research on composers' perception of computer-generated music is detailed in section 3. Primary and secondary research is combined, resulting in the development and implementation of IGME. Work from this chapter was originally presented at CSMC[1] 2017 (Hunt, Nash, and Mitchell, 2017).

5.1. Computer-Generated Music Techniques

Chapter 3 presented a framework for categorising different computer-generated music techniques, with the table reproduced in Table 5.1, and Chapter 4 looked at how such techniques had been adopted with software interfaces for music composition. Each dimension is discussed below, leading to an evaluation of each technique in Table 5.2. This process, therefore, proposes which techniques are likely to be most suitable for an end-user computer-generated music system, in the scope and context of this project's research aims and objectives.

Type

Both *generative* and *transformation* based techniques are suitable but *analytic* techniques have some inherent difficulties. Firstly, the composer needs content for analysis. Secondly, such processes can often take considerable time to analyse, creating unnecessary breaks in a composer's workflow. One proposal is that composers interact with a pre-determined dataset, and another proposal is

[1] Conference on Computer Simulation of Musical Creativity.

Technique	**Type**	**Feature-space {Music, Parameters}**	**Reproducible**	**Appropriate hierarchal level**
Random number generation	Generative	{None, None}	No	Note - Phrase
Deterministic Rules	Transformational	{None, Simple}	Yes	Note - Global
Probabilistic rules	Transformational	{None, Simple}	No	Phrase - Section
N-gram models	Analytical	{Dataset, None}	Yes	Phrase - Section
Markov chains	Analytical	{Dataset, None}	Yes	Phrase - Section
Simple grammars (type 3)	Transformational	{Fractional, Simple}	Yes	Note - Phrase
Complex grammars (type 0)	Transformational or Analytical	{Dataset, Complex}	No	Phrase - Section
L-Systems	Transformational	{Dataset, Complex}	No	Section
Neural networks	Analytical	{Dataset, Simple}	Yes	Section
RNN and LTSM	Analytical	{Dataset, Simple}	Yes	Global
Deep learning	Analytical	{Dataset, Complex}	Yes	Global
Evolutionary and genetic algorithms	Generative	{Fractional, Complex}	No	Section - Global
Cellular automata	Generative	{None, Simple}	No	Section - Global
Arpeggiators	Transformational	{Fractional, Simple}	Yes	Phrase
Harmonisers	Transformational	{Fractional, Simple}	Yes	Note

Table 5.1.: A table showing the categorisation of various computer-generated music techniques.

that the system is continuously trained in the background, with the complexities hidden from the user.

Feature Space

As the system envisioned will work alongside content created by the composer, the size of *music data* available is limited. Techniques requiring a large dataset (for example, deep learning) are difficult to obtain and implement effectively. On the other hand, Markov models can be trained with music already in the environment (session). Minimising the number of parameters creates a trade-off between ease of use, and making the effects highly customisable, however, where possible, the number of on-screen controls should be minimised.

Technique	Integrated	Reason for or against
Random number generation	Yes	Simple for composer's to understand.
Deterministic rules	Yes	Includes scale quantisers (essential in conjunction with other techniques).
Probabilistic rules	Yes	Simple to implement and chain up (includes a number of techniques).
N-gram models	No	Basic concept similar enough to a Markov model, so chosen instead.
Markov chains	Yes	Implement as a first-order transition table.
Simple grammars (Type 3)	No	L-system chosen instead.
Complex grammars (Type 0)	No	Is complex to implement and understand.
L-systems	Yes	Parallel rewriting makes it easier to implement and understand from the composer's perspective.
Neural Networks	No	Requires large datasets and impractical to use in realtime.
RNN and LTSM	No	
Deep learning	No	
Evolutionary and genetic algorithms	Yes	Despite being complex can be broken into stages, i.e. evolve a single musical element.
Cellular automata	No	Evolves over time, works on too high an hierarchical level.
Arpeggiators	Yes	Commonly understood and found in existing software/hardware.
Harmonisers	Yes	
Music programming	No	Avoiding programming based approaches are a key element of the project.

Table 5.2.: A table showing the different computer-generated techniques and processes used in this project.

Reproducibility

As some techniques cannot be reproduced[2], it is paramount to consider how the output and parameters of systems are stored. This is captured by the version control system discussed shortly.

Appropriate Hierarchical Level

Given that a key objective of the project is to create a human-computer creative system, anything that can generate an entire composition is undesirable. Therefore, the focus is primarily on generating content at the *note*, *phrase*, and *bar level*, that can be sequenced together into higher level structure.

Alpern (1995) notes that *"By building small parts with well-defined behaviour and linking them together, we can create a great variety of methods and compositional output."*. Therefore, smaller well-understood components, which can be combined, are preferable to larger 'do-everything'

[2] i.e. evaluating the same model parameters twice will likely result in a different output (i.e. stochastic)

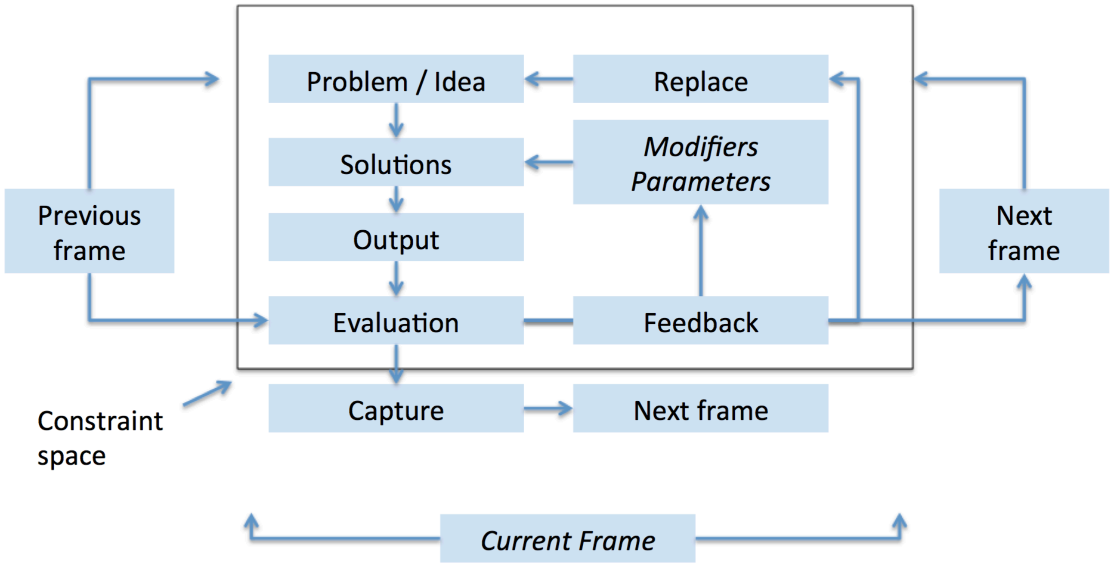


Figure 5.1.: Prototype model of computer-generated music interaction.

techniques in this research. Systems using rule-based and stochastic models are easy to comprehend and implement, as well as being fast to run, however produce only simple music. On the other hand, evolutionary techniques for producing music, can produce sophisticated work, but need greater domain-specific knowledge and computing time, often requiring the composer to frequently contribute to the process (Wiggins *et al.*, 1998).

It is hypothesised that composers will use computer-generated processes for initial ideation but will not want to surrender ultimate control by, for example, producing a full composition at the push of a button. Based on this (and themes discussed so far) Table 5.2 shows the techniques taken forward for implementation and the reasons for doing so. Implementation details for each process/plug-in in the IGME software are given in appendix D.

5.2. Modelling Computer-Generated Music

Although it is difficult to objectively model a creative workflow, this section attempts to define a simple model, drawn from the work in Chapter 2, that can be applied to human and computer composition workflows. However, the model is not a universal solution.

Figure 5.1 shows the prototype model used in this research for helping to define computer-generated music as an end-user composition process. The *idea/problem* is defined as the input material, which can be a collection of pitches or a sequence of notes. The process takes the input and modifies it based on a series of rules or transformations to create solutions. Certain *parameters* are exposed for controlling the process. Following this is the *output*, at which point the music becomes audible. The *evaluation* stage creates a branch, where the composer is either satisfied with the output, or provides feedback to the initial seed and parameters in the process stage. This loop continues until the *evaluation* stage is passed, where a *'capture'* is created and the next frame begins. The constraint space (Alty, 1995) governs restrictions on the composition. For example, using the key C-major makes certain pitches more likely (Temperley, 2007).

The previous frame influences the current frame, in that it can impose restrictions over the constraint space. The evaluation stage is considered in reference to the previous frame, as music is auditioned in respect of a timeline. Likewise, changes from the feedback stage in the current frame have a knock-on effect for the subsequent frames. A *frame* has no defined length; instead, it should be considered as more than a single note but less than an entire section. This approach emphasises creating carefully controlled frames of music that are unlikely to spiral into chaos but retains inter-frame influence (higher-level musical structure) through attentive links between frames. How these links are encoded is explored through the *reference part* system discussed in Chapter 6.

This model can be applied to both real-world music and computer-generated systems. For example, a melody will likely need to fit within an existing harmonic structure, so the constraint space is limited by a key, and the input is a chord. The mapping between input and output is not necessarily simple.

An arpeggiator is a transformation-based music generation technique that can be easily mapped to the above model. The input *(idea/problem)* is the notes of the arpeggiator. The *parameters* represent the speed, octave and rhythmic pattern controls. The input notes and parameters produce an output. Upon hearing the output, the composer may either update the input notes (seed) or the arpeggiator parameters, which subsequently creates a different output iteration. Once the composer is satisfied, a *capture* is created for a given number of bars. The following frame is influenced by *its* previous frame, in that certain parameters are likely to be shared between the two. The constraint space in this example could be governed by the limitations of the arpeggiator effect, such as working with a total of 6 input notes at a time.

An iteration would be defined in this model as a change in parameters producing a change in output (computation → playback), whereas an audition (playback) would be no change in parameters or output. Stochastic processes produce a different output without necessarily changing any parameters. Iteration and playback are further discussed in Chapter 7.

The constraint space can be either deterministic or stochastic. For example, the key of a piece in tonal music places an immediate constraint space on the stochastic choice of notes and harmonies (Temperley, 2007). A C# major chord is unlikely to surface in a composition written in C-major. The rules of strict species counterpoint prohibit certain tonal progressions, such as parallel fifths, thus placing a deterministic rule on that constraint space, although music is as much about following rules as breaking them.

5.2.1. Existing Systems and Models

The proposed model (Figure 5.1) builds upon the ideas of Logic Pro X's (Apple, 2016) smart drummer (discussed in Chapter 4), in that a computer-generated section of the music is defined as a set of parameters that are evaluated at runtime. The output is somewhat stochastic but, as mentioned previously, does not change unless a model parameter is changed too[3]; therefore, loading a session from disk preserves previously calculated music.

Papadopoulos and Wiggins (1999) describe algorithmic composition as a set of rules for solving a problem by combining musical parts into a whole (composition). Holland (2000) presents a framework for modelling open-ended creative activities, based upon choosing a goal, selecting constraints, and then iterating through solutions. A result is generated throughout each step and the constraints and goals are adjusted until some acceptance criteria are met. The model is built on a framework that presents sets of modular but interacting components. The works listed above also have many parallels with the model discussed in the previous section (Figure 5.1).

5.3. Composer Survey and Requirements Capture

A survey was conducted to evaluate composers' attitudes to technology concerning computer-generated music, digital audio workstations, notation packages, and usability. The survey's content was influenced by previous work in the field by; Boyd (2013), Raines (2015),

[3] This characteristic is specific to this effect but not to other more general computer-generated processes.

and Service (2012), whose work involved interviewing various professional music practitioners from diverse backgrounds.

A summary of key findings, focusing on the questions relating to computer-mediated composition, is given below. Many of the questions not discussed, were evaluated to be less pivotal to this research. The full survey is discussed in appendix A. The survey was completed by 24 music practitioners.

5.3.1. Background

Table 5.3 shows the musical background of the participants, with the majority being music technology students. Table 5.4 summarises the participants' experience of music. Important observations include that 21/24 participants had composed some form of music and 9/24 identified as professional music practitioners. Results from Figure 5.2 show that participants had a broad knowledge of musical activities. More notably, just over half stated they had *above average* experience of composing music using a computer.

Figure 5.3 shows a box and whisker plot for the length of experience each participant had under *'general musicianship', 'composing/songwriting' and 'professional composition'*, with the results showing wide variance. The distribution for *professional composition* is skewed as a number of participants entered 0 years.

Q1: Which of these best describes your background?

Music Technology Student	Music Student	Recreational Composer	Professional Composer	Other
11	4	2	5	2

Table 5.3.: The musical background of each participant.

Q2: What current musical experiences do you have? (tick any that apply)

Question	Count	Percetage
I play piano.	13	54.2%
I play guitar.	13	54.2%
I play another acoustic instrument.	13	54.2%
I play several musical instruments.	16	66.7%
I listen to a lot of music.	20	83.3%
I can read music.	16	66.7%
I have had music lessons.	19	79.2%
I have studied music theory (scales, etc.)	21	87.5%
I have performed live	15	62.5%
I have composed music/songs/tunes.	21	87.5%
I practise a lot.	8	33.3%
I am a professional performer.	5	20.8%
I am a professional composer/songwriter.	9	37.5%
I have performed with friends.	17	70.8%

Table 5.4.: The participant's experience of music.

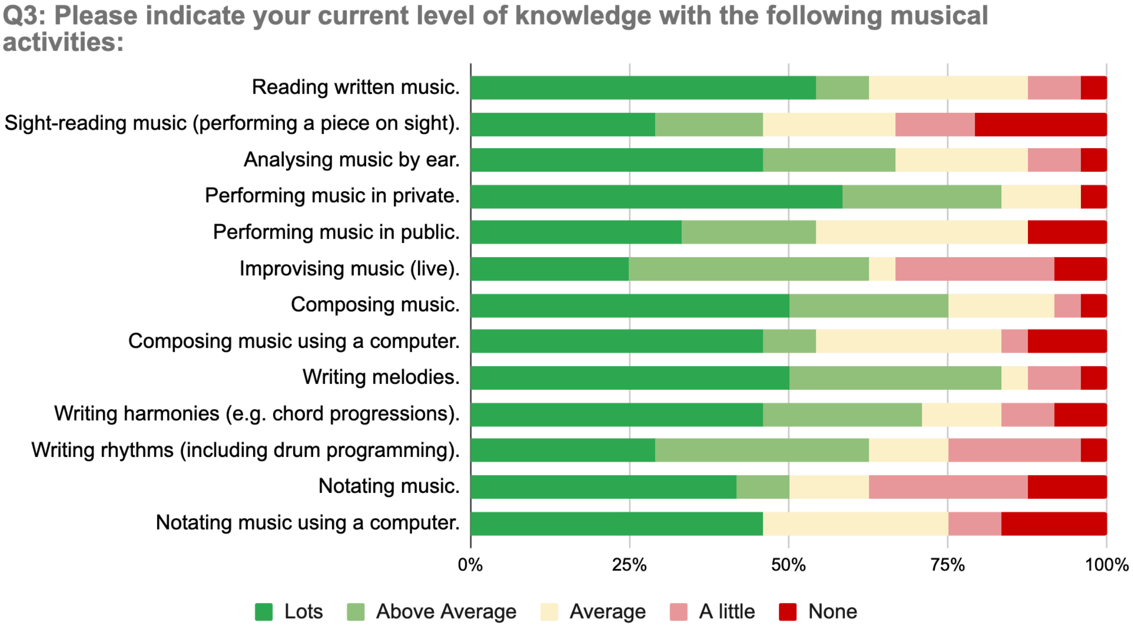


Figure 5.2.: Distribution of responses for participants' knowledge of musical activities.

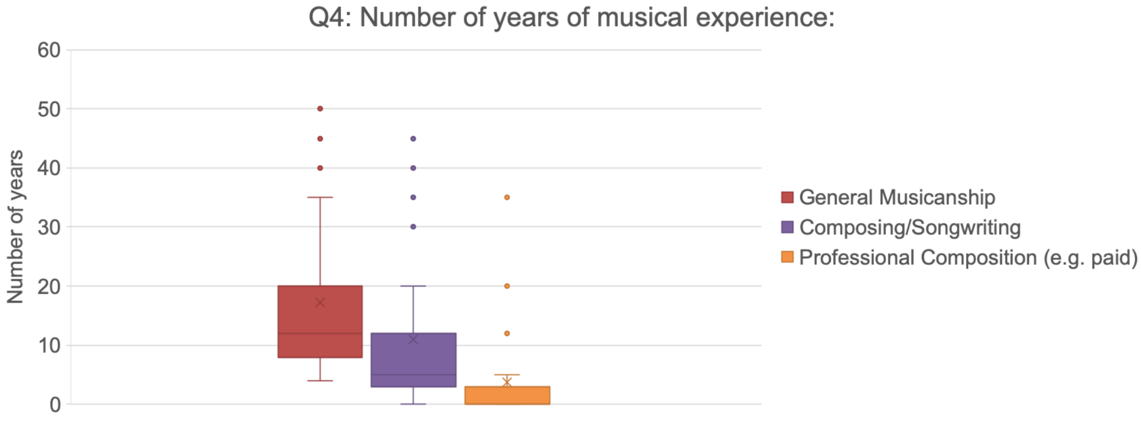


Figure 5.3.: Box and whisker plot of the number of years of musical experience, by type.

5.3.2. Experiences of Music

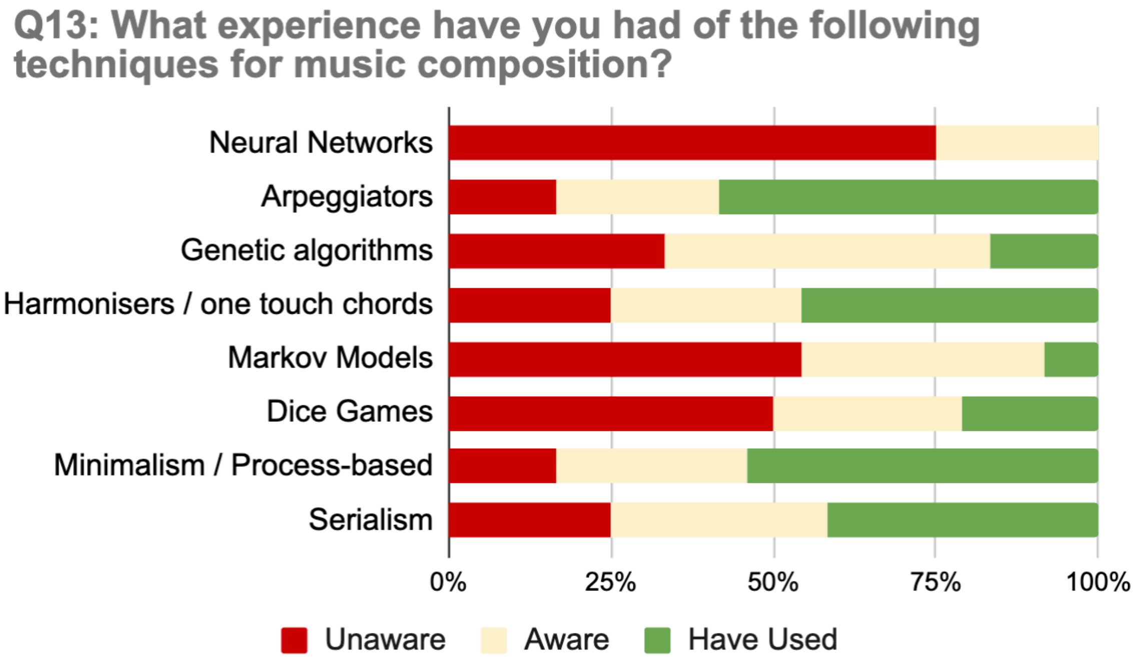


Figure 5.4.: Experience of composition techniques.

One of the survey questions was *"What experience have you had of the following techniques for music composition?"*. Figure 5.4 shows the arpeggiator is the most *used* technique. Arpeggiators and harmonisers are commonly found on consumer keyboards and feature widely in digital music sequencers. *Minimalism* and *Serialism* techniques also show positive *'usage'* and *'awareness'*. This is unsurprising as they are often taught on music education courses. Although neural networks and Markov models for computer-generated music are commonly explored in literature, most survey participants were *'unaware'* of them, and even fewer had *'used'* them. Many composers marked that they were *'aware'* of genetic algorithms, and three marked they had *'used'* them.

It could be argued that the reason composers have not used certain techniques is they are unaware of them [4]. Many of the more complex computer-generated techniques remain firmly in the research domain and have not transitioned into mainstream user-facing music software (as evidenced in Chapter 4).

Table 5.5 shows the results of asking survey participants *"If the computer was able to suggest musical ideas, would you find this feature..."*. Participants could choose from multiple selections. One-third found the suggestion *'interesting'* and/or *'useful for composer's block'*, with roughly the same number

[4] A potential research objective hypothesis based on this finding is discussed at the end of this chapter.

Q21: If the computer was able to suggest musical ideas, would you find this feature:

Metric	**Count**
Annoying	5/24
Intrusive	9/24
Crucial	0/24
Useful for 'composer's block'	13/24
Interesting	13/24
Novel	1/24
Inspiring	3/24
None of the above	0/24

Table 5.5.: Summary of responses for above question.

finding the idea *'intrusive'*. Obviously a clear separation is needed between making such a feature *'interesting'* to use and not making it *'intrusive'*. Few people found this to be *'novel'* or *'inspiring'*, and nobody found it to be *'crucial'*.

5.3.3. Summary

A recurrent theme throughout the study was the desire of users to have some control over a proposed computer-generated process. Many composers said such processes should not be intrusive. It is clear that processes need to be carefully designed and evaluated so the user retains authorship. One participant said they do not use automated techniques, such as retrograde, as the music would no longer be considered theirs. However, if they had completed the same process manually (resulting in exactly the same music), they would have considered it their own work.

Another hypothesis to be drawn from the results, is that understanding computer-generated processes leads to a higher chance of them being considered for use in composition. Composers are unlikely to use serialist techniques if they have never been taught them.

As the study identified, generating new ideas, exploring ideas, and creating accompaniments would seem practical roles that an automated composition process could take. This aspect is covered in more detail below, and helped to form research questions that were put to IGME users through user surveys (Chapter 8).

Generating New Ideas

This refers to computer-generated techniques that can compose original material, with the user specifying the parameters of a given model.
Research Question: *"I would use computer-generated techniques to help me to come up with new ideas".*

Exploring Existing Ideas

Allow the user to utilise computer-generated techniques for experimenting with different arrangements and support this in a suitable interface.
Research Questions: *"I would use computer-generated techniques to help me to explore different permutations of my own material"* and *"The interface provided allows me to effortlessly explore new ideas"*

Creating Automatic Accompaniments

Allow the user to create automatic accompaniments alongside their existing music.
Research Question: *"I would use the automatic accompaniments as a starting point in my composition practice."*

5.4. Formal Design Requirements

To meet the research objectives, a music composition system was developed similar in design to its contemporaries, this is arbitrary length MIDI clips arranged across a timeline of tracks sequenced from left to right (Figure 5.5). A conclusive list of design requirements are discussed at length in Chapter 6, using the Cognitive Dimensions of Notation framework as guidance.

5.4.1. Version Control

As discussed in Chapter 4, version control systems are rarely found in existing music software, but here they are deemed of paramount importance for computer-generated systems. For example, a composer may use a CGM technique to generate many iterations, only to find the most suitable (i.e. a previous revision) has been overwritten. Some techniques produce the same output on each

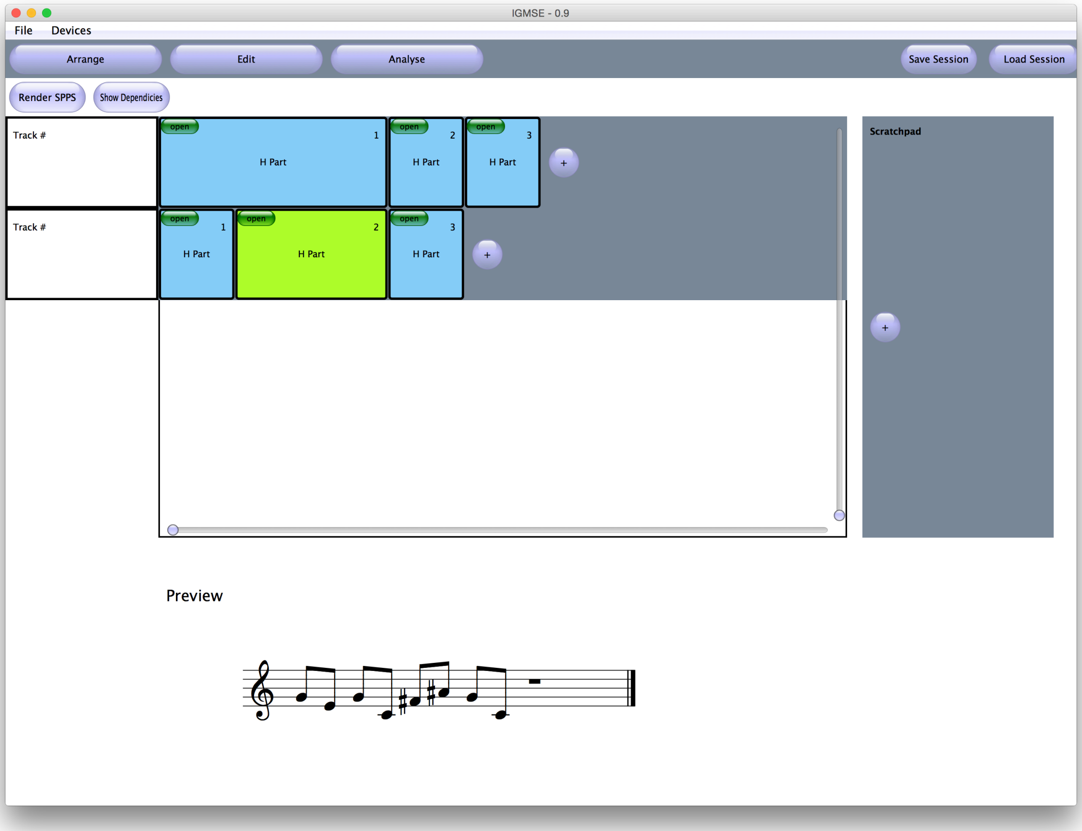


Figure 5.5.: Prototype IGME software.

iteration, whereas others produce stochastic unreproducible results. To address this, the system proposed here makes use of source/version control techniques, providing both a method for capturing unreproducible music parts and the ability to recall computer-generated interactions in a list, facilitating and encouraging experimentation. Chapter 6 further details this feature.

5.5. A Part-Based Interaction Model

It is clear that an end-user computer-generated music system should follow the metaphor of a clip, suggesting the music is broken down track by track into clips, sharing common functionality with most music sequencers. The content of a clip is composed by either a human-led process or from a computer-generated one, creating a clear divide between two mediums and giving complete control over the ratio of human and computer music. Many other benefits of using clips are the same as in other software, such as they are easy to replace, stretch, copy and paste, and distinguish. To avoid confusion, clips are referred to as *'parts'* in this project due to their slightly extended role.

A part in IGME is made of an initial musical idea and a set of optional parameters to configure how this music may be either transformed or generated. Such a part has three distinct components: the seed (initial material), parameters (for configuring computer-generated processes), and result (output music). A more technical definition of parts is given in the next chapter. The idea of formalising composition as a series of independent *'blocks'* was also suggested by Gartland-Jones (2002). The part concept was formed through the model presented in section 5.2. Chapel (2003), while interviewing prominent computer music researcher Dr Max Mathews, asked *"how would you use music made algorithmically"*. Dr Mathews replied:

> *"I would be interested in keeping an interaction with the algorithm; a part from the computer and a part from myself. I am interested in algorithms for improvising. With these algorithms, the musician and the computer play the music together. The algorithm chooses the notes, but the musician can select, among the options given by the program, the one he likes ."*

This quote heavily influenced this work.

5.6. Evaluating Computer-Generated Music Systems

Evaluating the musical output from any proposed system is neither a high priority nor a current research focus. Firstly, Sorensen and Brown (2008) believe not only is it impossible to measure objectively the aesthetic nature of music composition, but such a measurement can be dangerous as it can lead to explanations that do not bear scrutiny (Brown, 2004). Papadopoulos and Wiggins (1999) note that musical output in isolation, such as generative melodies from Markov models, are redundant without harmonic context. Papadopoulos and Wiggins (1999) suggest that evaluation of the musical output should be reviewed by professional musicians. Such a suggestion is however impractical and remains subjective.

Instead of focusing on the music composed, we focus on the methods used to create it, as music loved by one user may be loathed by another. The research objective is to understand how composers interact with computer-generated music techniques, within the software built for this project. A mixed-methods approach is used for studying user interaction, using questionnaires, interaction logging, and screen recordings, spread across controlled and uncontrolled conditions. Several studies were conducted and are further discussed in Chapter 8. Compartmentalising the music into parts meant the composition process could be observed at both micro (editing of notes) and macro (arranging of parts) levels.

5.7. Conclusion

This chapter discussed the development of the formal models and frameworks that underpin IGME, with more details given in the following chapters. Chapter 6 uses the Cognitive Dimensions of Notations framework to compare existing interfaces for CGM and uses this to help define a set of design guidelines for building end-user CGM systems. Through this process, the development and rationale behind IGME's features are detailed. Chapter 7 details the technical architecture of IGME. Chapter 8 discusses various pilot studies, the participatory design process and formal methodologies, which in turn finalise many aspects of the software.

6. A Cognitive Dimensions Approach for the Design of End-User Computer-Generated Music Systems

This chapter describes how the Cognitive Dimensions of Notation framework (CDN) provides a set of metrics and vocabulary both for guiding the design of computer-generated music software and evaluating it. As noted in Chapters 3 & 4, existing systems for computer-generated music offer interaction paradigms that are too dissimilar to mainstream music composition software. Therefore, this chapter has three purposes:

- To compare existing computer-generated music systems and mainstream music software under the CDN.
- To propose a good usability profile for end-user computer-generated music systems.
- To discuss some of IGME's core features and design heuristics under the CDN.

This chapter is a stepping stone between research, development, and the later user-facing evaluation, serving as a broad look at design choices implemented in IGME and comparing them with other music sequencing software and computer-generated systems. The chapter includes work presented at Tenor[1] 2018 (Hunt, Mitchell, and Nash, 2018).

6.1. Context and Rationale

Whereas the previous chapter explored the research questions through an exploratory lens, this chapter imposes a formal approach, which in turn leads to a set of formal design proposals for end-user CGM systems. These proposals are evaluated and redefined in the conclusion, becoming a set of prescribed design heuristics for end-user CGM systems.

[1] The 2018 International Conference on Technologies for Music Notation and Representation.

Dimension	Description
1. Visibility	How easy is it to view and find elements of the music during editing?
2. Juxtaposabillity	How easy is it to compare elements within the music?
3. Hidden Dependencies	How explicit are the relationships between related elements in the notation?
4. Hard Mental Operations	How difficult is the task to work out in your head?
5. Progressive Evaluation	How easy is it to stop and check your progress during editing?
6. Conciseness	How concise is the notation?
7. Provisionality	How easy is it to experiment with ideas?
8. Secondary Notation	How easy is it to make informal notes to capture ideas outside the formal rules of the notation?
9. Consistency	Where aspects of the notation mean similar things, is the similarity clear in the way they appear?
10. Viscosity	Is it easy to go back and make changes?
11. Role Expressiveness	Is it easy to see what each part of the notation means?
12. Premature Commitment	Do edits have to be performed in a prescribed order, requiring you to plan or think ahead?
13. Error Proneness	How easy is it to make annoying mistakes?
14. Closeness of Mapping	Does the notation match how you describe the music yourself?
15. Abstraction Management	How can the notation be customised, adapted, or used beyond its intended use?

Table 6.1.: A table showing the terms of the Cognitive Dimensions of Music Notations framework used in this research - taken from Nash (2015).

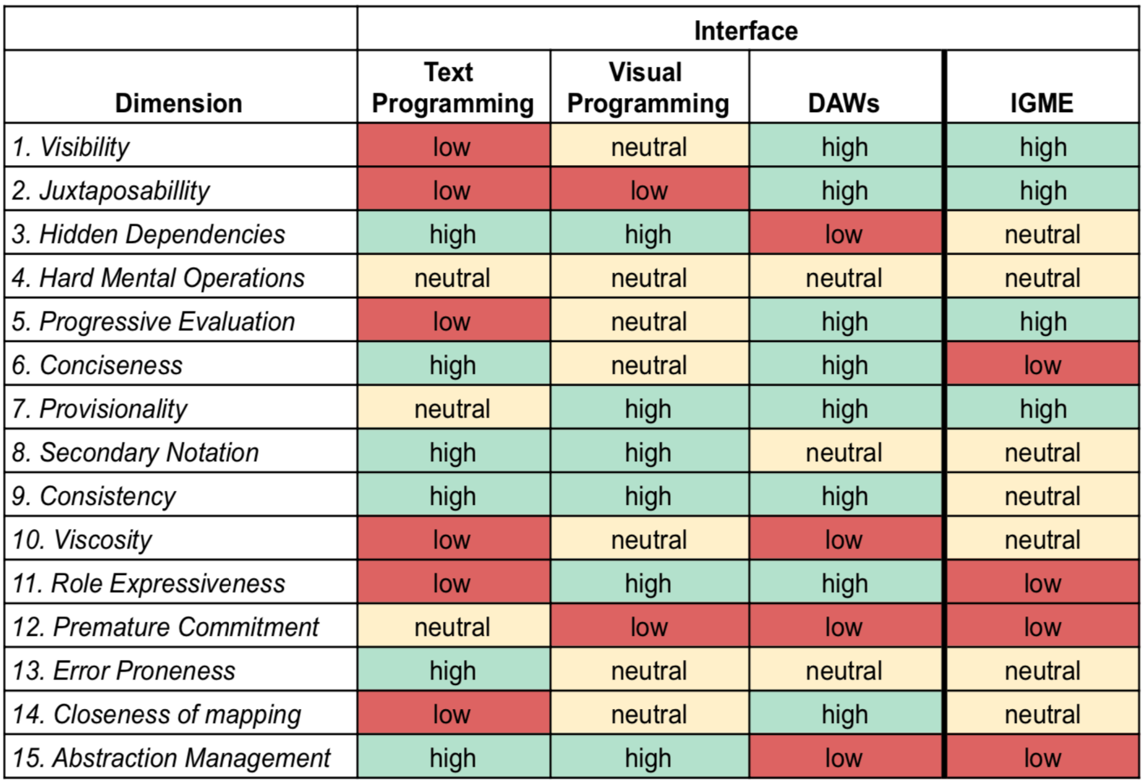
Chapter 2 identified that many frameworks (Carroll, 2003) for designing software focus on result or performance-based metrics (i.e. the speed at which a task can be completed). Such frameworks conflict with creativity-based activities which have minimal measurable goals, therefore leaving a paucity of formal design frameworks in which to scope software for supporting creativity. Nash (2011) suggests using more holistic frameworks, such as the CDN help design tools, to support creativity. Given the many parallels between Nash's work and this research, the CDN framework was chosen.

Nash (2011) highlights the existence of minimal quantitative-based approaches for comparing CDN dimensions across different software interfaces (for example, between music trackers and score notation software). Nash's work, based on empirical evaluation of music software, revealed a common profile among different software. The identified characteristics broadly corresponded with existing design principles (Blackwell and Collins, 2005; Duignan, 2010) for building software designed for musical creativity. These (from Nash, 2011) are given below as dimensions to either prioritise or mitigate:

- High visibility (ease of viewing and finding data)
- High juxtaposability (ease of comparing data)
- Low viscosity (resistance to changing data)
- Low diffuseness (conciseness, helping visibility and editing)
- High role expressiveness (ease of determining the role of objects)
- High provisionality (ease of sketching and experimentation)
- High progressive evaluation (ease of checking progress)
- High consistency (facilitating sense of control and learning)
- Low premature commitment (freedom to change paths)

IGME is not designed as a universal music sequencer; as such, it imposes a few limitations that guided this design process. Firstly, users are already expected to have domain-specific knowledge of music sequencing, but minimal skill in programming-based activities. IGME broadly follows the strengths and weaknesses inherent in other music sequencing software and focuses on working with Western music notation.

The remainder of the chapter goes through each dimension iteratively, with descriptions (Table 6.1) taken from Nash's (2015) work. Figure 6.1 shows a comparison between programming-based approaches (both visual and text), digital audio workstations, and what IGME aims to be under each dimension. It is important to note that the dimensions are often discussed as *qualitative* descriptors, making a full *quantitative* comparison difficult and beyond the scope of this research. The values in the table were assembled from the work of Nash (2015) and Bellingham, Holland, and Mulholland (2014b) and summarise findings from the more rigorous discussion of the proceeding sections. The values are not intended to be *ground truth*, but are presented simply to create a broad-brush comparison.



	Interface			
Dimension	Text Programming	Visual Programming	DAWs	IGME
1. Visibility	low	neutral	high	high
2. Juxtaposabillity	low	low	high	high
3. Hidden Dependencies	high	high	low	neutral
4. Hard Mental Operations	neutral	neutral	neutral	neutral
5. Progressive Evaluation	low	neutral	high	high
6. Conciseness	high	neutral	high	low
7. Provisionality	neutral	high	high	high
8. Secondary Notation	high	high	neutral	neutral
9. Consistency	high	high	high	neutral
10. Viscosity	low	neutral	low	neutral
11. Role Expressiveness	low	high	high	low
12. Premature Commitment	neutral	low	low	low
13. Error Proneness	high	neutral	neutral	neutral
14. Closeness of mapping	low	neutral	high	neutral
15. Abstraction Management	high	high	low	low

Figure 6.1.: A simple CDN comparison between different interfaces for CGM.

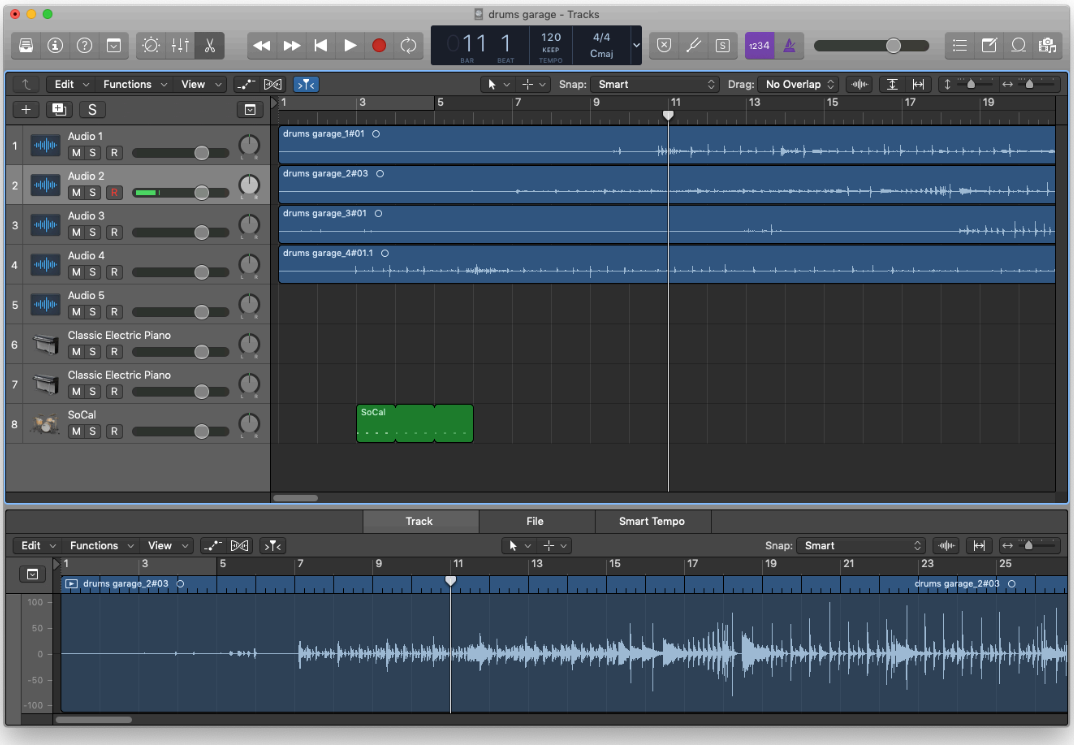


Figure 6.2.: The arrange view (timeline) in Logic Pro X.

6.2. Visibility

"How easy is it to view and find elements or parts of the music during editing?"

For any GUI interface, the ease at which information can be viewed and found is paramount. Most existing music sequencers display information in a timeline (Figure 6.2) based view, whereby clips of music are arranged respective to time (left-right) and track (up-down). Additional information, such as plug-in parameters and clip content, are visible as breakout windows or sub-components (Figure 6.3).

Bellingham, Holland, and Mulholland (2014a) criticises patch-based interfaces for their unclear structure, with layers spread across windows, which scales as system complexity grows. The same criticism can be directed at code-based systems with their many layers of abstractions. For, example a parameter for controlling a CGM effect may be hidden with several layers of classes or functions.

End-user CGM systems should prioritise this dimension by following the interaction paradigms that already exist within DAWs that enable high visibility. In addition, computer-generated effects

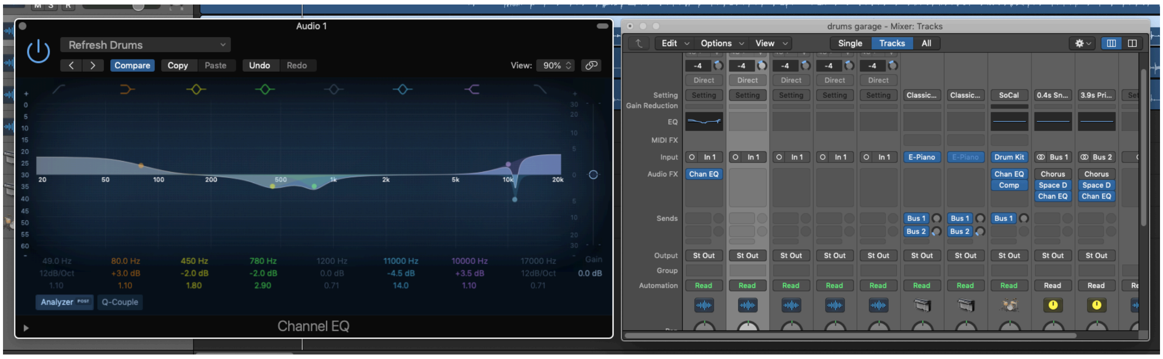


Figure 6.3.: Breakout windows in Logic Pro X (EQ plug-in and mixer)

should be controlled with graphical controls similar to those found in audio/MIDI plug-ins. The values of the plug-ins are easier to see compared with programming-based systems whereby data is often represented in variables and can be defined in a different place from where it is used.

6.3. Juxtaposability

"How easy is it to compare elements within the music?"

Existing tools, such as volume, solo, and mute controls, in DAWs permit parts of the music to be compared quickly and easily. As noted in Chapter 5 there is minimal support for version control in existing music software. Bellingham, Holland, and Mulholland (2014a) note that form-based systems such as Tune Smithy (Walker, 2008) and the Algorithmic Composition Toolbox (Berg, 2018) do not allow users to see older entries as they are replaced, potentially harming *juxtaposability*. Therefore, either undo has to be used or everything has to be kept in working memory, increasing *hard mental operations*.

It can be difficult to compare elements in a tightly-coupled programming-based environment for computer-generated music. Some techniques, such as 'commenting'[2] out blocks of code, can be beneficial. More complex changes often require the program to be recompiled. Where systems output music in real-time, it can be finicky to A/B test different variants, so MIDI or music data needs to be piped into a third party program for offline comparison.

The version control systems developed for IGME increases *Juxtaposability*. Each iteration can be quickly swapped in and out with the result auditioned in context with the music around it. The included *diff* tool (Figure 6.4) shows the explicit difference between two parts.

[2] A process by which code is marked as code that is not compiled

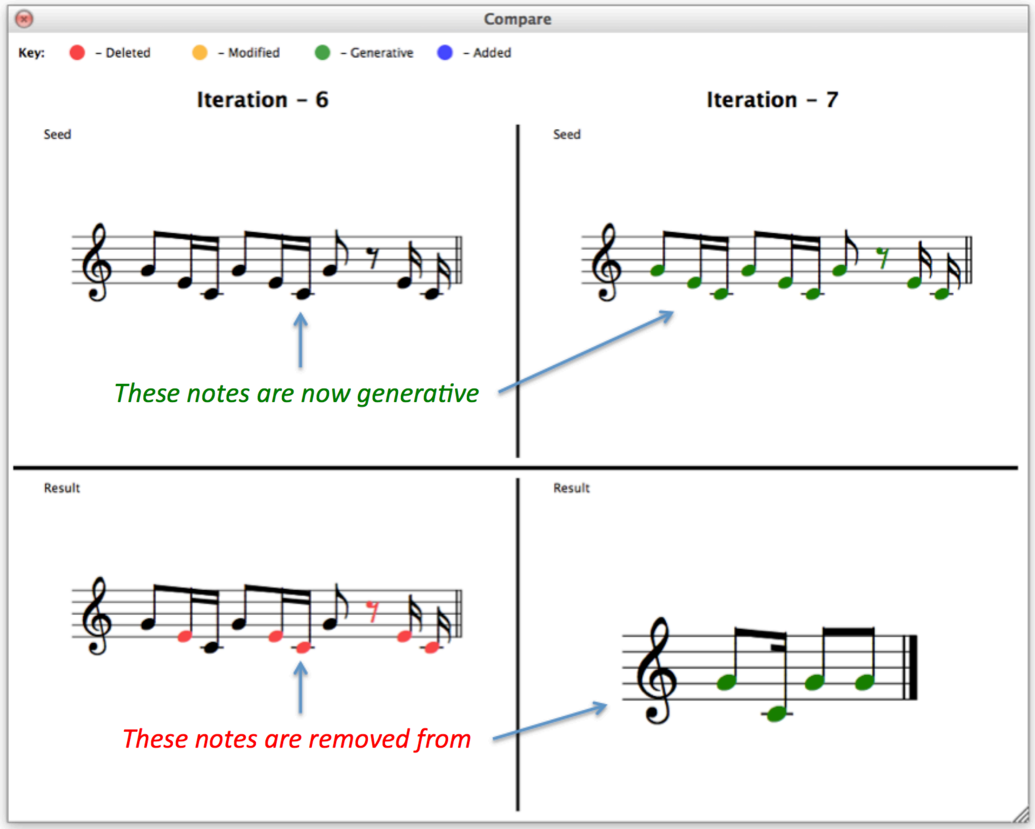

Compare
Key: ● - Deleted ● - Modified ● - Generative ● - Added
Iteration - 6
Iteration - 7
Seed
Seed
These notes are now generative
Result
Result
These notes are removed from

Figure 6.4.: IGME's *diff* tool comparing two different iterations.

6.4. Hard Mental Operations

"When writing music, are there difficult things to work out in your head?"

Music is often seen as a problem seeking and subsequently problem-solving activity (discussed in Chapter 2). Therefore, the composer needs an inherent challenge in which to engage (Nash, 2011). Certain aspects of composition can be made *'hard'* through bad interfaces, which is something that should be minimised.

Existing systems for CGM often require the design and implementation of algorithmic techniques using either code or a graphical programming environment, therefore placing a high mental load on the user. To engage non-programmers, IGME focuses on using in-built, rather than designing, processes for composition.

Bellingham, Holland, and Mulholland (2014a) stress the need for a clear visualisation showing the signal flow between components. This is a common problem for systems without a clear indication of a timeline, such as in coding or patching systems. IGME, just like a DAW, uses a

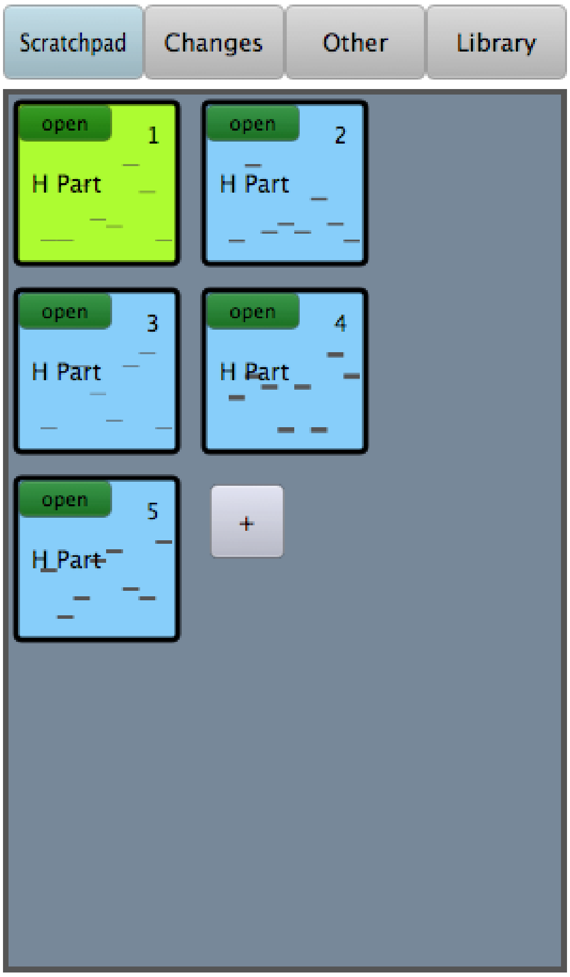


Figure 6.5.: IGME's scratchpad for creating musical ideas.

timeline, with time flowing from left to right. The user is not required to predict control flow, thus reducing cognitive load. It can be argued that such stochasticism is ideal for generative music. However, the focus is on a human-computer co-creation system, so it is important to make this signal flow more formal.

6.5. Progressive Evaluation

"How easy is it to stop and check your progress during editing?"

The *timeline* concept found in DAWs permits high progressive evaluation, allowing users to freely move the play-head around. When systems do not have a '*timeline*' (code and patch-based) it can be much more challenging to do this and perhaps impossible in the case of a programmatic stochastic music system that evolves over time.

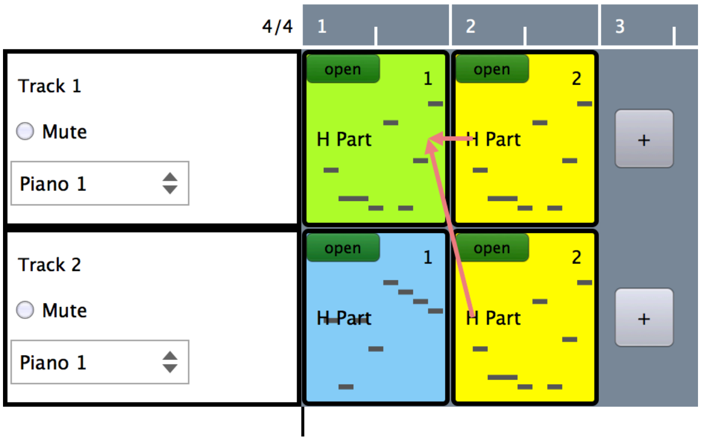


Figure 6.6.: Part 1 on track 1 is being repeated (referenced) by 2 other parts.

Collins (2005) states that evaluating the material is important for music software, as iteration is a primary concept in composition. A user interacting with any form of composition software is likely to apply a trial and error approach, testing many ideas and combinations. Nash and Blackwell (2012) note that a rapid edit-audition cycle contributes to a high state of *flow*, a desirable mental state for users engaging with creative exercises, such as music. Most existing music software makes it easy to stop and audition parts throughout and, usually, to make quick edits.

Code-based environments can have several roadblocks that can limit progression. Developing intricate systems can often entail building several classes or containers that must be reasonably complete and error-free before their output can be used/heard. The introduction of compile-time errors can cause the composition workflow to stop altogether.

6.6. Hidden Dependencies

"How explicit are the relationships between related elements in the notation?"

The connection between components in a DAW is generally quite explicit and minimal. Automation lanes and external bus *'sends'* in DAWs can create dependencies, but these are mitigated through having high *visibility* and remaining optional features for *'power users'*. Music can have innate hidden dependencies, such as pitches being dependent on *key* and instruments that are transposed. Hidden dependencies are often a side effect of a complex system. In general, dependencies are not desirable. However, they can be made more explicit.

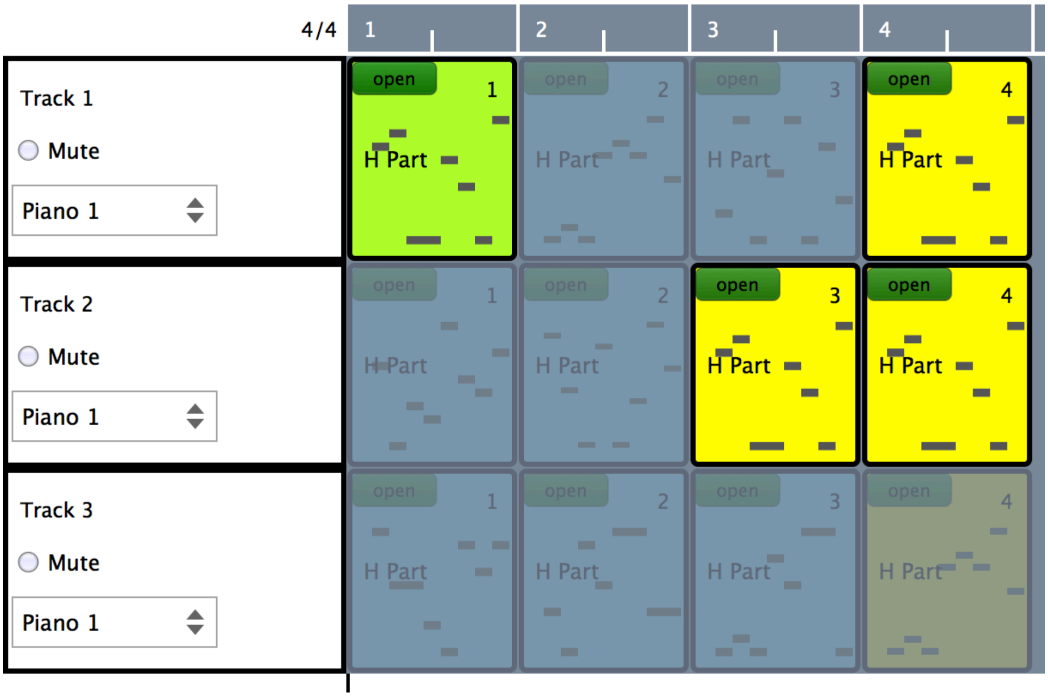


Figure 6.7.: IGME's show dependants features, highlights all parts that are dependant on part 1 track 1.

Programming-based systems can have a plethora of dependencies, such as it being difficult to see exactly where a variable is later used and what impact changing it will have on the overall output. Changing the internals of a function or patch can have knock-on effects if other parts of the program depend on the original behaviour. Programming systems, such as Max, make dependencies more explicit by using patch cables (Bellingham, Holland, and Mulholland, 2014a), which show explicit relationships between components. Similarly, classes in object-oriented programming can encapsulate information, and mark certain relationships (i.e. inheritance) explicitly.

To increase the development of high-level structure, the part referencing system was introduced in IGME (this concept is briefly introduced here but discussed at length in Chapter 7). For example, part 2 on track 1 can take its initial content from the output of part 1 on the same track. This facilitates simple repeats or more complex processed-based music. To reduce the complexity of reference parts, dependencies can be highlighted by uni-directional coloured arrows (Figure 6.6). This feature is similar to the patch cable metaphor in Max (Manzo, 2016) and Reason (Duignan *et al.*, 2004). In addition, parent-child relationships can be shown by inspecting the part (Figure 6.7).

6.7. Conciseness / Diffuseness

"How concise is the notation? Does it make good use of space?"

The various levels of hierarchical views in DAWs generally create a concise interface that can be collapsed and expanded dynamically (Nash, 2011). The different input notations offer differing levels of *conciseness* (Nash, 2015).

Both programming and patch-based systems can have both good and bad *conciseness*. Such a trade-off is often a result of a user's interaction and decision-making. In text-based programming, the system can be encoded with minimal semantic information (i.e. single letter variable names). However, Bellingham, Holland, and Mulholland (2014a) note that the verbosity of language should be increased for variable names in coding environments even if this negatively impacts the dimension.

In general, end-user systems should use a concise notation (or interface) where it makes sense to do so (for example, encapsulating information) but add verbosity (i.e. "number of notes to be generated", instead of "num of notes") for reducing ambiguity and increasing *role expressivity*.

6.8. Provisionality

"Is it possible to sketch things out and play with ideas without being too precise about the exact result?"

In the physical world, instruments afford high levels of *provisionality*. For example, opening a piano lid permits an immediate interaction with the instrument, whereas loading up a DAW and a VST is a little more involved. This philosophy extends to software interfaces. For example, Nash (2015) states that digital score notation interfaces are weak for supporting this dimension compared with paper notation, which is far more flexible, as it allows for informal sketching. Programs like Sibelius are beneficial for preparing final scores, but not necessarily for rapid ideation. To mitigate this IGME permits users to rapidly enter note sequences without the restrictions imposed by bar lines or time signatures. Eliminating bar lines in the initial note entry process removes the need for tied notes that cross bars, as notes can simply be displayed as their absolute length. The two-stage editing paradigm takes care of the creation of these formalisms (Figure 6.8).

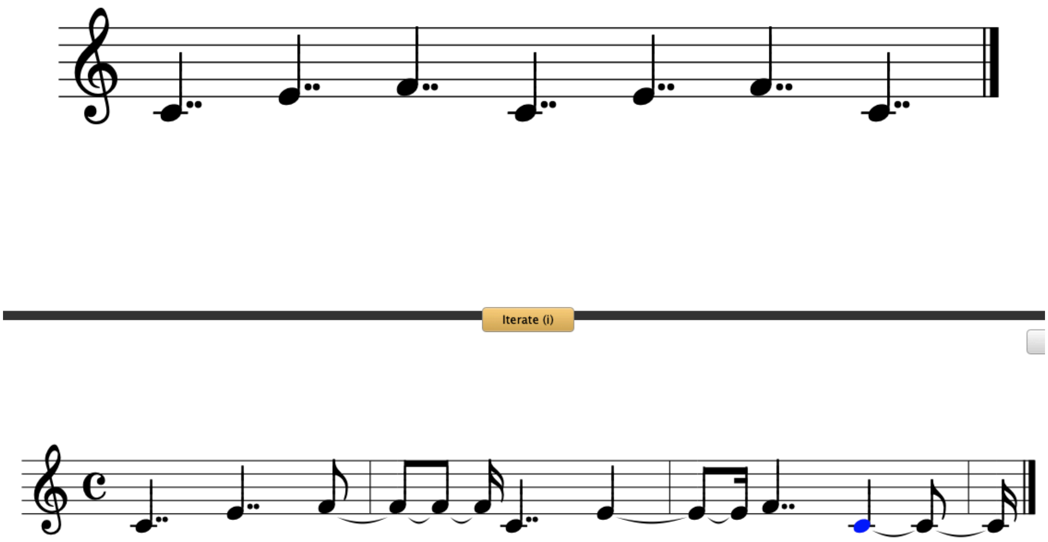


Figure 6.8.: The two-stage editing process 'formalises' the notation.

Systems like Max permit rapid experimentation with little setup, whereas programming systems offer high *provisionality* to experienced practitioners but can be weighed down with documentation and steep learning curves for novices. Programming generally enforces precision and determinism, which is at odds with the above definition of *provisionality*.

Existing computer-generated music software often emphasises *provisionality* also, and can create musical output quickly with few configured parameters. Bellingham, Holland, and Mulholland (2014a) note that Improvisor's (Keller and Morrison, 2007) pre-set algorithms can be used to quickly create musical sketches based on chord progressions. Bellingham, Holland, and Mulholland (2014a) also note that Logic Pro's in-built loops facilitate *provisionality*, as the content can be used as a place-holder and replaced later.

The *scratchpad* feature (Figure 6.5) in IGME supports the creation of parts outside the scope of the arrange view. In most other software, this could only be achieved by storing clips many bars in the future or by creating a new session entirely. This ability to create provisional material is also a feature of PreSonus (Electronics, 2020). Older iterations in a given part can be placed in the scratchpad so they can be later re-purposed.

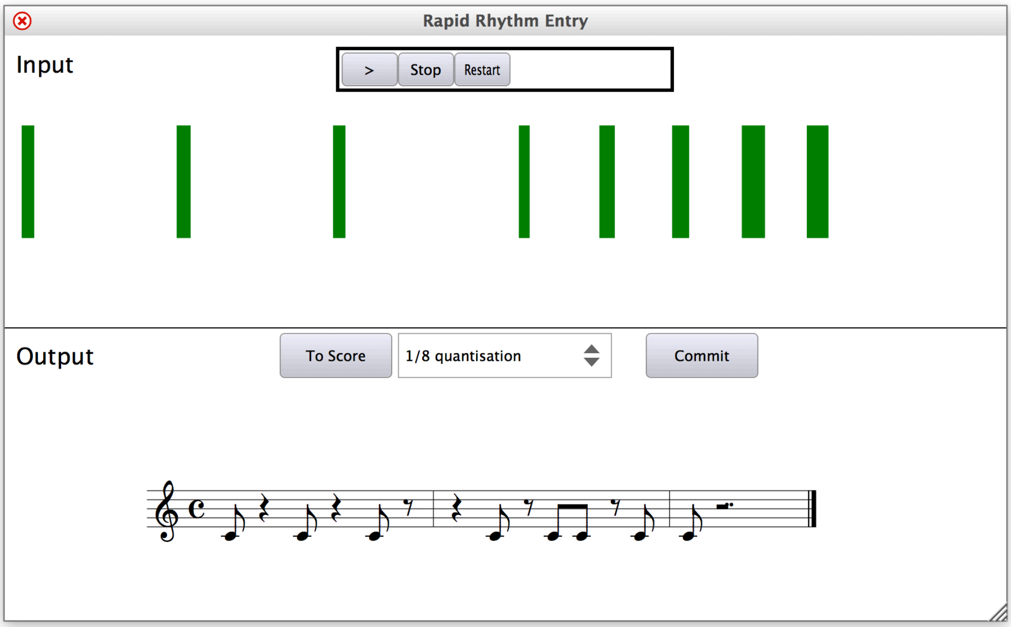


Figure 6.9.: IGME's rapid rhythm entry window allows user to enter rhythm patterns quickly.

6.9. Secondary Notation

"How easy is it to make informal notes to capture ideas outside the formal rules of the notation?"

Nash (2015) notes that handwritten scores have an almost unlimited ability to make informal notes that can be interpreted by the user across the score. Digital score editors make this much harder, due to limited interaction with the keyboard and mouse. Programming environments offer excellent secondary notation through comments (text that is ignored by the compiler).

Bellingham, Holland, and Mulholland (2014a) note that adding colour to elements of an interface can aid a program's usability. Logic Pro can display each track as a different colour. Max allows patch cables to be given different colours, which could, for example, represent different types of signal flow (e.g. MIDI, mathematical, GUI controls). Such roles must however be remembered by the author.

To support sketching, various in-built tools for quickly capturing ideas that could be formalised at a later stage[3] was added to IGME. Figure 6.9 is an example of a technique that uses secondary notation to quickly input material. The "rapid rhythm entry" window allows users to simply tap a rhythm using the space bar. The associated pitches and exact rhythm can be edited later. These

[3] These were used sparingly in the user studies.

techniques support a sketching metaphor (Nash, 2015) where informal ideas can be recorded quickly and effortlessly.

6.10. Consistency

"Where aspects of the notation mean similar things, is the similarity clear in the way they appear?"

Bellingham, Holland, and Mulholland (2014a) note that a consistent interface is easy to learn. For example, most programming language supports the same core principles, i.e. loops, functions and variables that bring coherency and consistency.

A criticism of many existing CGM systems is that they require users to learn a new workflow with which they are not familiar. To mitigate this, and to engage existing music practitioners with CGM systems, familiar interaction paradigms and features should be reused from existing music sequencers. This design philosophy was fundamental in IGME, requiring that its interface is consistent with other music sequencers. For example, the arrange view and edit view are influenced by similar elements in other linear music sequencers. Likewise, editing music in either score notation or piano roll shares many interaction routines. An important principle is that people who are familiar with digital music composition should find it easy to pick up and use IGME. The computer-generated processes operate much like plug-ins, with pre-sets and graphical controls.

6.11. Viscosity

"Is it easy to go back and make changes to the music?"

A highly viscous workflow results in a system that is easy to use and learn (Bellingham, Holland, and Mulholland, 2014a) whereas a system free of restriction provides less structure. Music editing systems should have low levels of viscosity so that changes can be made rapidly and easily, which in turn helps *provisionality* and ideation (Nash, 2015).

Viscosity in programming systems can vary significantly. A well-designed program can make changing components easy, but in a tightly coupled one much harder. Moreover, systems without timelines can make editing music mid-piece much harder, and, in stochastic systems, sometimes impossible (unpredictable).

Attempting to increase the length of a note in existing score editors has knock-on *viscosity* (Nash, 2015), where the resultant effect will often discard notes from the end of a bar or prohibit note entry. Guitar Pro (Arobas-Music, 2020) solves this in a novel way by not removing anything, instead highlighting the bar as an error (in red), requiring a manual fix from the user. The editing stage in IGME has been designed with low *viscosity* in mind. The removal of bar lines for editing notes means users are not required to supply tie lines for notes that cross bar lines, which is an idea borrowed from Dorico (Steinberg, 2020b).

Repetition viscosity (Bellingham, Holland, and Mulholland, 2014a) is where sections of music are copied and pasted to create repeats. When the user wants to update a single copy, the rest must be updated manually. IGME's reference feature can mitigate this, as changing the initial part will trigger all parts that reference it to update. A downside is a slight increase in hidden dependencies, although this is addressed in other ways (section 6.6).

Bellingham, Holland, and Mulholland (2014a) note highly viscous workflows can improve stability and create well-defined use cases. This is not to say the effects are not powerful, but that there are carefully designed interfaces that facilitate a fluid experience for the user. Nash (2015) notes increasing *viscosity* is a trade-off for avoiding *hidden dependencies*, which can be observed in patch and code-based environments.

For an end-user CGM system, some parts of the workflow should be *viscous*, for example, having several computer-generated processes, each with limited degrees of control, that can be combined to provide more complex behaviour. Version control systems help alleviate otherwise

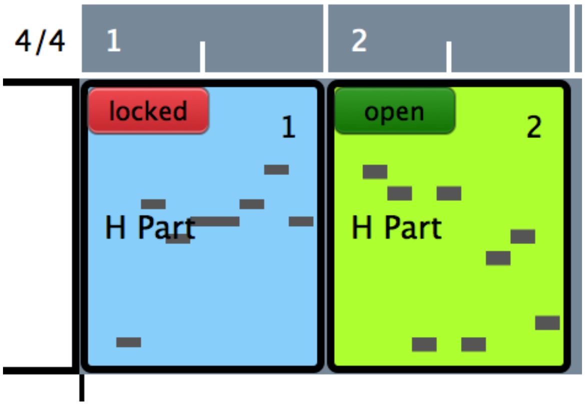


Figure 6.10.: Part 1 is locked and cannot be moved or edited, whereas part 2 is open

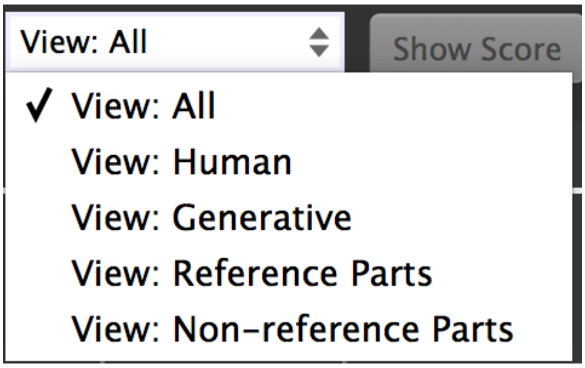


Figure 6.11.: IGME's part highlighting options.

viscous parts of the interface and encourage users to go back and make changes. Sometimes giving users the flexibility of configuring the level of viscosity is useful. For example, the lock feature of IGME (Figure 6.10) prevents the position or internal content of a part being modified, creating a trade-off between *viscosity* and *provisionality*. Therefore, the software does not impose either option; instead, the responsibility lies with the user.

6.12. Role Expressiveness

"Is it easy to see what each part is for, in the overall format of the notation?"

A key requirement of IGME is that each design element makes use of existing metaphors, such as score editing, arranging clips on a timeline, and graphical controls for editing effects, which already offer high *role expressiveness*. This consideration should apply to any end-user music system that wishes to offer unorthodox features (i.e. CGM) to existing users, which in turn allows them to quickly understand the interface Nash (2015).

A key consideration for editing computer-generated music is to ensure the user knows what notes will be altered by any such processes. This is achieved in IGME by changing the colour, so green indicates stochastic events and black shows deterministic ones (Figure 6.3). Also, parts are coloured depending on their use, i.e. blue for a normal part or orange for a purely generative part (Figure 6.2).

Given the explicit roles that computer-generated and human-created music should take, systems should ensure that users can differentiate these roles. For example, IGME allows types of parts to be highlighted or hidden, so users can easily find specific types of parts from the *arrange* view (Figure 6.11). This is significantly more difficult to realise in patch or code-based systems, where the role of each musical element in the final output is not well distinguished or easily separated, and the contribution from human or computer processes is tangled. Such a use case may be desirable for certain compositions and, might be easier to realise with programming approaches, but this is not the focus here.

6.13. Premature Commitment

"Do edits have to be performed in a prescribed order, requiring you to plan or think ahead?"

The timeline metaphor in DAWs encourages a linear left-to-right workflow and supports various forms of development, including part-by-part, bar-by-bar or top-down arrangements (Nash, 2015). Likewise, programming systems support various ways of working but often necessitate the use of object-oriented programming (i.e. inheritance) requiring some *premature commitment*.

Bellingham, Holland, and Mulholland (2014a) stress the importance of having an option that states *"I don't know what is going on here"*. Timeline-based interfaces support this, whereas more specialised CGM systems, without this concept, are weaker in this regard. In general, code-based systems are not supportive of more structured or orchestrated compositions. Nash (2015) notes that an advantage of using an arrange view metaphor is that it allows musical parts to be easily inserted, moved, and copied.

It is unclear if the user of parent-child relationships in IGME (through reference parts) increases premature commitment. This is discussed further in Chapter 11.

6.14. Error Proneness

"How easy is it to make annoying mistakes?"

Annoying mistakes are not desirable for any software or interaction. With most CGM systems, the processes themselves can generate *mistakes*[4]. With a subjective topic like music, it is important to remember that what might be deemed annoying mistakes by one composer can be considered wholly acceptable by another.

[4] For example, inserting an out of key chord could commonly be considered a mistake.

6.15. Closeness of Mapping

"Does the notation match how you describe the music yourself?"

Nash (2015) notes that DAWs score highly in this dimension due to having interaction paradigms based on recording studio workflows. Programming-based interfaces are often dissimilar and therefore score badly here. For example, a collection of variables (i.e. note, length and onset) representing a MIDI note is less *'close'* to its expression in a piano-roll editor.

6.16. Abstraction Management

"How can the notation be customised, adapted, or used beyond its intended use?"

Green and Blackwell (Carroll, 2003) describe three classes of system: abstraction-hungry, abstraction-tolerant, and abstraction-hating. Code and patch-based systems rely heavily on user-defined abstractions which permit the development of complex and nuanced CGM systems, not permissible or feasible by other interface types. This reliance on abstraction design can often have a negative effect on usability (Bellingham, Holland, and Mulholland, 2014a). Providing users with pre-built abstractions (graphical interfaces) rather than with the components for building them lowers the barrier to entry for CGM.

Bellingham, Holland, and Mulholland (2014a) suggest that *"An effective design would be for the software to have a low abstraction barrier but be abstraction-tolerant. Such a design would allow new users to work with the language without writing new abstractions, while more advanced users could write abstractions when appropriate."*

In general, IGME's inbuilt processes are abstraction-hating, as they cannot be customised internally but can be controlled only through the exposed graphical controls. IGME encapsulates sequences of events into parts that can have further processes applied to them, and can reference and reuse each other. Overall, the system is abstraction-tolerant. It is unclear, without more conclusive user studies, how powerful IGME's inbuilt generative features and part-referencing will be if there is no ability to create more complicated abstractions.

6.17. Conclusion

This chapter has demonstrated that the CDN is a powerful framework that permits broad comparisons to be made between opposing interfaces while providing a common vocabulary for discussing them. It has shown how some of the issues associated with computer-generated music systems can be mitigated by respecting established music sequencing workflows (DAWs), and by eliminating some of the drawbacks of programming based approaches[5]. For example, having a timeline increases *visibility* and *progressive evaluation* for which programming systems are often lower in these dimensions.

The CDN is not without limitation. Despite having 15 discrete dimensions, it omits one deemed critical to this work - *learnability* Simply put, this is,"how easy is it to learn the interface". Learning to program is hard (Guzdial, 2010), and acts as a barrier to entry for engaging with CGM, whereas using graphical interfaces democratises tools. Learnability is more formally evaluated in Chapter 10.

The broad-brush comparison in this chapter of existing interfaces results in a list of desirable design hypotheses (a CDN profile), given under each dimension (shown in Table 6.2). IGME was therefore designed around these requirements as a *software hypothesis* (Leinonen, Toikkanen, and Silfvast, 2008) which in turn allows them to be formally assessed in Chapter 11, once the software has been evaluated through user studies. These design hypotheses result in a set of design heuristics applicable to many different interfaces for engaging with CGM.

[5] Patch and code-based systems have many advantages but are not a focus of this research

Dimension	Design Hypotheses
1. Visibility	Minimal layers of hierarchical views. Ensure that the role of each musical element is clearly defined visually.
2. Juxtaposabillity	Allow different variants of the music to be swapped quickly and easily. Make use of version control technologies.
3. Hard Mental Operations	Focus on using, rather than designing, computer-generated processes. Use a timeline to make control flow simple to predict.
4. Progressive Evaluation	Ensure that rapid edit-audition cycles are supported.
5. Hidden Dependencies	Minimise hidden dependencies, and ensure that any required dependencies can easily be visualized.
6. Conciseness	Give controls verbose names.
7. Provisionality	Ensure that the system can rapidly ideate. Support this by using version control technologies. Consider adding a secondary timeline (scratchpad).
8. Secondary Notation	Allow for informal notes to be added to various musical components (for example, track description).
9. Consistency	Use common design and interaction paradigms found in other music software (for example, clips, tracks, and timelines).
10. Viscosity	Allow users to lock down the position or final output of the music. Ensure that it is otherwise easy to go back and make changes.
11. Role Expressiveness	Distinguish between computer-generated and human-composed music.
12. Premature Commitment	Use an arrange view metaphor, and support part-by-part, bar-by-bar or top down arrangements.
13. Error Proneness	The computer should not automatically fix musical errors, but instead, allow users to judge and fix these for themselves.
14. Closeness of Mapping	Focus on graphical rather than textual controls.
15. Abstraction Management	Can be used without creating abstractions, but allow advanced users to develop their own abstractions.

Table 6.2.: A list of design heuristics under each dimension in the Cognitive Dimensions of Music Notations framework used in this chapter.

7. IGME: Interactive Generative Music Environment

This chapter discusses the internal architecture of IGME and provides a vocabulary for discussing its features in future chapters. This chapter details IGME in its final form and contains excerpts from an amalgamation of Hunt, Mitchell, and Nash (2017b, 2018, 2019, 2020). Notable revisions made to IGME after its evaluation in pilot studies are set out in Chapter 8.

IGME is a platform built purely to test research theories and is a means to an end, rather than a research contribution. It is hoped that other practitioners will integrate elements of this research into their products/work. The considerable time spent developing IGME was justified by the need for an end-user platform integrating with complex and nuanced real-world workflows. Incomplete or simplistic proof-of-concept tools offer no meaningful insight into interaction issues and need to be tested extensively and put to robust use.

This could explain why there is limited research on macroscopic interaction in music software. As researchers need to control many aspects of the study, simply extending an existing open source project or using an API (for example, Max for Live) would not achieve the same outcome, and in many cases not permit the integration of interaction logging technologies (discussed in Chapter 8).

7.1. IGME

IGME supports the exploration of computer-generated music techniques, through a user-friendly interface built around common digital music software paradigms. As summarised in Chapters 3 and 4, many existing computer-generated music systems use workflows that are not familiar to non-programmer composers. IGME is designed so it can be quickly learned by composers already familiar with the basic concepts of digital music sequencing and notation, allowing them

Feature	Rationale
Uses existing music sequencing paradigms and design metaphors.	Creates a low-entry threshold for composers so they can focus on using computer-generated techniques rather than learning new workflows.
Provides a full version control system, for tracking edits and iterations.	Fundamentally required for stochastic processes which produce output that differs each time and is then easily lost or overwritten.
Uses graphical widgets, rather than code-based interfaces.	Programming requires learning a new set of skills, whereas graphical widgets are a fundamental part of existing music software.
Takes a modular approach to composition while retaining a linear timeline.	Code and patch-based systems do not have a concept of a timeline, making control flow difficult to predict. A simple linear timeline makes this much easier.
Uses a multi-layered assembly stage to create the final score from individual parts.	Breaking music into parts ensures a clear distinction between human and computer-composed content.

Table 7.1.: Design principles for end-user computer-generated systems.

to experiment immediately with computer-generated processes. IGME follows the *'low threshold'* (easy to get started), *'high ceiling'* (permits sophisticated ways of working) and *'wide walls'* (can be explored in multiple ways) design principles for supporting creative thinking set out by Resnick *et al.* (2005). The five core design principles of IGME are detailed in Table 7.1.

7.1.1. Technologies

IGME is built using JUCE (Roli, 2016), a cross-platform C++ framework that is well-suited to music software development. Two external libraries, FluidSynth (Henningsson and Team, 2011) and GUIDO (Hoos *et al.*, 1998), are also used. FluidSynth provides a sound-font synthesiser to produce all the inbuilt instrument effects and is populated using a sound-font supplied from MuseScore (Watson, 2018). GUIDO, a score-drawing framework, provides the Western score notation interfaces used in this project. IGME contains a sub-application called IGMESynth that is responsible for producing audio.

IGME is distributed as a closed source application, as many conditions needed to be controlled academic study. This also protects the inbuilt logging software from being inappropriately re-purposed with data being sent elsewhere. The IGMESynth is open source with a GNU LGPL (Almeida *et al.*, 2017) licence and is distributed alongside IGME, as Fluidsynth's (compiled

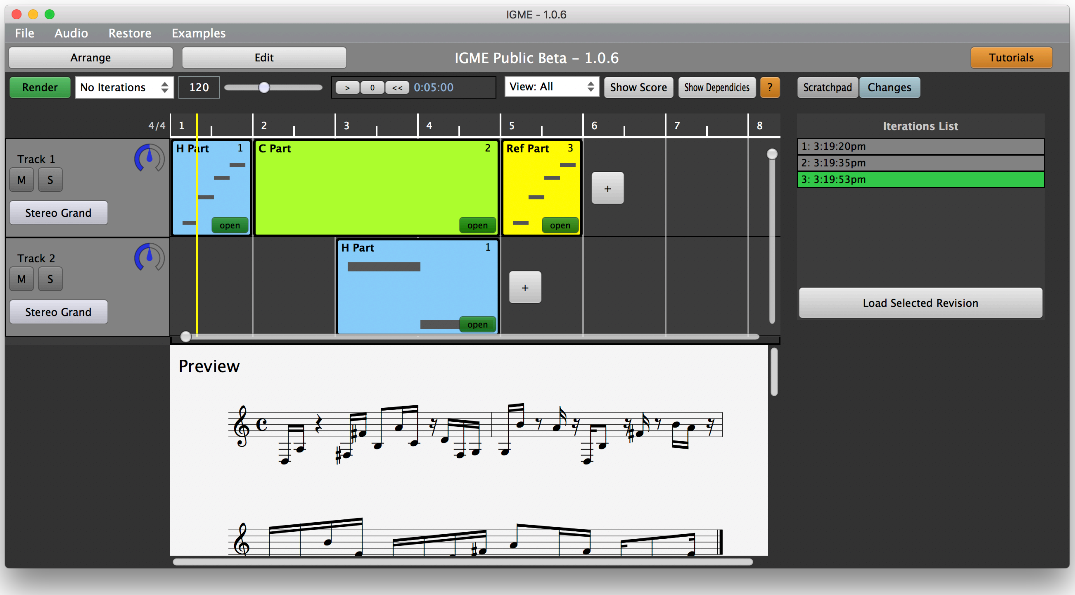


Figure 7.1.: IGME's arrange view where parts are arranged across the timeline. **Middle:** Timeline editor. **Bottom:** Part preview. **Right:** version control system/scratchpad.

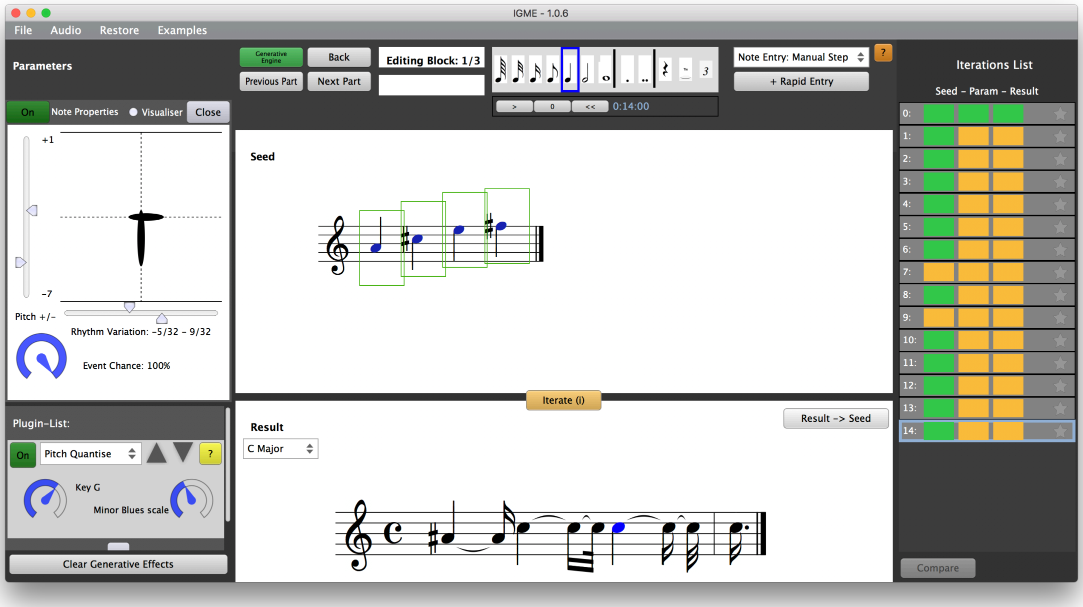
within IGMESynth) licence prohibits its use in closed source software. IGME and IGMESynth communicate over open sound control (Freed, 1997), with IGME being the parent application that launches IGMESynth as a child. Both JUCE and GUIDO permit closed source development. IGME was originally built for OSX, later ported to Windows, and remains cross-platform.

7.2. Arrange View

IGME is fundamentally a linear music sequencer that promotes arranging parts[1] (sequences of notes) of music onto a timeline of different tracks, sharing many parallels with existing music software. Figures 7.1 and 7.2 show IGME's arrange and edit views. To produce output, a *'render'* stage is added, this collates and sequences parts on the timeline to produce a single multi-track sequence (illustrated in Figure 7.3). Although this introduces an additional dependency, it has two distinct extra uses specifically for computer-generated music, as discussed at the end of this section.

Music is arranged on a linear timeline of tracks, whereby the order of the music is explicit (i.e. left to right). This is in contrast with code or patch-based systems which can be difficult to predict the order of events (Nash, 2015). The scratch-pad editor is an additional timeline/track that

[1] As mentioned previously, parts are metaphorically similar to MIDI clips found in other music software but have a more complex role.



Editing Block: 1/3

Note Entry: Manual Step

Rhythm Variation: -5/32 - 9/32

Event Chance: 100%

Key G

Minor Blues scale

C Major

	Seed	Param	Result
0:	🟩	🟩	🟩
1:	🟩	🟧	🟧
2:	🟩	🟧	🟧
3:	🟩	🟧	🟧
4:	🟩	🟧	🟧
5:	🟩	🟧	🟧
6:	🟩	🟧	🟧
7:	🟧	🟧	🟧
8:	🟩	🟧	🟧
9:	🟧	🟧	🟧
10:	🟩	🟧	🟧
11:	🟩	🟧	🟧
12:	🟩	🟧	🟧
13:	🟩	🟧	🟧
14:	🟩	🟧	🟧

Figure 7.2.: IGME's edit view where individual parts are edited. **Left:** plug-in editor. **Centre:** note editor. **Right:** version control system.

allows a composer to develop individual parts offline without worrying about their location within the overall musical structure. Users can copy and paste between the timeline and scratch-pad, additionally using it as *'cache'* for ideas discussed further in Chapter 6.

7.3. The Seed, Parameter, and Result Model

Parts in this project are defined as the core musical elements that are sequenced together to produce a musical artefact (Figure 7.3). Parts have explicit roles that differentiate human-composed and computer-generated music, facilitating an interactive generative composition process. These parts are referred to as: human; human-computer; computer; reference; iterator; and repeater. Table 7.2 summarises these.

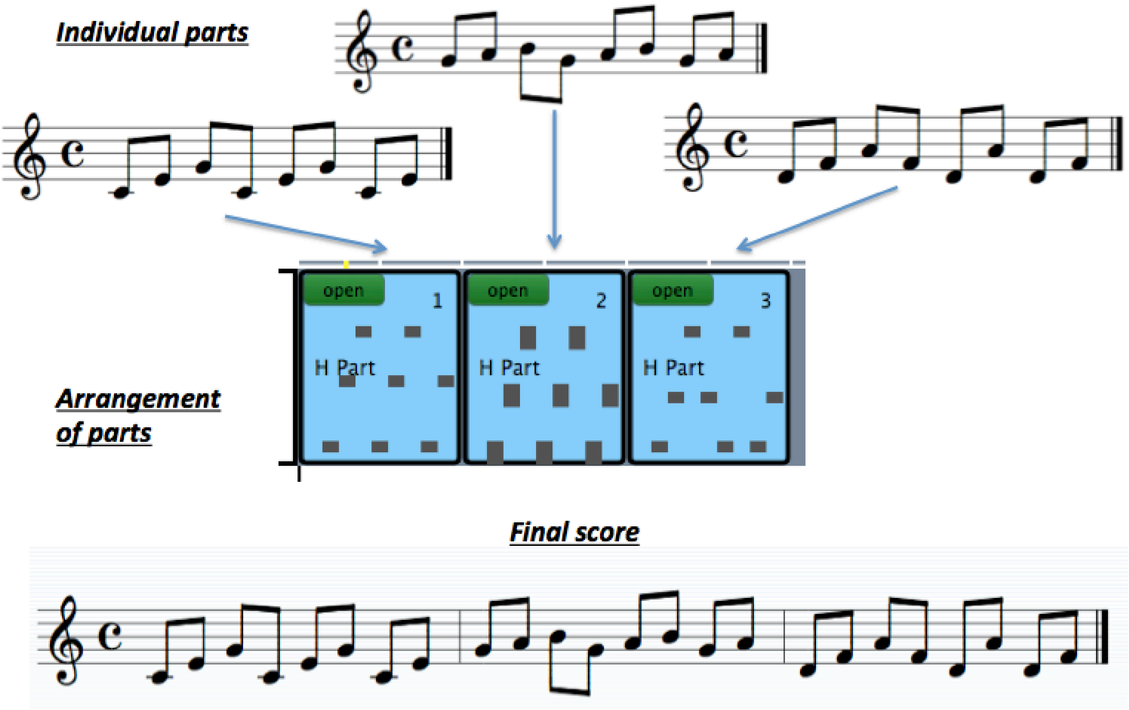


Figure 7.3.: The part arrangement process.

Group	Part type	Input (seed)	Can apply plug-ins
Regular parts	Human	Edited by user	Yes
	Human-computer	Edited by user	Yes
	Computer	Computer generated	Yes
Reference parts	Reference	From another part's seed or result	Yes
	Iterator	Another part's output	No
	Repeater	Multiple part's output	No

Table 7.2.: A summary of the different part types.

A part has three distinct sub-components: the seed, parameters, and result (Figure 7.4). The *seed* is the music material edited by the composer. The *parameters* are a series of computer-generated effects (plug-ins) that are applied to the seed material. The *result* is simply the outcome of the parameters applied to the seed material, and is the music sequence auditioned by the user. Should the composer not configure any parameters, the result will be a carbon copy of the seed. In this situation the part is referred to as *human part*, whereas a *human-computer part* would contain computer-generated effects. In a *computer part*, the seed comes from a seed generator using a secondary set of plug-ins. The result can be converted back into the seed, permitting a recursive

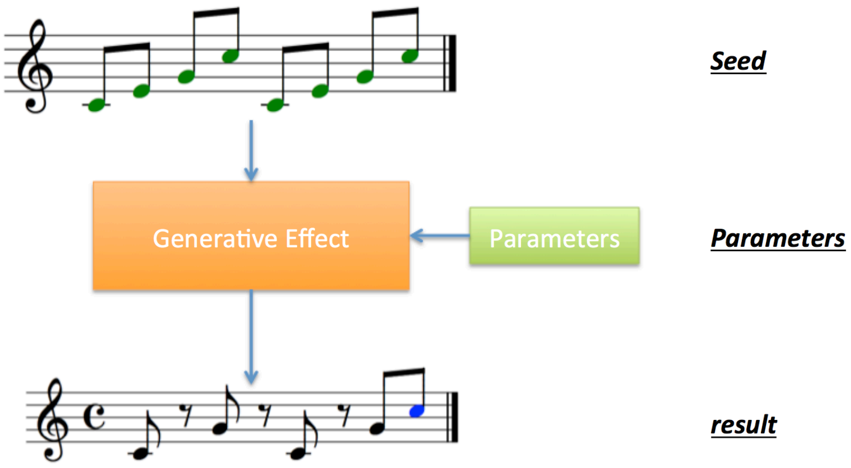


Figure 7.4.: The seed, parameter, and result model.

editing process. Similarly, a computer part can be converted into a human-computer part. In both cases, the output can be further edited (fine-editing)[2].

Note List

IGME internally encodes musical events into a double-linked list[3] container. Each event contains the following values: note, velocity, length, timestamp, bar number and unique-id, as well as a dynamic container for storing specific notation information. This additional metadata supports rapid music analysis. Every time a new event is added to a sequence, it receives an incremental unique-id. Editing a note does not update this id. A part contains two such sequences, one for storing the seed and one for storing the result.

Reference Parts

One way of encouraging the development of higher-level structure and repetition is to use *references*. Rather than each part having a unique seed, parts can base their *'seed'* on the *'seed or result'* of another part. The new part is referred to as a *reference part* type. There are two referencing methods: pre-reference and post-reference. If the pre-reference method is used, the current part's seed is taken from another part's seed. If the post-reference is used, the current part's seed is taken from the result of another part. An example of pre/post reference is shown in Figure 7.5, which

[2] The computer's generative role is no longer required, and the users take ownership of the music. This concept was used repeatedly in the user studies of Chapter 10.

[3] Inserting, deleting, and processing sequences is therefore made simple, compared with other data structure types.

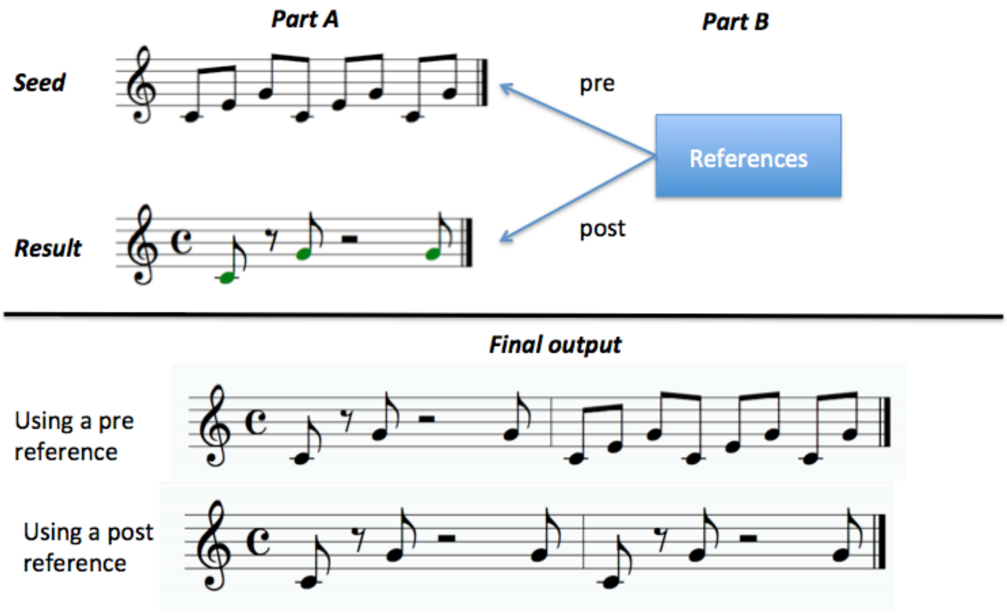


Figure 7.5.: Pre vs post referencing.

illustrates the difference in final output when part B references part A, using either pre or post referencing.

Parts that use references prevent the composer editing the seed material, but they are still permitted to apply generative effects. A key advantage is that the composer can update the original part (A), and those changes will be propagated to all the other parts that were referencing it (B). If the composer wishes to edit a reference part's seed, copying and pasting from the original part would be more appropriate. Reference parts appeared confusing for composers during user studies, but also led to interesting compositional choices. These concepts are discussed further in Chapter 10. Based on feedback from pilot studies (Chapter 7), two more part types were added to IGME using the same principles as references, the *iterator* and *repeater*.

An *iterator* part can be set to iterate either a human-computer or a computer part, allowing another iteration of a part to be placed on the timeline. This could be used to produce multiple versions of a given effect, resulting in several parts with similar musical characteristics.

Repeater parts allow the user to repeat previous bars of music that can contain any arbitrary amount of part types. Simply put, a repeater part repeats content already on the timeline. For example, a setup to repeat bars 1-3 will take the result of parts 1 and 2 and repeat them. Should parts 1 or 2 change, so will the output of the repeater.

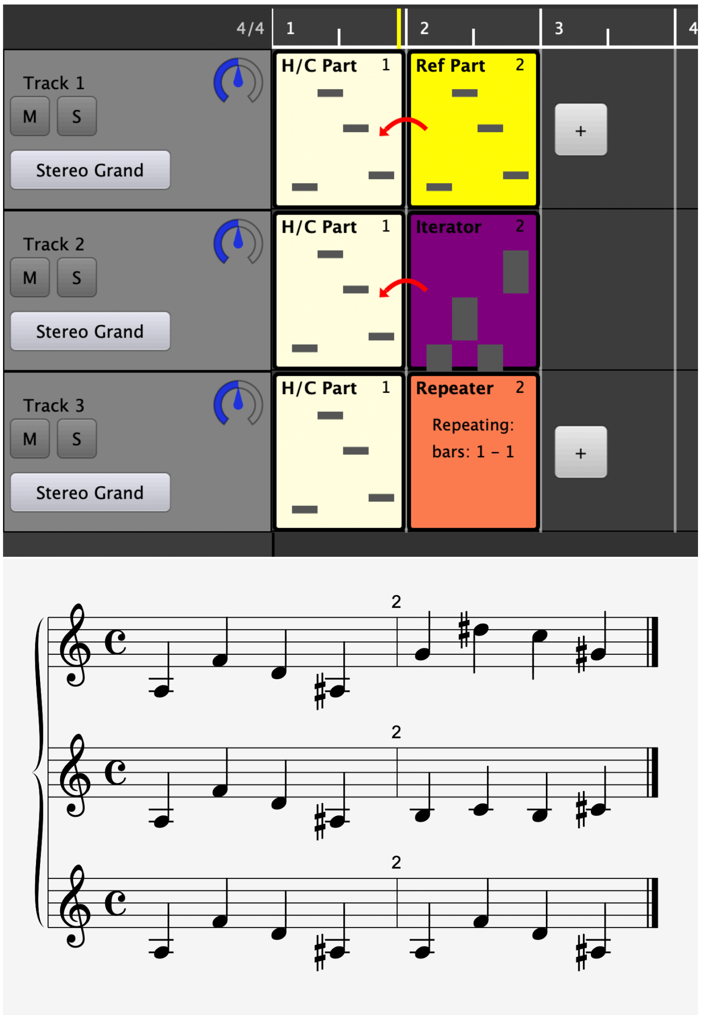


Figure 7.6.: **Top**: reference part (track 1), iterator (track 2), and repeater (track 3). **Bottom**: Output of the above processes.

The differences between these constructs are illustrated in Figure 7.6. Track 1 uses a reference part, with the second part applying a randomise effect. Track 2 uses an iterator and therefore part 2 is another variant of the effects set-up in part 1. Finally, track 3 uses a repeater, meaning bar 2 is the same as bar 1.

During this PhD, supplementary research in studying repetition was conducted and published by (Hunt, Mitchell, and Nash, 2019; Hunt, 2020), making use of IGME and the reference part system. In summary, a large proportion of existing music can be encoded within IGME, using the reference parts explicitly to show this repetition. If existing music can be encoded in this way we propose that these mechanisms are useful for both regular music composition and for adding explicit structure to computer-generated music pieces.

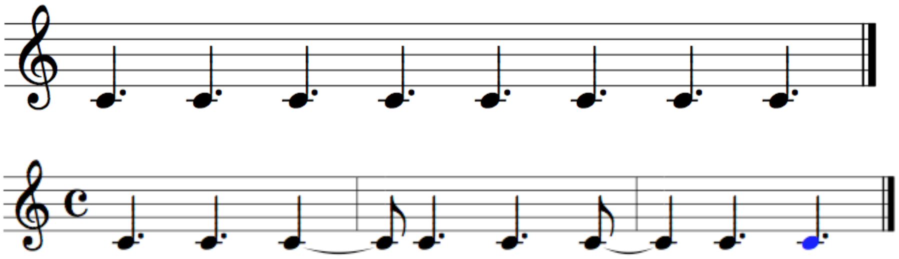


Figure 7.7.: **Top:** the seed. **Bottom:** result. The result automatically applies the formalism of 4/4.

7.4. Editor View

IGME's edit view is where parts are edited and generative effects configured (Figure 7.2). When editing the basic music material (seed), the composer can avoid the formalities of working with bar lines, as IGME automatically adds bar lines at the result stage (Figure 7.7). A composer is, therefore, free to edit and arrange sequences with few constraints.

Secondly, when working with built-in computer-generated effects, the input (seed) material is processed using various effects before becoming the output (result) music. This editing paradigm is referred to as the two-stage editing process and enables a rapid edit-audition cycle[4] (Nash, 2011). Users can toggle between piano roll notation and score notation in either the editor or output window. The need to add piano roll support was identified through user studies and is discussed in Chapter 8.

The two-stage editing process enables users to work easily with computer-generated music. The content of the music (the result) is dynamic. Editing the seed is simply a blueprint, with the final result computed and subsequently auditioned when an iteration is created (Figure 7.2). The two-stage editing process shows the before and after stages visually, revealing mappings and dependencies between seeds, parameters and results (permitting verbose debugging). This should help composers learn about computer-generated processes, and develop a cognitive understanding of each process.

7.5. Version Control

IGME contains two independent, but related, version-control systems (VCS). Each part has its own local VCS that tracks independent variables, such as the contents of the seed, the configuration of computer-generated effects, the resultant output, and any references. A new version is added to

[4] Discussed at length in Chapters 9 and 10.

the VCS when one of those four variables (seed, parameters, result, or reference state) changes as a result of the composer creating an iteration. A composer can easily recall a previous version. There is no provision for deleting unused versions, but composers can mark *'favourite iterations*[5]*'*. The second VCS works on the arrange view and tracks the arrangement of parts on the timeline. When a composer renders the arrange view, if the content of a part, or a part's position on the timeline, has changed, it is registered automatically as a new version.

The VCS supports composers working with stochastic music, permitting them to rapidly create permutations of the music and, if they wish, to recall earlier versions. This rapid edit-audition cycle contributes to a high state of flow, which is desirable for composers engaging with creative exercises such as music (Nash and Blackwell, 2012). In essence, the VCS is a key component for any end-user generative music composition system (evaluated in Chapter 9). Crucially, the VCS provides the ability both to recall and track the compositional process. The methods for sending, collecting and analysing each entry in the arrange-level VCS are discussed in Chapter 8.

7.6. Computer-Generated Processes

As mentioned in Chapter 4, existing computer-generated music systems require a composer to be literate with programming languages, or to learn an alien interface (Bellingham, Holland, and Mulholland, 2014a). IGME gives access to the underlying computer-generated effects engine through a series of simple graphical interfaces. One downside is that composers are unable to design their own effects or processes (abstractions), although there is scope to permit this in future versions. Though accessible to novices, IGME may appear too shallow for more experienced composers.

The three different types of computer-generated processes are: note properties, seed generators, and effect plug-ins[6]. The note properties panel (added late in development) provides a method for applying stochastic properties to individual notes (discussed further in Chapter 7). Seed generators (Figure 7.9) generate seed material using more complex generative effects (e.g. transition table, or L-system). Effect plug-ins process entire sequences of notes by, for example, reversing the input order or by quantising to a scale (see Figure 7.8). Human-computer parts can

[5] This feature was added through various pilot studies and is discussed more in Chapter 9.

[6] These are refereed to as type A, B, C respectively in appendix item D

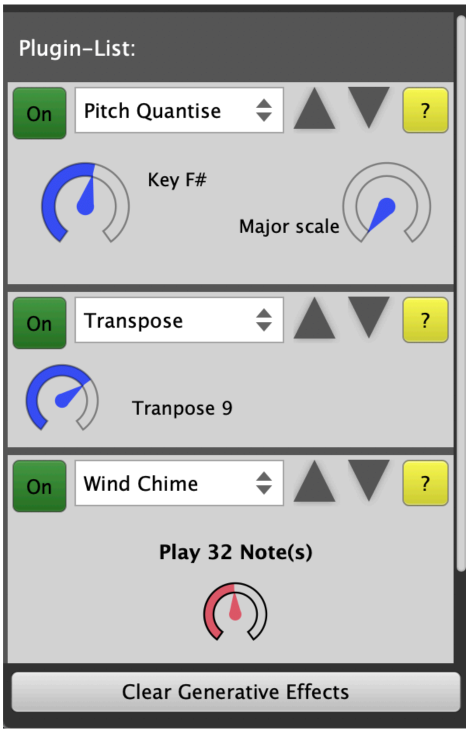
contain both note properties and any arrangement of effect plug-ins (both optional). A computer part always contains a seed-generator, but may also contain effect plug-ins.

Some parts use a generative model, whereby the data provided for the model is formed through analysing other parts on the timeline. An example is the *distribution sample* plug-in, which uses a series of slider-banks to show the distribution of music characteristics, such as pitch and rhythm. Several options for determining what informs the analysis are presented to the composer. Appendix D provides more information on the implementation details for each computer-generated effect.

Iteration and playback appear coupled but actually serve different purposes in IGME. Iteration causes any computer-generated effects to be recomputed producing a new version, whereas playback simply auditions the last or selected version. When editing parts, an iteration will also induce playback. There are three options at the arrange view level. By default, playback (option 1) is selected, which simply takes each version, in each part, and sequences them together to produce a song. The composer can also compute iterations (option 3) in the arrange view editor. This in turn goes through all parts on the timeline and produces a new iteration on each. Individual parts can be locked ensuring their content remains fixed regardless. Option 2 is used to update the reference part, such as when a parent dependency has changed. Although this process could be done automatically, the option is determined by the user.

7.7. Conclusion

IGME was developed as our proposed solution for working with generative music. This chapter is a cursory look at its core features, and appendix D gives further detail on all the computer-generated processes. IGME was developed iteratively under a mostly participatory design umbrella. This involved pilot studies and subsequent redesigns. A timeline of changes is discussed in Chapter 8, along with a description of the methodologies for studying user interaction.



Pitch Quantise

Key F#

Major scale

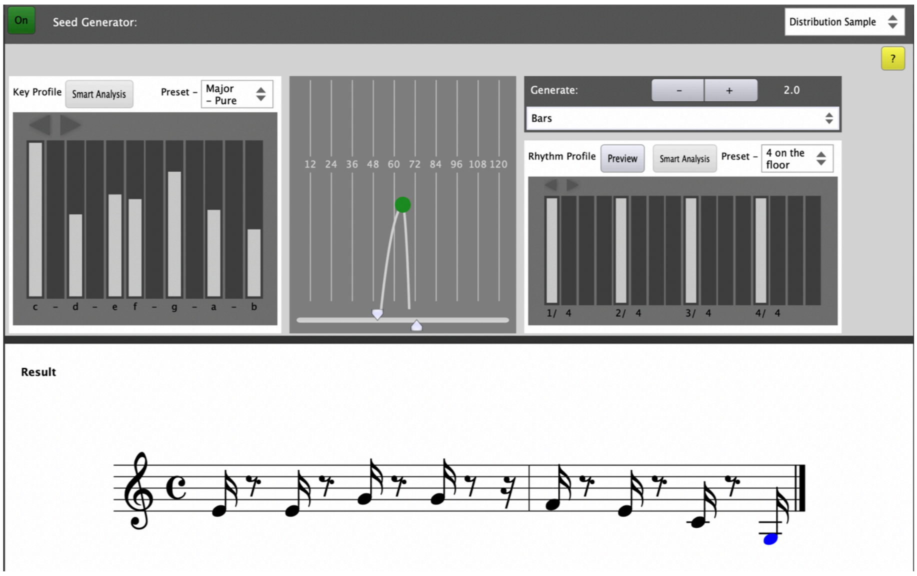
Transpose

Tranpose 9

Wind Chime

Play 32 Note(s)

Figure 7.8.: Three different computer-generated plug-ins are chained up.



Distribution Sample

Preset – Major – Pure

Generate: 2.0

Bars

Preset – 4 on the floor

Figure 7.9.: One of the seed-generator processes (distribution sampler).

Part IV.

EVALUATION

8. Methodologies, Pilot Studies, and Participatory Design

The ultimate purpose of putting IGME in front of users was to evaluate it as a paradigm for computer-generated music composition, testing its novel features and discovering what a computer-generated music workflow entails. Several methodologies, used by different groups of users, were deployed in the evaluation. These can be broken down into: initial pilot studies (participatory design); 1-hour workshops; video study (longitudinal); public beta; and remote study. In all the studies, interaction data from the IGME software was logged and sent remotely to the researchers. All studies involved a formal survey (discussed shortly). Table 8.1 summarises the various studies.

8.1. Interaction Logging

Interaction logging is common in the field and is often used to measure usability (Brown, Nash, and Mitchell, 2017). An example of this includes, Jeong, Kim, and In (2020) who took the novel approach of logging images representing the user's screen (GUI), resulting in interaction similar to that of recorded video but with far less bandwidth. Lettner and Holzmann (2012) take an automatic approach that analyses applications in real-time and reports on changes to GUI elements and their life cycle.

Nash's (2011) work studied composers interacting with music software, using interaction logging-based approaches. Summarising the benefits of this, Nash states: *"Significantly, the non-invasive logging of interaction enabled the study of real-world creativity, without interfering with the individual's creative process or intruding in their environment. In this capacity, the Internet has proven a powerful tool that can be instrumental in the remote observation of subjects in creativity research."*

Study Type	Data Collected	Questioniare type	Conditions	Length	Date	IGME Version	Sample Size
Initial pilot study	Basic interaction log.	A	Controlled	1 hour	03/2018	0.8	9
Second pilot study	Basic interaction log.	B	Controlled	1 hour	09/2018	0.9	10
1-hour workshops	Interaction log and music.	B-2	Controlled	1 hour	11/2019	1	23
Public beta	Interaction log and music data (optional).	B-2	Uncontrolled	n/a	06/2019	1.0-1.1	*n/a*
Video study	interaction log, music, screen recording.	B-2 and C	Controlled	4 hours	03/2020	1.06	4
Remote study	interaction log and music data (optional).	B-3	Uncontrolled	2 hours	10/2020	1.2	31

Table 8.1.: A table summarising the studies completed in this research.

Gerken *et al.* (2008) observes that interaction logging does not answer all the usability questions and in some cases probably raises new ones[1]. In summary, interaction logging should be used in conjunction with additional qualitative methods. For this reason, and others, the research here takes a mixed-methods approach for studying IGME.

8.1.1. Implementation and Analysis

IGME contains a sophisticated inbuilt interaction logging system, which provides a fine level of granularity - reconstructing in sequential order and in detail, the users' actions[2]. Each message includes a timestamp value (millisecond resolution). These interactions are logged to a text file and sent remotely to a server. Each file also contains the following values: user-id, session-id, system-id, and version number. User-id[3] is the product registration key prescribed to the user during registration, which can be looked up by the researchers to concatenate initial sign up, consent forms, user data, and survey forms. Session-id is generated each time IGME starts up.

[1] Indeed the interaction data collected during Chapter 9 did, in fact, raise more (unanswerable) questions

[2] A list of interactions can be found in appendix item E.

[3] The user-id is meaningless to anyone other than the researchers as it is an arbitrary value.

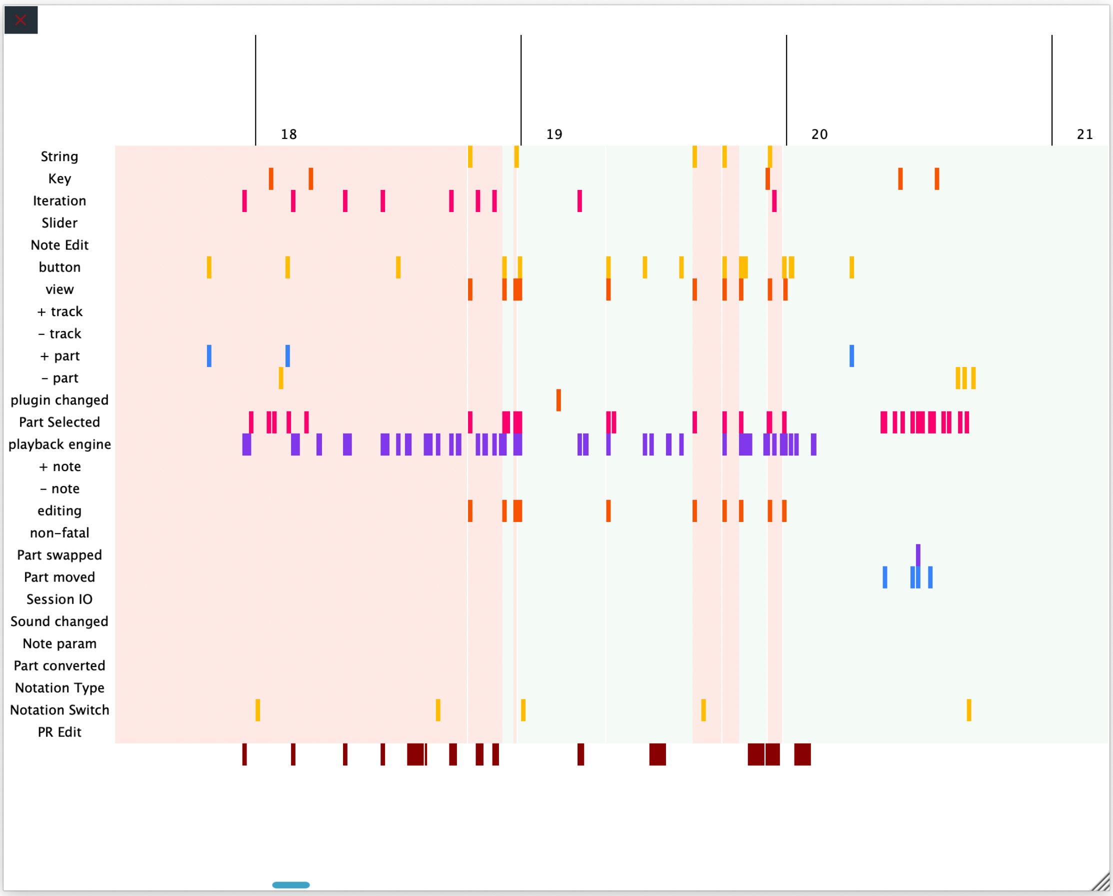


Figure 8.1.: IGME interaction data analyser. Each row represents an type of data, the black lines signify minute markers. The background colours determines what view the user was in.

System-id is an MD5 hash (Rivest and Dusse, 1992) of system information and is used to distinguish registration information shared between multiple systems.

In addition to the interaction data, each iteration in the arrange-level version control system (music data) is sent. These are tagged with the same header data as the interaction data. A clock within IGME triggers every minute (and on shut-down/start-up), sending any new versions of either data type via an HTTP post request to a server. Data is cached locally, so that any interruption in internet connectivity can be mitigated, with data sending resuming automatically once a connection is re-established.

A small tool was developed for inspecting and analysing the interaction data (see Figure 8.1). This tool was built using the concepts set out in the visual information-seeking mantra, (Shneiderman, 1996; Craft and Cairns, 2005), giving an overview of the data and the ability to zoom in and filter details, and discover relationships between interaction types. Data types were divided into rows, with colour indicating that an event had been logged. This allowed for a cursory look at what was otherwise arbitrary data. Figure 8.1 shows an overview, with data flowing from left to right, and the black markers signifying each minute. A red background indicates the user was in the editor, while green indicates the user was in the arrange view.

8.2. Surveys

Three types of surveys were developed and used in the research: preliminary evaluation (A), cognitive dimensions evaluation (B), and prior experiences of music (C). Templates for each survey are included in Appendix B.

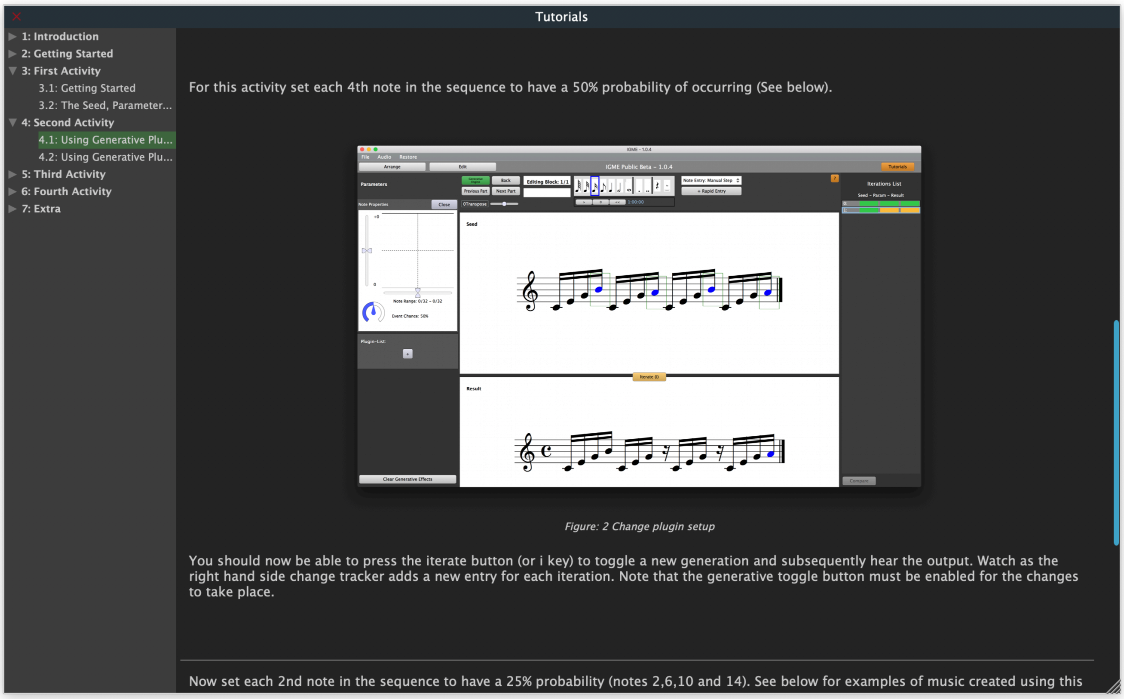
Survey A: Preliminary Evaluation

Survey A was used only for the initial pilot study and participatory design workshops. The short survey was designed to capture basic information for evaluating the research objectives. It asked participants: how much experience they had of different music software; their experiences of computer music techniques[4]; what they had found enjoyable and frustrating; and, what they thought could be improved within IGME.

Survey B: Cognitive Dimensions Evaluation

Survey B, the most commonly used survey, was split into two main themes. The first evaluating the various features of the software and the latter for evaluating a user's perceptions of engaging with computer-generated music techniques. Questions in the first theme were aligned (where possible) to an aspect of the cognitive dimensions of notations framework. Work by Nash (2011) used a similar methodology for comparing experiences of users interacting with music trackers, and evaluating the software under the cognitive dimensions. Questions in the second theme were more open-ended, gathering qualitative data. The survey was amended twice - these changes are discussed in section 8.5.1.

[4] The same as questions 3-4 from Appendix A



For this activity set each 4th note in the sequence to have a 50% probability of occurring (See below).

Figure: 2 Change plugin setup

You should now be able to press the iterate button (or i key) to toggle a new generation and subsequently hear the output. Watch as the right hand side change tracker adds a new entry for each iteration. Note that the generative toggle button must be enabled for the changes to take place.

Now set each 2nd note in the sequence to have a 25% probability (notes 2,6,10 and 14). See below for examples of music created using this

Figure 8.2.: IGME's Inbuilt tutorial system (also available via a website).

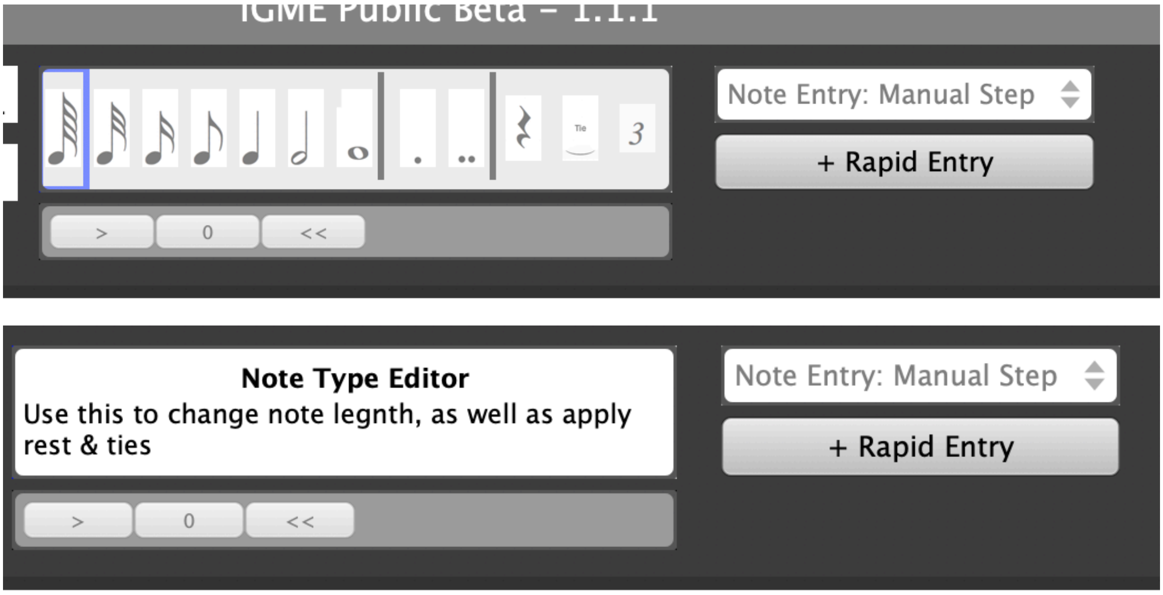
Survey C: Prior Experiences of Music

Survey C was used to uncover an individual's musical background and was presented to participants who engaged in the longer studies. This was a reworked version of the survey used in Chapter 5 for understanding the experiences of music practitioners.

8.3. Tasks

IGME contains a range of inbuilt tutoring systems, including tutorial exercises (Figure 8.2), a dynamic interactive help system (Figure 8.3), and common usability features (i.e. cursor highlights). The inbuilt tutorials are a step-by-step guide to using IGME[5]. Although free to do as they wished throughout, participants in all studies were guided towards the tutorials in order to familiarise themselves with the system, with any remaining time spent experimenting. This created a balance between *'a specific task'* type study and *'open exploration'* (Brown, Nash, and Mitchell, 2017). The researcher explicitly told all participants they were more interested in the methods used to create music than the resultant artefacts, preventing participants from feeling pressured into making a certain type or style of music.

[5] These two systems were added following feedback from the 1st round of pilot studies.



Note Type Editor
Use this to change note legnth, as well as apply rest & ties

Figure 8.3.: IGME's dynamic help system, which, once enabled a user can scroll over a component to get a description of its purpose.

8.4. Initial Pilot Studies

Two primary rounds of pilot studies were conducted with IGME, built on the concept of participatory design. These studies resulted in incremental changes to the software, and helped to evaluate and fine-tune the methodologies. Participants were given one hour to experiment with the IGME software. The researcher was available to answer questions and engage in informal discussions. IGME was supplied on a USB stick. This contained the software and was the vessel for collecting interaction data from participants. Informed written consent was obtained from each user before the memory stick was distributed. Each USB stick had a unique name, which was included on each consent form, so all data could be collated later. Each participant used an identical computer and operating system (iMac).

8.4.1. Study 1

The first round of these studies took place through March-May 2018 involving 9 participants. A preliminary study was undertaken with an additional user to catch any unforeseen issues. The participants were all first-year music technology students studying at the researcher's university. They spent an hour with the software, were given a set of printed instructions and tutorials (see appendix item F), and were asked to complete survey A at the end of the study.

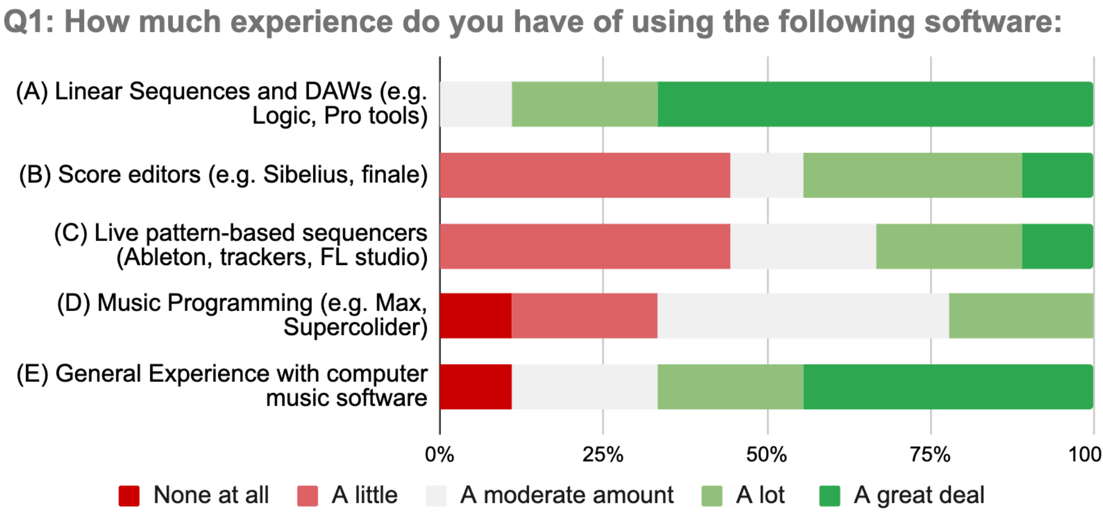


Figure 8.4.: Results for Q1 - music software experience.

The two initial questions polled users' experience of music software. Answers to question 1 (Figure 8.4), revealed participants were mostly experienced with linear music sequencers, but were less experienced with score editors and pattern-based sequencers [6]. Question 2 asked: *"what is your preferred composition/DAW package?"*, five participants answered *'Logic Pro X'*, with each of the rest answering, respectively, *'Sibelius', 'Reason', 'Live'* and *'Pro Tools'.*

Question 3 evaluated the participants' pre-existing exposure to computer-generated music techniques (Figure 8.5). Most participants had minimal exposure to the majority of these techniques. In line with other observations (Section 5.3 and appendix A), none of the participants had used neural networks, Markov models, or dice games, but around half had used algorithmic forms of music composition (3G, 3H). Notably, all but one had used an arpeggiator process. Overall the results suggest users remained mostly unexposed to generative and algorithmic music processes.

Question 4 asked: *"what did you find enjoyable about using IGME"*. Answers included *"using generative plug-ins....the range of effects and parameters make it versatile"* and *"working alongside the computer"*. One participant noted specifically *"I enjoyed how quickly ideas could be generated"*. Two users commented the interface was similar to other existing DAWs.

Question 5 asked, *"what did you find particularly frustrating[7] about using IGME?"*. Most criticisms were about a lack of features and familiar workflows, such as the ability to *"edit with mouse drags*

[6] One user recorded 'none' for their experience of general music software (E), but this seemed to have been an error as the same user also recorded 'lots of experience' for (A).

[7] Question 5 focused on frustration, which is rarely covered by HCI research (Brown, Nash, and Mitchell, 2017).

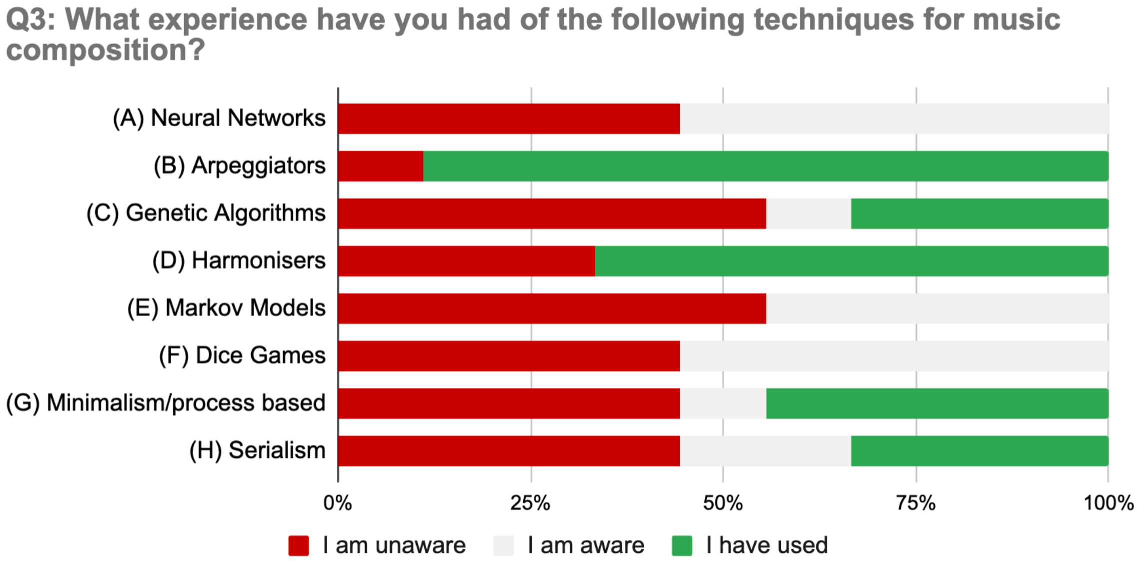


Figure 8.5.: Request for Q3 - using generative music techniques.

and use keyboard shortcuts". As an alpha version, the software frequently crashed or exhibited bugs. The biggest complaint was that, as part of the interaction sequence for generating music output, it was necessary to use the generative engine toggle (disabled by default) and the user had to *"render the piece to activate the generative effect"* manually. This also created hidden dependencies, as triggering playback caused iteration only when the button was toggled. This was redesigned in future versions (see section 7.5.2).

Question 6 asked, *"what part(s) of IGME could be improved (not including the synthesised audio quality*[8]*)"*. Most comments focused on improving keyboard shortcuts to match interaction sequences in similar software. A few users suggested that more effects could be added, specifically at the per-note level, and would also like to *"chain up multiple processes"*. The plug-in architecture was redesigned following the second round of pilot studies (discussed shortly). Finally, one user said *"the program's aesthetics could be improved"*. It is unclear how using software that appears unfinished (prototype) affects the overall user experience. Although IGME's interface and graphical design were refined throughout the project, the research made no explicit attempt to address or study this, which could be an exciting avenue for future research.

Promisingly, when asked, at the end of the survey, whether they would be willing to take part in similar studies in the future, all 9 participants said *'yes'*. The final question, inviting additional

[8] The audio at this point in development was notably primitive, with only two instruments inside the software. MIDI could also be sent out of IGME to an external application for synthesis.

comments, elicited no significant responses. Although this study was informal, it was paramount for testing and evaluating the research.

8.4.2. Study 2

The second round of studies took place in November 2018 involving ten participants. All were second-year music technology students (from the same university as the researcher) and none had previously used generative music techniques. The format was mostly the same as for the first study, except this study used survey B. The components that had been added to IGME (based on round 1) before this study are discussed at the end of this chapter. This round tested and finalised many of the research elements including the inbuilt tutorial system, complete logging of both music and interaction data, and the updated user survey.

A full discussion of the results from this sample of the questionnaire is not given, but the methodology is revisited in Chapter 9, in which participants had a feature-complete version of IGME and the sample size was larger. Rather, the results here are presented as a cursory analysis of the research at that point in time. Many questions received neutral answers, which could have been down to the wording of the question or a lack of time spent with the software, possibly suggesting participants had gained insufficient knowledge to form a view.

Results

Answers for questions 1-3 are summarised in Figure 8.6. The version control system (VCS) was mostly well-received, although question 1C *("I feel that the VCS allows me to check my progress")* had a mixed response and is analysed further in Chapter 9. It was hypothesised that the VCS would be increasingly useful in longitudinal studies (tested in Chapter 10). As expected, given its novel workflow, the two-stage editing process (question 2) elicited more varied answers.

The generative plug-ins (question 3) were useful for coming up with new ideas (3A). However, in general, users wanted more control over the plug-ins (3C) and to design their own processes (3D). The underlying computer-generated process architecture was redesigned following this study and is discussed at the end of this chapter. Some participants found the generative plug-ins made annoying mistakes (3B), but most were neutral about them, highlighting how unpredictable stochastic processes really are, with it being very easy to generate extreme music, but less easy to make '*human-esque*' music. The answers suggest that knowing how easy it is to generate a new

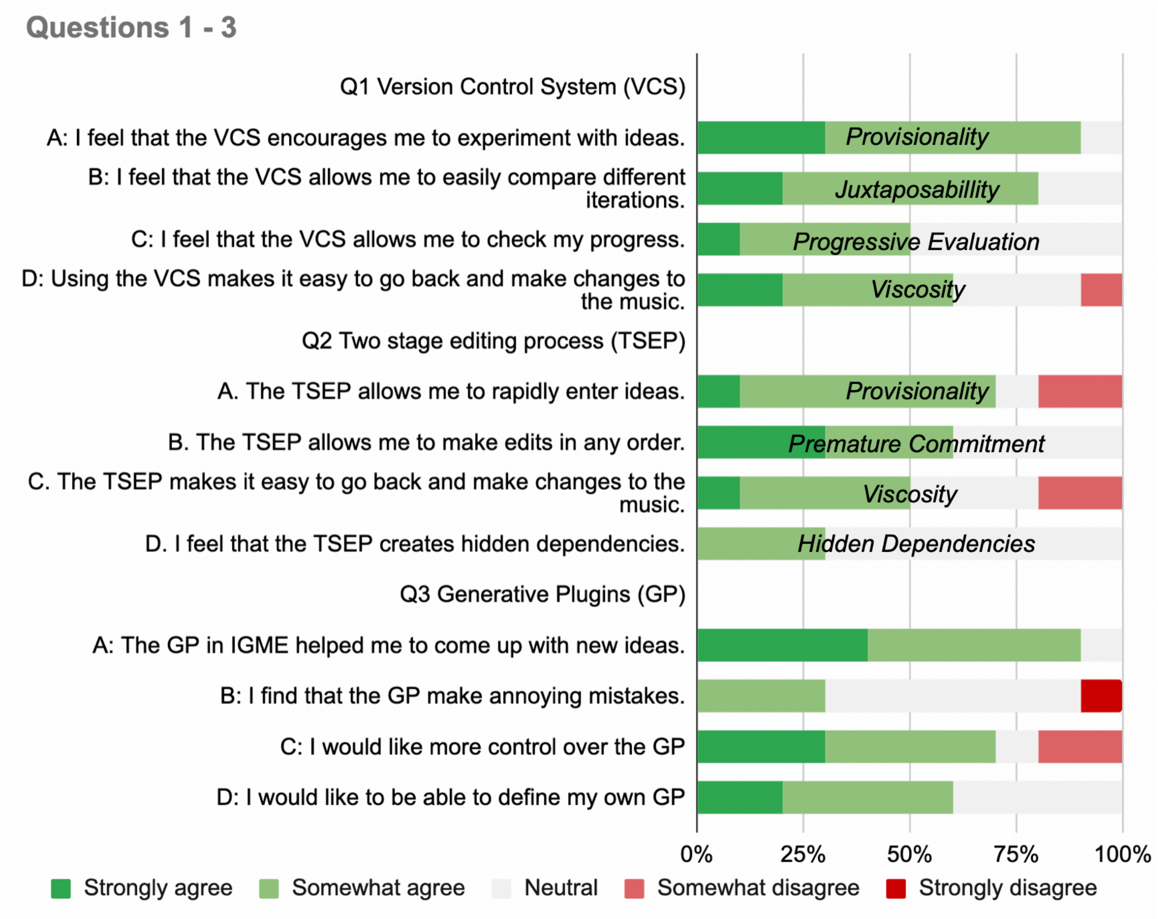


Figure 8.6.: Results for questions 1-3 from survey B.

variant and stop playback, and that a more pleasing result is possible, users expected or even tolerated mistakes.

Answers to questions 4-7 are summarised in Figure 8.7. Question 5 is excluded from the discussion as it was optional (for those who had used reference parts) and only 1 participant completed it. For question 4, most people agreed it was easy to distinguish the role of different parts (4A), whereas 4B and 4C received an almost equal split between *agree* and *neutral*. Question 6A revealed most users found IGME a challenge to work with. As discussed previously, maintaining challenge is an essential characteristic of *flow*, so such a result is not necessarily alarming. Participants mostly agreed they had enough control over IGME's generative processes (6B), but also said the computer had taken away control (6C). The majority of participants found IGME similar to other sequencers (6F). It is unclear if the tutorial system helped users learn to adapt (6G). Most participants agreed their knowledge of generative music (6D) had improved as a result of their involvement in the study.

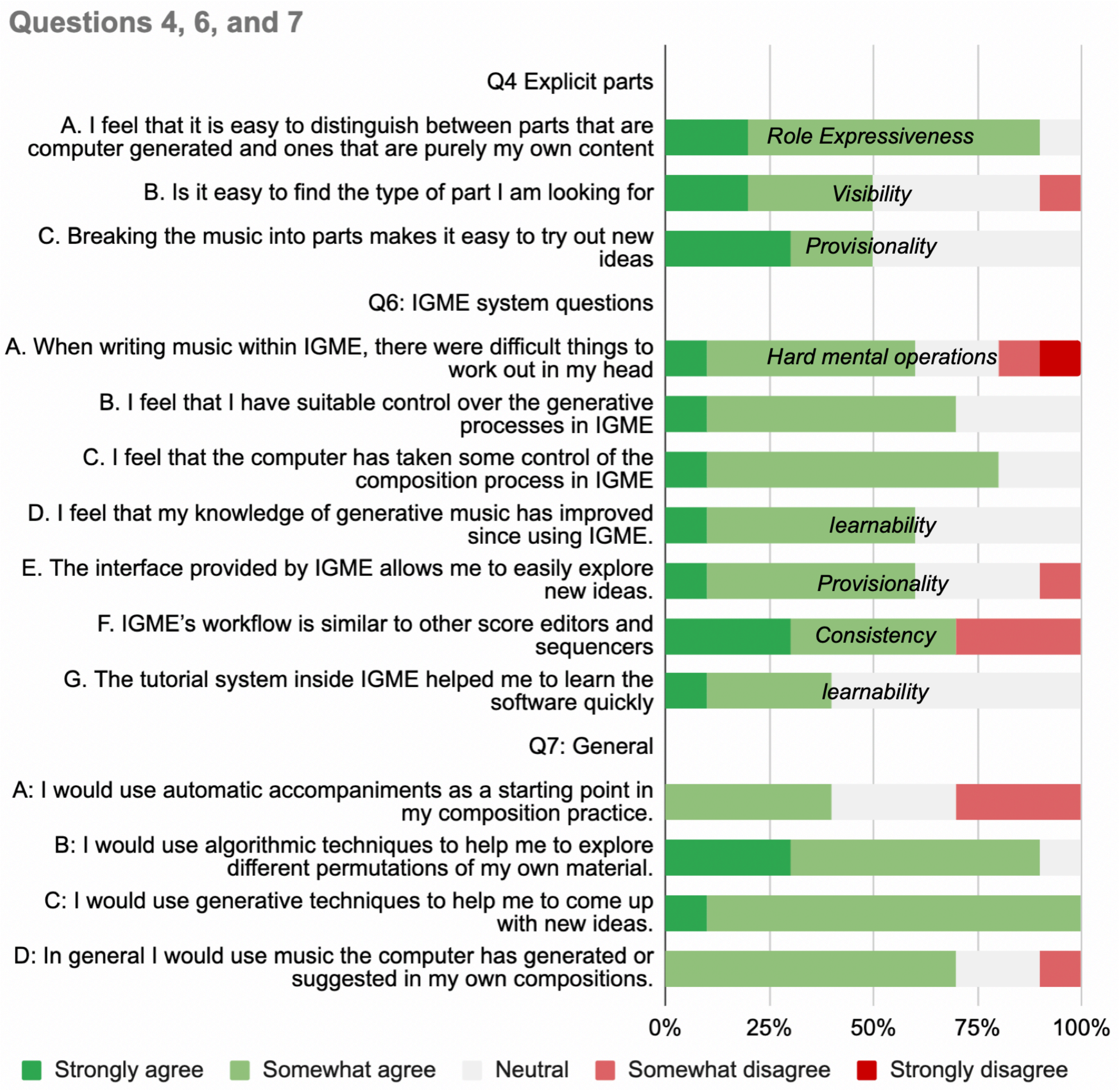


Figure 8.7.: Results for questions 4, 6, and 7 from survey B.

For question 7, participants indicated they would use generative music for coming up with new ideas (7C), and as a tool alongside their own practices (7B), but they were less open to the notion of automatic accompaniments (7A). On reflection, question 7A was confusing[9], so the ordering and wording of questions were modified in future versions of this survey. Question 7D's conclusion that 70% of participants would use computer-generated music in their own compositions is encouraging.

[9] As using accompaniments would assume the user already had a starting point (initial music), which conflicts with question's wording.

All	Some	Neutral	Somewhat me	All me
1	3	4	2	0

Table 8.2.: Results for question 8, *"Having used IGME, how much of the musical creativity do you attribute to the computer?"*

Question 8 (Table 8.2) polled *"having used IGME, how much of the musical creativity do you attribute to the computer?"* with the results leaning towards computer attribution[10]. To supplement this question 9 asked *"is the authorship of your creative output a concern when using generative techniques?"* with nine participants answering *'no'* and one *'yes'*. Therefore, even if composers are attributing the output to the computer, they are not concerned about it. Future versions of this survey included additional questions to address shortcomings in this area.

The last two questions were *"what are the positive/negative aspects of using generative music?"* with answers given as textual responses. Positive responses included *"this could be used as a time-saving instrument"* and *"for coming up with ideas people would not have had on their own"*. As one response states: *"it's free from the bounds of human imagination and creativity which can be limiting sometimes"*. Negative responses were critical of both IGME and wider issues already documented (Chapter 3) and included that users sometimes had insufficient control, or that some of the generated output did not sound pleasing or even make any musical sense. One response was: *"you could end up becoming too reliant on the computer"*, which as another noted *"human input is a fundamental part of music"*. As previously mentioned, the aim of this research was to work alongside humans, not replace them.

8.5. Amendments

The pilot studies, and subsequent iterations of IGME development, were crucial for catching issues and errors. Fortunately, only minor amendments were needed between and after the initial pilot studies.

[10] The answers here probably depended on the extent to which users had engaged with generative processes (discussed further in Chapter 9).

8.5.1. User Surveys

Following the second pilot study, survey B was converted from a paper handout to an online form, with the content amended slightly[11]. Participants were asked for their email address and IGME product key. Question block 3 added the sub-question, *"using generative music/algorithmic techniques helped me to come up with ideas that I would not otherwise have created on my own"*, and block 6 added *"which feature of IGME would you most like to see in your usual music sequencing application"*.

The ordering and wording for question block 7 changed. Based on responses in the pilot study, suggesting that generative music could be a productivity increasing tool, the question *"using generative/algorithmic techniques helps to increase my productivity"* was included. Blocks 8 and 9 were extended to include an extra open-ended text response box and an additional question asking *"what is the maximum percentage of automated creativity you would tolerate?"* with answers given on a continuous scale, from *'me only'* to *'fully the computer'*.

8.5.2. Updates to IGME

This section presents an overview of the major changes to IGME during development and user evaluation. All changes were the result of user feedback. This list is not exhaustive as there are many smaller insignificant changes, such as bug fixes, typos, and UI resizes.

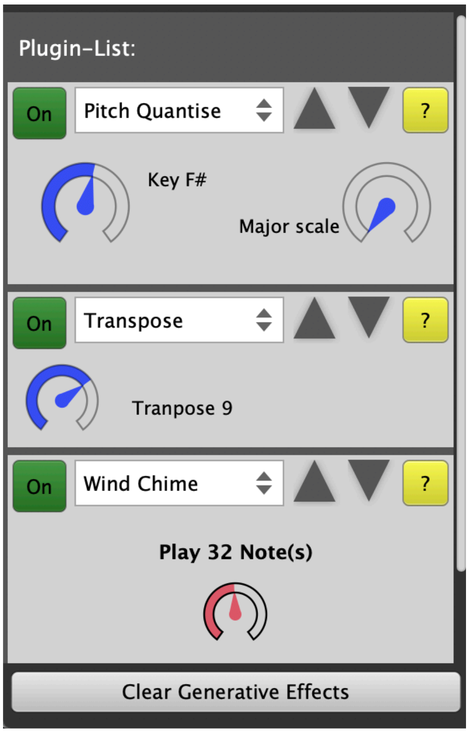
Iteration and Playback

At first, users were confused over the difference between playback and iteration. When iterations were triggered by initiating playback with the generative engine toggle button on (off by default), users were unclear on the exact workflow combinations that would result in playback or iteration. After the 1st pilot study, playback and iteration were more clearly defined and given separate buttons. Additional material was also added to the inbuilt tutorials. The need for users to understand the distinction between deterministic and stochastic music became an identified threshold concept in Chapter 10.

Plug-in Architecture Change

After the second round of the pilot study, the plug-in architecture was found to limit compositional options. Many users wanted to chain up more than one computer-generated process. This was

[11] The survey was also further amended for the real-world conditions study with this being discussed further in Chapter 9.


Plugin-List:

Pitch Quantise

Key F#

Major scale

Transpose

Tranpose 9

Wind Chime

Play 32 Note(s)

Figure 8.8.: IGME's plug-in list, showing several effects chained up.

achieved both by developing another process, referred to as the note properties editor (discussed below), and through redesigning the plug-in component.

The note property editor (Figure 8.9) permits the user to select notes in the editor and apply one of three stochastic processes; chance (the chance of the note happening), pitch range (the range in which the pitch of the note will be randomised), and duration (the length at which the note will be randomly increased or shortened). A visualiser was added to show the effect this would have on each note (Figure 8.10). The note parameter allows per-note generative effects, whereas the plug-ins process the entire sequence.

Additional Part Types

Reference parts are a more complicated feature of IGME. They were used in the pilot studies only by the most engaged participants, who found them challenging. Their full potential (discussed in Chapter 10) was unappreciated until users had more experience of IGME, at which point they were seen as crucial. Based on user feedback, the iterator part and repeater part were added (discussed in Chapter 7).

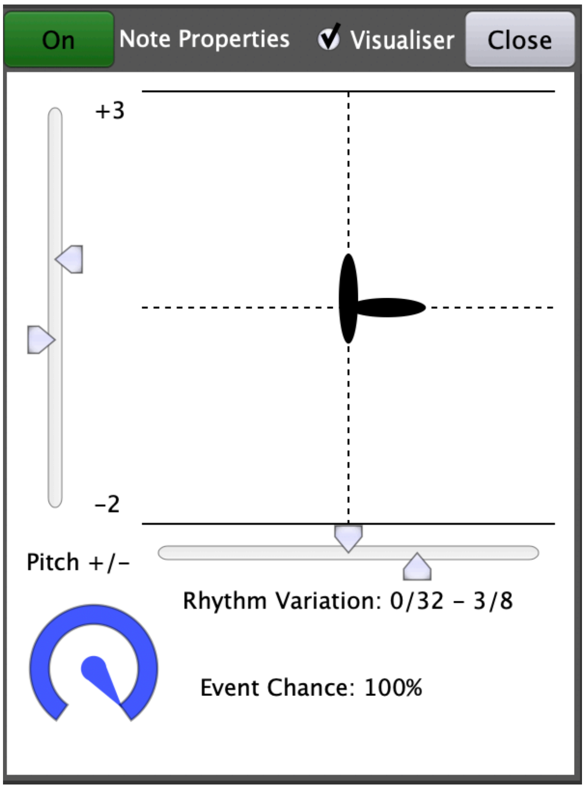


Figure 8.9.: IGME's note properties editor.

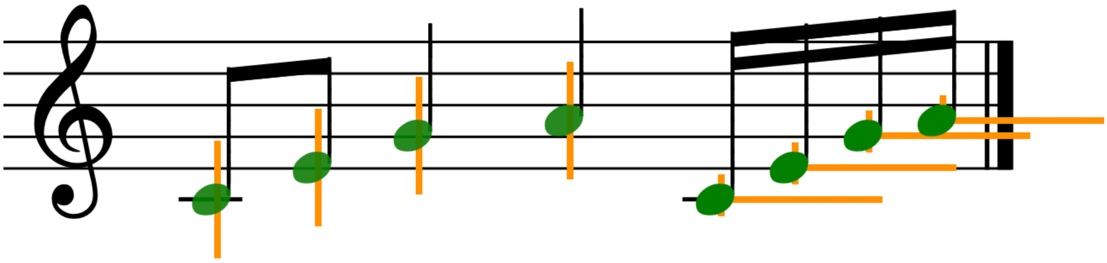


Figure 8.10.: Note properties visualised in IGME's note editor. Up-down represents the pitch variation, left-right the note length variation, and opacity the note's chance. Green coloured notes have these properties applied to them, whereas black notes (default) do not.

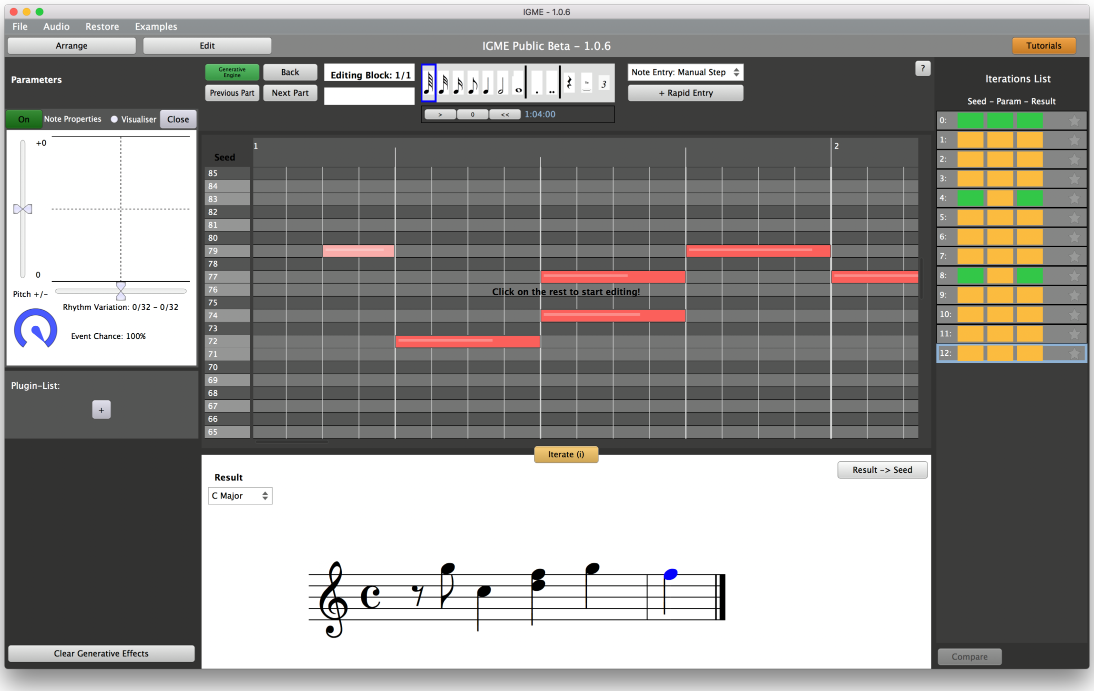


Figure 8.11.: IGME's piano roll editor used for editing the seed, and the result viewer (output) shown in score notation.

Piano Roll Editor

Originally IGME intended to exclusively use score editor notation, however, some users had too little notional literacy to use IGME effectively. Instead, many were adept at using piano roll notation, so a piano roll editor (Figure 8.11) was added after the 1-hour workshops (November 2019), before the longitudinal studies. Users were free to switch between score or piano roll in one of three places: the arrange view inspector, the editor, and the result viewer.

8.5.3. Methodology Amendments

IGME was originally developed to be run in controlled conditions in user-facing workshops. Giving IGME to composers in their own environments has many benefits, notably mitigating the Hawthorne effect (McCarney *et al.*, 2007), in which participants modify their actions, on the perception of being observed. But this presents many technical challenges, including sending all data remotely, obtaining consent and ring-fencing product activation. The biggest hurdle was porting IGME to Windows, warranted by its substantial market share (Sethi, 2018). With little or no marketing, the researcher struggled to draw attention to the project; as a result, few participants signed up, and of those many only sent limited interaction data. Even with regular conference attendance and a public outreach talk (Watershed, 2019), attracting participants remained a challenge throughout. The public beta approach was eventually abandoned, but the developed technology permitted the remote study (discussed in Chapter 9) to take place.

Due to the COVID-19 outbreak of 2020 the longitudinal video study (Chapter 10) had to be adjusted, resulting in a small sample size (4 total). The researcher deemed this sufficient for analysis, due to the long periods of time each participant spent with the software. The analysis, in conjunction with other studies, allowed for conclusions to be drawn. An additional study was conducted to provide a different set of conditions that had not yet been explored. This presented the opportunity to run the formal 1-hour workshop of Chapter 9, repeated remotely (discussed further in Chapter 9). Time and resources prevented the researcher from running additional studies.

8.6. Conclusion

This chapter details the methods for evaluating IGME and is the foundation for the following chapters. Chapter 9 discusses the results of 54 participants each using IGME for 1 hour, 23 in controlled conditions, and 31 in real-world conditions. Chapter 10 discusses longitudinal studies of IGME, involving screen-recording technologies. The overall findings from the user studies are collated, discussed, and concluded in Chapter 11.

9. Studies Using Controlled and Real-World Conditions

This chapter discusses two studies in which participants explored computer-generated music composition using IGME in a university computer lab (controlled conditions) and a similar study conducted remotely as a conference workshop tutorial (real-world uncontrolled conditions). The participants of these studies are referred to as group 1 (23 participants) and group 2 (31 participants) respectively.

Quantitative interaction data and qualitative survey data summarising computer-generated music (CGM) experiences were used to evaluate IGME and to make recommendations for future systems and interfaces, with the findings extended through the work of the next chapter. Work from this chapter was originally presented at NIME[1] 2020, in Hunt, Mitchell, and Nash (2020).

In total, 54 participants used IGME across two different studies, with the overall aim of understanding:

- *"How do the interfaces, tools and workflows engender interactive CGM composition?"*
- *"What role can CGM take within composition?"*

9.1. Background

As stated in previous chapters, computational tools for automatically composing music have been thoroughly explored in literature. However, little research has been undertaken on the use of such techniques alongside human composition within traditional music sequencing software (Chapter 4). Instead, existing computer-generated systems use workflows that are unfamiliar to non-programmer composers (Bellingham, Holland, and Mulholland, 2014a).

[1] Conference on New Interfaces for Musical Expression.

	Session Length	Sample Size	Complete Survey Results	Renumeration	Survey	Date
Controlled	1 hour	23	23	£10	B-2	11/2019
Real-world	2 hours	31	9	no	B-3	10/2020

Table 9.1.: Comparison and summary of the two studies.

Limited empirical studies have been conducted on how users interact with music composition software (probably due to the inherently invasive nature of doing so). Even less research exists on users of CGM systems. When such studies were conducted, heavy emphasis was placed on evaluating the quality of the computer-generated output, but not necessarily on the human factors relating to the use of such systems (discussed in Chapters 4). Moreover, evaluation remains occasional rather than frequent (Ariza, 2009). In summarising user experience studies from 132 papers from the NIME, SMC and ICMC[2] conferences Brown, Nash, and Mitchell (2017) found the composers' perspectives were rarely evaluated.

Nevels (2013) studied a student composing a song with off-the-shelf software, and Collins (2005) conducted a three-year case study of a single composer. Both studies were limited by their small sample size of one, whereas Nash (2011) completed a larger observational study of tracking software, summarising 1000s of hours of interaction data. Duignan (2010) studied 17 music producers and their use of abstraction mechanisms in common music sequencing software. However, none of these studies focused on CGM.

9.2. Methods

Two similar, but related, methods were employed for studying interaction. These are discussed further below but summarised in Table 9.1. Both groups were given survey B (previously discussed in Chapter 8) at the end of the session. Group 1 were given B-2, and group 2, B-3 which was amended by adding three new questions (discussed in section 9.2.2). Additionally, interaction and music data were collected autonomously for all participants. Each participant in group 1 was remunerated £10 in Amazon vouchers, whereas group 2 were not remunerated.

[2] New Interfaces for Musical Expression, Sound and Music Computing, and International Computer Music conferences.

9.2.1. Controlled Conditions

For the controlled conditions study, 23 first-year undergraduate music technology students *(aged between 18-25, F = 3, M = 20)*, enrolled at the same university as the researcher, were invited to spend an hour experimenting with IGME in a university computer lab. 7 of the 23 were creative music technology students and the others, audio music technology students. The former specialises in composition, and the latter on a more technical side of audio (for example digital signal processing).

Two separate sessions, conducted on the 12th November 2019 were run inside a Mac computer lab at the university with 10 and 13 participants in each. Participants were asked to work through a series of tutorials[3], these can be loosely summarised as follows:

1. Getting started (editing notes, adding parts)
2. Editing Notes (human parts)
3. Adding computer-generated processes (human-computer part)
4. Computer-generated music (computer parts)

Any remaining time could be spent freely. The researcher was present and answered questions if needed, but otherwise remained a passive observer. After the session, all participants completed survey B-2.

9.2.2. Real-World Uncontrolled Conditions

The real-world study was conducted on October 19th at the 2020 Joint Conference on AI Music Creativity (Sturm, 2020). The format was adapted to work under the Covid-19 lockdown conditions and therefore run remotely with 31 participants. Two separate, two-hour[4] sessions took place, with 21 and 10 participants respectively.

The tutorial used the Blackboard Collaborate Ultra (BBCU) e-learning tool (Hill, 2019), allowing the researcher to demonstrate the software and PowerPoint slides. A chat window enabled participants to ask questions and talk among themselves. Despite the intention to follow a plan, many participants left after an hour, and those that remained were happy enough not to warrant further demonstrations (step 5 below). The proposed format of the sessions were:

[3] The tutorial material can be viewed at (Hunt, 2021) or in Appendix Item F.

[4] Although timetabled for two hours users did on average attend for less than half this time, see Figure 9.7.

1. Introduction and research background (10 minutes)
2. Software demonstration 1 (5 minutes)
3. Setup (5 minutes)
4. Working through tutorials (40 minutes)
5. Demonstration 2 (20 minutes)
6. Free time (30 minutes)
7. Complete survey (optional)

At the end of the session, participants were emailed and encouraged (but not required) to complete questionnaire B-3. Of the 31 participants, 9 completed the survey in full and 4 abandoned it, while 18 did not respond to the email request. Interaction data was automatically collected for all 31 participants.

Unlike previous studies involving IGME, group 2 participants had broader backgrounds than the music technology students who, up to that point, had been the only users. All group 2

Participant self described background
The following participants submitted a complete survey response:
• Classical guitarist. Wide interests but particularly Baroque and Jazz.
• I've been taking piano lessons for the past 7 years or so.
• Composer, musician, coder. Classically trained on violin, self-taught on piano and theremin.
• 4 years in classical performance, 1 year in electronic composition. 1 year in creative computer science.
• Music enthusiast and self-taught musician.
• Music producer, Executive producer, Bandleader, Tour manager, Composer, MusicAI researcher.
• Newbie, just played around with GarageBand, MAX and web app to create music but have no music theory knowledge.
• Somewhere between screaming into the void and getting a PhD in composition.
• Electronic music producer/engineer for 20 years. Experience with Digital Performer in the 90s and 2000s and Ableton Live for the past 10. Have dabbled with Max and Reaktor.
The following participants submitted only a partial survey response:
• BA in Music performance-voice, classical. Concerts and opera projects.
• Sound engineer, audio plugin developer, multi-instrumentalist musician.
• I am a musical enthusiast and I've learned most of music theory by myself (youtube videos, playing with friends). I have no formal musical background. I can play the guitar and the bass and have a computer science degree. I am interested in generative models and I have tried some of the generative tools available today, such as Google Magenta's plug-ins and custom models available on github.
• Electronic music, performance, live coder, write own tools in Max/MSP, CSound.

Table 9.2.: Responses when asked: "briefly in your own words describe your musical background".

participants were registered conference attendees and had signed up via a link on the conference homepage a week earlier. The unedited descriptions of their backgrounds are listed in Table 9.2. Given the conference subject, and subsequent attendance, group 2 participants were inherently more interested in CGM than those in group 1. 61% of users in this study had used CGM before compared with 39% among group 1.

9.2.3. Survey Amendments

An additional question block (using a 5-point Likert scale) was inserted and asked: *"I found learning IGME using the provided materials (tutorials, presentation and video)"*, and *"I feel that with time I could master IGME"*. Compared to group 1, group 2 participants had less immediate help from the researcher, who was also less able to gauge the *'mood of the room'*, requiring learnability to be captured more explicitly[5]. Additionally, participants were asked *"briefly in your own words describe your musical background"* as previous versions of this study collected minimal information about the participant's background.

9.3. User Survey Results and Discussion

The number of complete survey responses for group 1 (23) was considerably larger than that of group 2 (9), so comparisons are difficult to make concretely. Although the sample size of group 2 is smaller, it contains self-selecting intrinsically motivated individuals.

To test if the distributions in results between both groups differed significantly, a Mann-Whitney U test was employed (Bertram, 2007), as the data is ordinal, non-parametric, and contains independent samples. This was computed between questions 1-7 (excluding 5[6]), resulting in 28 separate tests. With an α value of 0.05, we find for 24/28 tests to fail to reject the $H0$ hypothesis (the distributions are the same) and for questions (2A, 2B, 4B, and 7D) reject $H0$ (the distributions are different), with these being discussed further in the relevant sub-sections below. Table 9.3 shows the resultant values for each test.

In summary, the outcomes of the group 1 and group 2 studies are similar enough, such that observations, suggestions, and findings are comparable. Table 9.6 and 9.7 additionally shows the mode and median for both group's answers, which also shows similarity in the responses received.

[5] Learnability is a theme of the video study discussed at length in the next chapter.

[6] Was optional and collected few responses.

Question		U statistic	p	Fail to reject $H0$
1	A	105.5	0.465	True
	B	103.5	0.431	True
	C	89.5	0.224	True
	D	97.5	0.334	True
2	A	50	0.005	**False**
	B	55	0.012	**False**
	C	74	0.08	True
	D	74	0.058	True
3	A	90.5	0.233	True
	B	107	0.492	True
	C	102.5	0.416	True
	D	80.5	0.123	True
	E	102.5	0.416	True
4	A	103	0.424	True
	B	67	0.043	**False**
	C	105.5	0.465	True
6	A	69	0.05	True
	B	103.5	0.427	True
	C	81	0.124	True
	D	98	0.341	True
	E	71	0.054	True
	F	89.5	0.218	True
	G	86	0.175	True
7	A	94	0.271	True
	B	96.5	0.315	True
	C	78	0.1	True
	D	63	0.031	**False**
	E	78	0.107	True

Table 9.3.: Mann-Whitney U test results, $\alpha = 0.05, n_1 = 23, n_2 = 9$.

The remainder of this section briefly discusses the results for both groups for each question using a series of figures, with each question having two sets of results, one for each group.

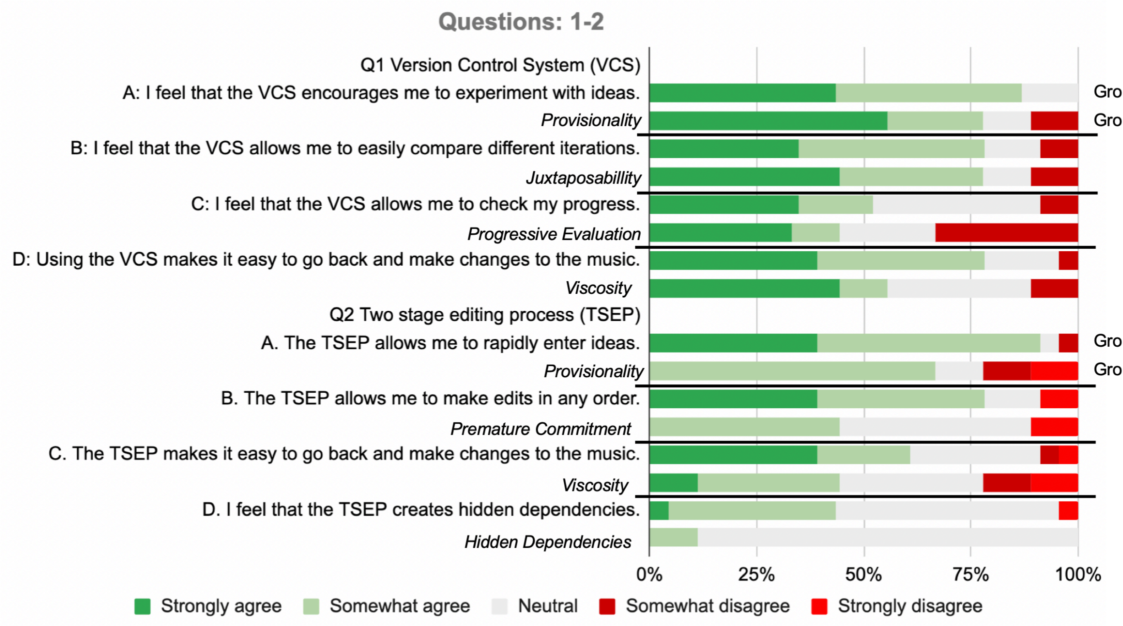


Figure 9.1.: Questions 1 (version control) and 2 (two-stage editing process) results.

9.3.1. Version Control System

The version control system (VCS) results (Figure 9.1) show most participants agreed the VCS supported *provisionality* (1A) and *juxtaposabillity* (1B). Many participants did not find the VCS useful for checking progress (*progressive evaluation*, 1C). With an open-ended task such as music composition, it is not easy to define *'progress'*. The VCS enables a user to evaluate progress by loading previous versions and comparing them. However, the survey results suggest this does not happen often in practice[7]. While most participants thought the VCS made it easy to make changes, therefore having a low level of *viscosity* (1D), there was some slight variance between both groups.

9.3.2. Two-Stage Editing

The two-stage editing model is designed to increase *provisionality* (2A), while reducing *premature commitment, viscosity, and hidden dependencies* (2B, 2C, and 2D respectively). The results (Figure 9.1) show most participants agreed that *provisionality* increased (2A), *premature commitment* decreased (2B) and *viscosity* decreased marginally (2C). The last question (2D) in this section was on *hidden dependencies*. Although 43% of participants found hidden dependencies had increased in group 1, no meaningful conclusions could be drawn, given the large proportion of neutral answers in both

[7] This is discussed in detail in Chapter 10

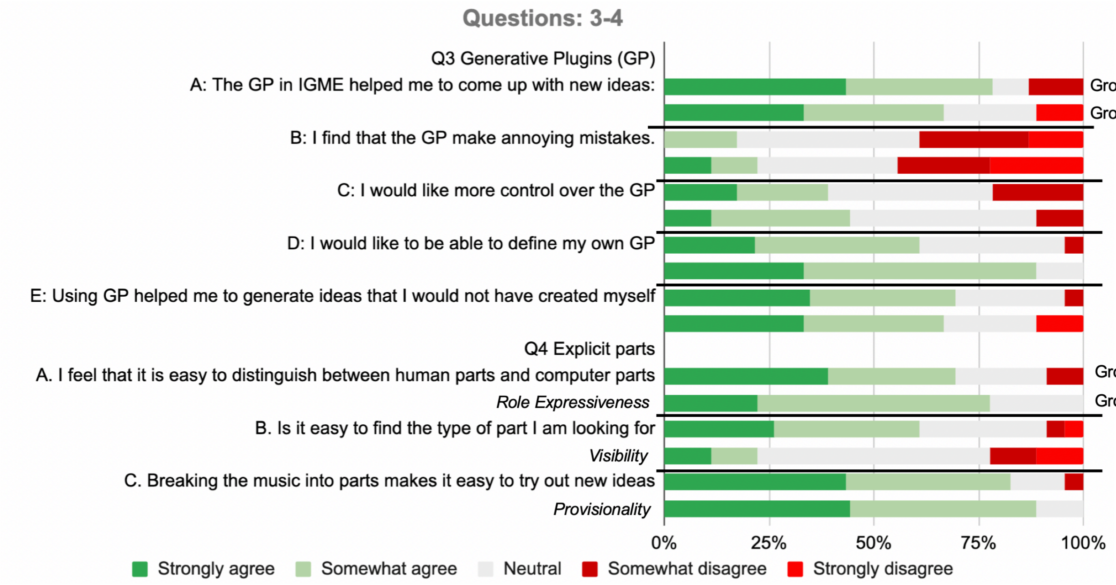


Figure 9.2.: Questions 3 (generative plug-ins) and 4 (explicit parts) results.

groups. Although $H0$ is rejected for both 2A and 2B, it remains unclear on the reason for different responses between groups.

9.3.3. Generative Plug-ins

The questions in this block were not explicitly aligned with any of the cognitive dimensions but were intended to evaluate IGME's computer-generated effects. The results (Figure 9.2) show the majority of participants agreed that the computer-generated plug-ins helped them *"to come up with ideas"* (3A). Similarly when asked *"if the program generated ideas they would not have come up with on their own"* (3E) most agreed this was the case. The majority of participants said they would like to be able to define their own generative effects (3D), but they remained generally neutral when asked if they would like more control over the generative processes (3C). Slightly more *'agree'* answers were given among group 2 for 3D, although the median and mode were the same for both groups. Group 2 contained more experienced music practitioners, who perhaps wished to have more control. When asked if the software made annoying mistakes (3B), answers were mixed. It is difficult to draw meaningful conclusions when assessing musical quality, given its subjective nature. For example, a musical pattern may sound terrible to one person while being wholly acceptable to another.

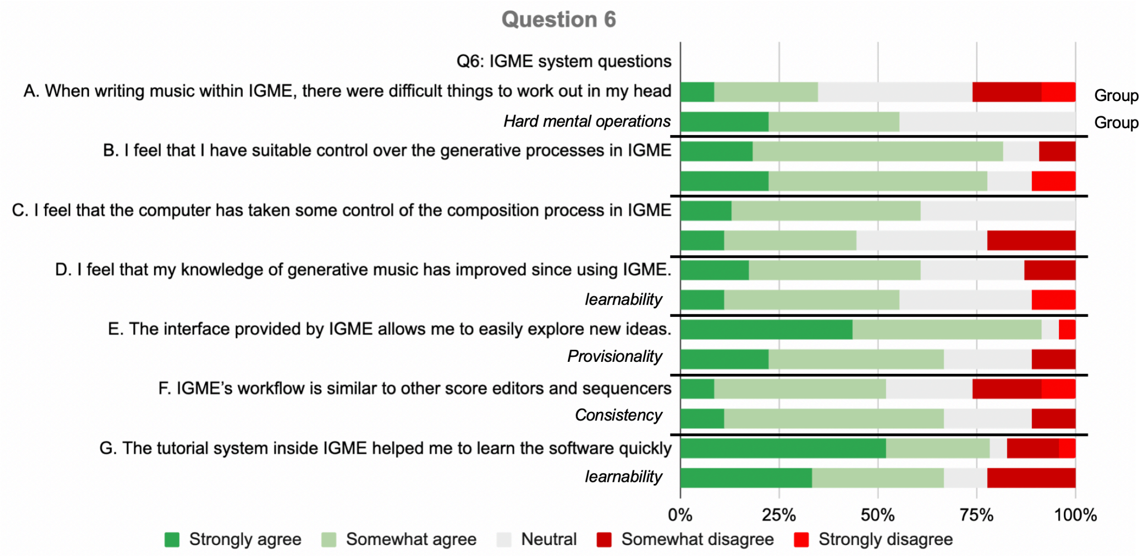


Figure 9.3.: Question 6 (IGME system) results.

9.3.4. Explicit Parts

Regarding the 4th question (Figure 9.2), most participants agreed that it was easy to distinguish between parts (4A). Group 1 agreed it was easy to find the part they were looking for when editing (4B), whereas group 2 were more neutral. The median differed and $H0$ was rejected for question 4B. The majority of participants agreed that breaking the music into parts made it easy to try out new ideas (4C).

9.3.5. IGME System Questions

This block of questions was about IGME in general. One shortcoming of the cognitive dimensions is that they fail to capture *learnability*. Work by Nash and Blackwell (2012) addresses this by defining a *virtuosity* dimension. Given the limited time spent with IGME, it is unlikely the users would have become virtuosic with it. However, questions 6D and 6G can be aligned with our definition of learnability, which is *"how easy is it to learn the interface"*. Questions 6B and 6C did not map strictly to any dimension.

Answers to question 6A were balanced for group 1 (Figure 9.3), suggesting that any *hard mental operations* are relatively forgiving, whereas group 2 found these slightly more taxing. The majority of respondents to question 6B felt they had suitable control over the generative processes. Concerning 6C, the majority of group 1 agreed that some control had been handed over to the computer, while a few participants in group 2 disagreed with this.

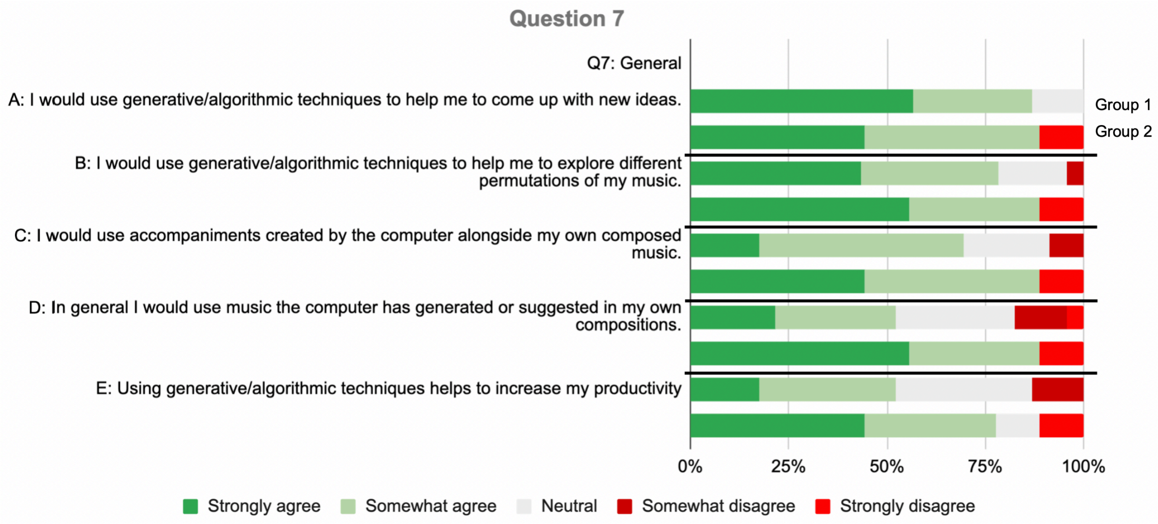


Figure 9.4.: Question 7 (using generative music) results

IGME was designed to be similar to other music software, borrowing design metaphors and workflows to aid *learnability*, although its unique features make it sufficiently different from other software. In terms of *learnability*, most participants agreed their knowledge of generative music had improved (6D). Asked whether IGME's inbuilt tutorial system (6G) had helped them learn the software, the majority agreed. Responses to 6F were mostly positive.

Finally, question 6E evaluated the ability of IGME users to explore new ideas. Almost all respondents in group 1 (91%) agreed that IGME did help them to explore new ideas, and group 2 gave similar results.

9.3.6. Using Generative Music

The questions in this section looked at in what situations CGM might be used and were not related to the cognitive dimensions (Figure 9.4). The majority of participants would use computer-generated techniques to come up with ideas (7A), and also to use such techniques for exploring different permutations of their own music (7B). Participants were equally happy to use automatically generated accompaniments (7C). Responses to question 7D were more polarised among group 1, whereas the majority of group 2 were in agreement, $h0$ was also rejected, which can probably be explained by group 2 being inherently more interested in the subject (computer-music).

As reflected in the number of *'strongly agree'* responses to questions 7A and 7B, and the slightly larger number of *'neutral'* or *'disagree'* responses to 7D, it could be argued that computer-generated music is best used as a catalyst for composition and for influencing and motivating individuals rather than for replacing them. Participants said they would use computer-generated music to create and alter ideas, but would not necessarily use it directly. Both of these crucial findings are discussed at length in Chapter 11.

When asked if CGM can be used to increase productivity (7E), results were balanced for both groups. Although CGM can be used to replace composition (potentially saving time), it can take many iterations to create something useful, thereby hindering productivity.

One stand-out observation is that few people from group 2 gave a neutral answer to the questions in this block. Given that more participants of group 2 had used CGM than group 1 they might already have a strong opinion of its merits or deficiencies[8].

9.3.7. Creativity

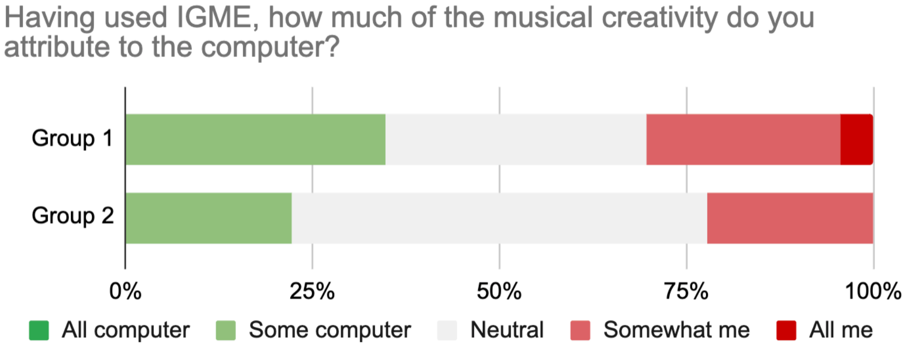


Figure 9.5.: Responses for both groups when asked: having used IGME, how much of the musical creativity do you attribute to the computer?

Balanced responses were given to the question *"having used IGME, how much of the musical creativity do you attribute to the computer?"* (Figure 9.5). Notably, no-one responded *'all'* for the computer, suggesting that, with CGM, a degree of authorship remains with the user.

When asked *"is the authorship of your creative output a concern when using generative techniques?"* 43% in group 1, and 33% in group 2 answered *'yes'*. Given the option to comment further, one participant stated, *"once the system begins to be an AI, issues of authorship arise, but for adding what is essentially a curated random chance, authorship and credit belong to the user."* Another gave this additional response:

[8] As noted previously CGM techniques are not for everyone.

> *"When using AI the composer sometimes becomes a "selector" in the Jamaican DJ sense of that word. I'm perfectly comfortable with this role. These types of algorithms don't give me authorship concerns per se; however if I imported a Beatles MIDI file culled from the internet, and non-creatively asked the algorithms to create more music like the imported MIDI, there's a nonzero chance I could run afoul of copyright law. Ultimately like for every tool creativity resides largely with the creator. It's tough for me to answer the last 0%-100% question. I'd tolerate 100% if I could set an AI music algorithm on autopilot, let it automatically upload music to streaming services, profit handsomely from the activity and spend all my newfound free time studying composition, producing, and playing jazz piano."*

Participants used a sliding 0-100 scale when responding to *"what is the maximum percentage of automated creativity you would tolerate?"* (see Figure 9.6). Participants in group 2 had a higher *tolerance* for accepting the computer's input, likely due to their background. In summary, users appeared to accept some of the computer's creative input[9].

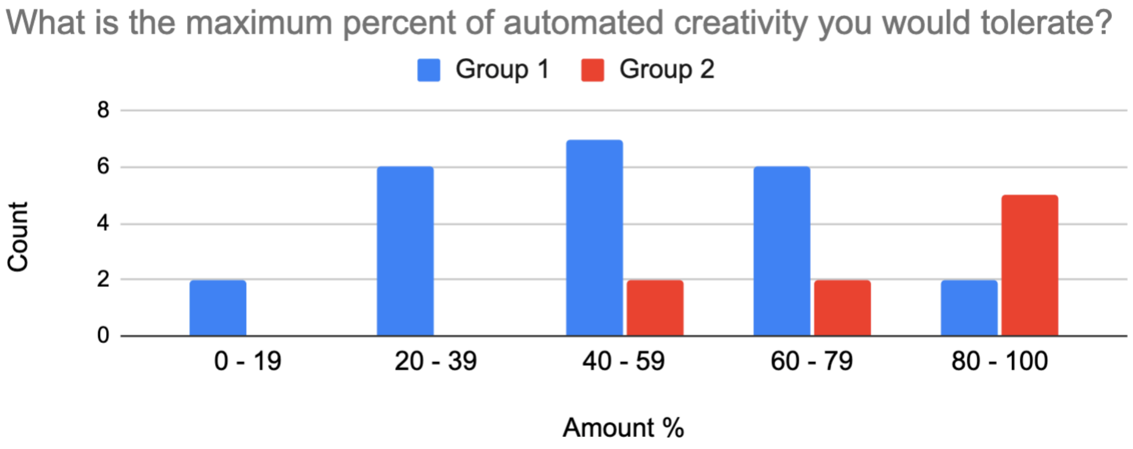


Figure 9.6.: Results for the above question.

[9] This finding reoccurs throughout the user studies, and is discussed at length in the conclusion (Chapter 11)

9.3.8. Open Questions

The final two questions were *"what are the positive and negative aspects of using generative music"*, with qualitative responses given. A list of all responses for these questions is given in appendix item E. For positive responses in group 1, one stated: *"It can create ideas that you would never otherwise come up with"*, with two others echoing similar thoughts. One individual said computer-generated music could *"help people stuck in a creative rut"*, while another said *"you don't have to type different sequences in over and over to try things out"*. One participant from group 2 stated: *"It is faster and more efficient, and can easily be rearranged according to the users' needs... and also allows people to compose their own music without having to learn musical theory"*, while another said: *"That I can understand my own creative process better, and develop new ideas using strictly the values I've predetermined"*.

For negative responses in group 1, two participants said the music created would not be considered their own and they would have lost some control over the composition process, although one stated *"you aren't forced to use exactly what's generated"*. It was observed that it could be time-consuming to create something that sounded *'decent'* and that *'chance'* played a key role, with another stating specifically, *"it can take very long to get something you like by chance"*. Another wrote that: *"there can be a lack in originality/identity in generative music over music that a person has made themselves"*. One participant said that it is *"almost too easy to make something, takes away part of the challenge?'*. Music composition can be considered a problem-solving (or constraint satisfaction) activity (Alty, 1995), and in computer-generated music, the problem-solving activity is perhaps the *challenge* of configuring parameters to produce a good result, rather than simply arranging a sequence of notes.

One participant from group 2 responded, *"the system could be a threat and could be detrimental to composer employment"*. Another stated: 'AI' is sometimes seen as a threat by *"people who know nothing about it"*. Another response that echoed the project's objectives was: *"generative music systems should always be thought of and developed as tools that enhance and support human creativity rather than replacing it."* One user believed more constraints were needed (a recurring theme in Chapter 10) as the system was too random, requiring *"more constraints based on music cognitive science"*.

I found learning IGME using the provided materials:

Very easy	Somewhat easy	Neutral	Somewhat challenging	Very challenging
2	3	0	4	0

Table 9.4.: Results for the above question.

I feel that with time I could master IGME:

Strongly agree	Somewhat agree	Neutral	Somewhat disagree	Strongly disagree
6	3	0	0	0

Table 9.5.: Results for the above question, $n = 9$.

9.3.9. Specific Questions for Group 2

The results in Tables 9.4 and 9.5 show the two additional questions added exclusively to the survey presented to group 2. The responses suggest users mostly found IGME easy to learn, but that it came with challenges. As already observed, a degree of challenge is needed for problem-solving/seeking exercises, and to induce a state of flow (Nash, 2011). All users agreed they could master IGME, but it is unclear how long this would take.

9.4. Interaction Data

This section briefly looks at the interaction data collected from both user groups. Interaction and music data are further discussed in Chapter 10 when users had additional time to use IGME. The sample size of group 2 was reduced to 24 (from 31) after inspecting the data, with the reasons for this discussed below.

9.4.1. Time

The time each user spent with IGME is shown in Figure 9.7. Group 1 spent on average 44 minutes with IGME, with the remaining time spent registering the software, and completing the questionnaire. Group 2 spent on average less time, however, the results show several outliers. Excluded from further analysis were 5 participants who stopped using IGME within 3 minutes of starting, and one who despite having IGME open for 2 hours recorded no activity until the last 5 minutes. Another participant in group 2 made no note edits, but recorded almost 1000 interaction points for configuring note parameters, switched notation 40 times, and changed instrument sound 90 times (over the 2 hours). That individual did not trigger any iterations, or produce any

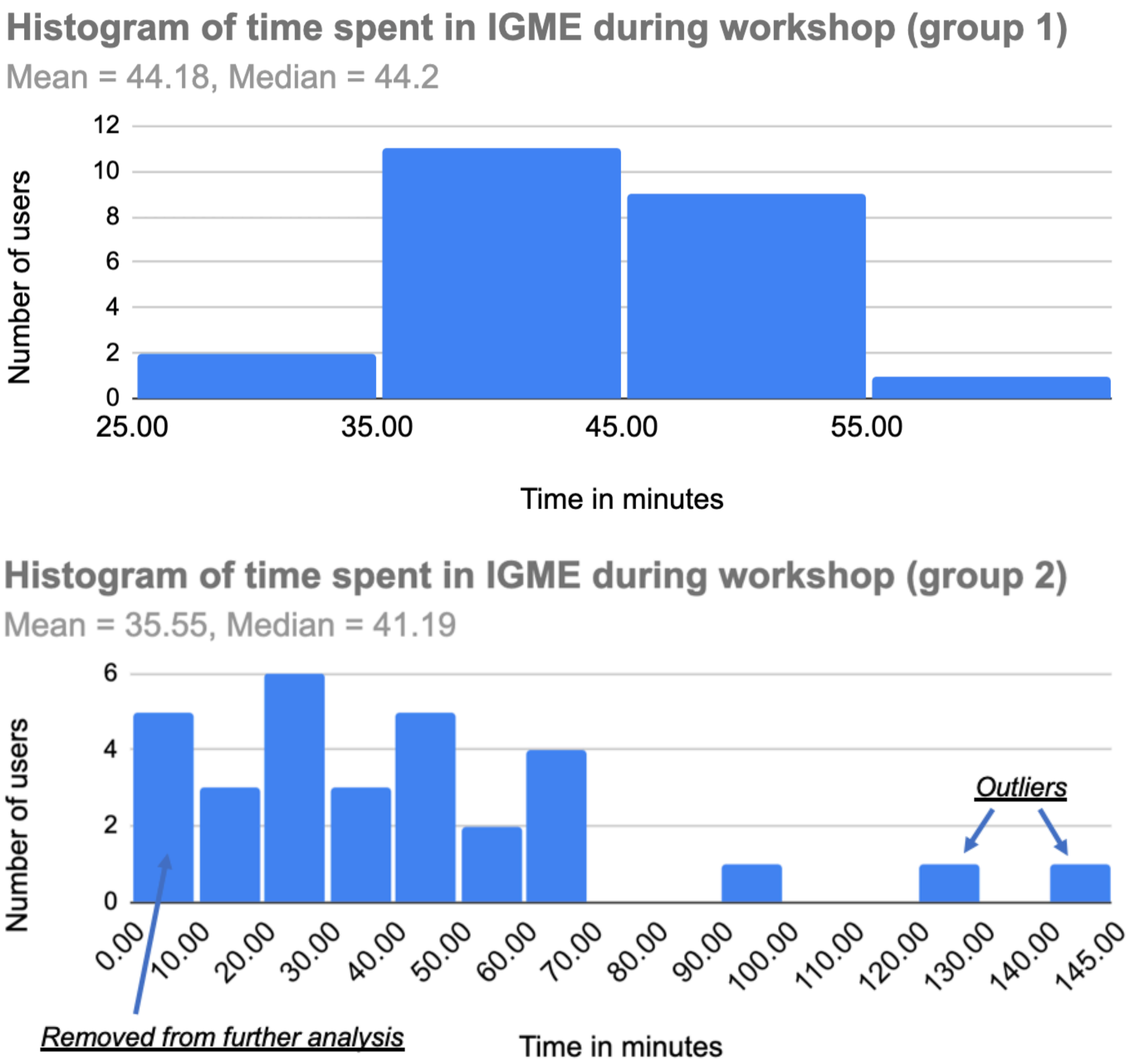


Figure 9.7.: Amount of time users spent with IGME (group 1 and 2)

audio playback, suggesting severe problems with learning the software and again removed from further analysis.

None of these 7 participants (outliers) provided a survey response. As noted by Gerken *et al.* (2008), users' behaviour cannot always be explained by interaction data, and in this case, left more questions than answers. In addition, such anomalies were not found in group 1, likely as a result of the controlled conditions.

One difference between each group was the duration spent between editing and arranging (Figure 9.8). Group 1 spent almost twice as much time in the editor than in the arrange view, whereas in group 2 the time spent was more varied.

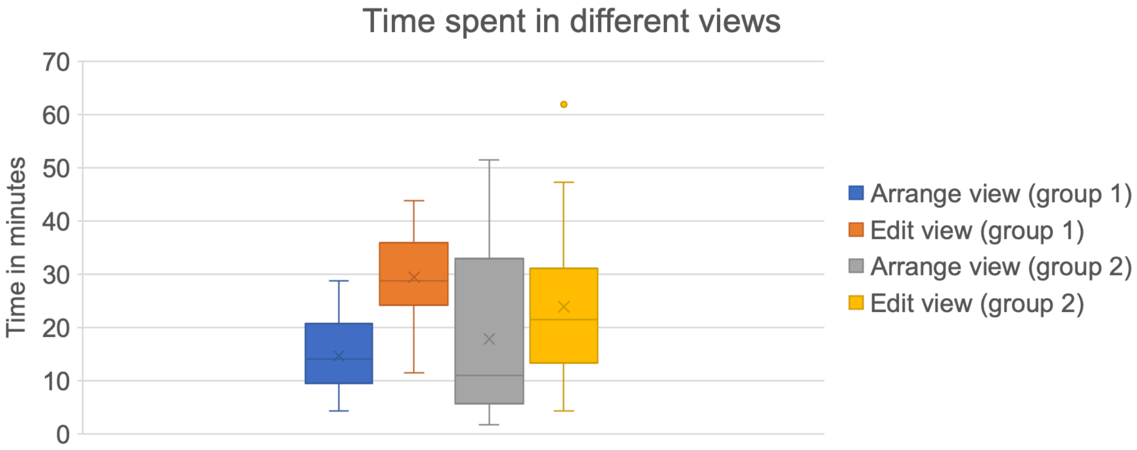


Figure 9.8.: Time spent in edit vs arrange views in IGME for both groups.

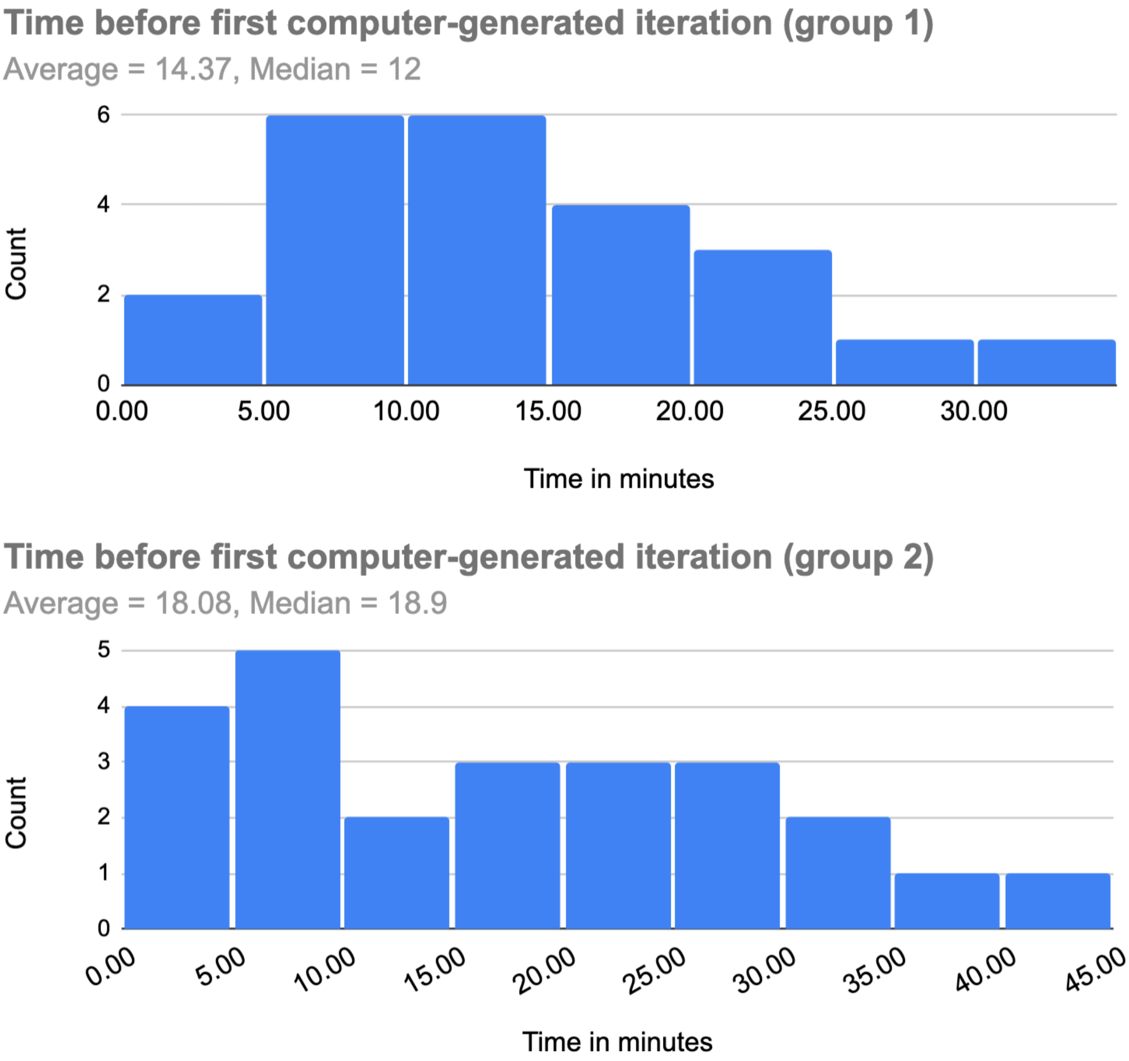


Figure 9.9.: The time taken to produce first iteration (group 1 and 2)

9.4.2. Iteration

Inspecting the time stamp of the first *iteration* (in the interaction data) reveals how long it took to interact with CGM techniques. Figure 9.9 shows that on average group 2 were slightly slower than group 1, but contained several outliers - notably several participants were far quicker than average. Both groups had access to the same tutorial material, but group 2 were given a 5-minute demonstration of IGME, therefore these are not necessarily comparable. However, in summary, users were generally able to quickly learn IGME's interface, and begin to make CGM in both the controlled or real-world conditions.

The interaction data showed that participants were triggering iterations, but not fully listening to the resultant output, instead of either triggering another iteration or returning to editing. This will be referred to as *'partial iteration'*. The results in Figure 9.10 show wide variance and large outliers (particularly in group 1), suggesting some users were keen to hear many permutations of CGM. Reasons for this phenomenon are further explored in Chapter 10.

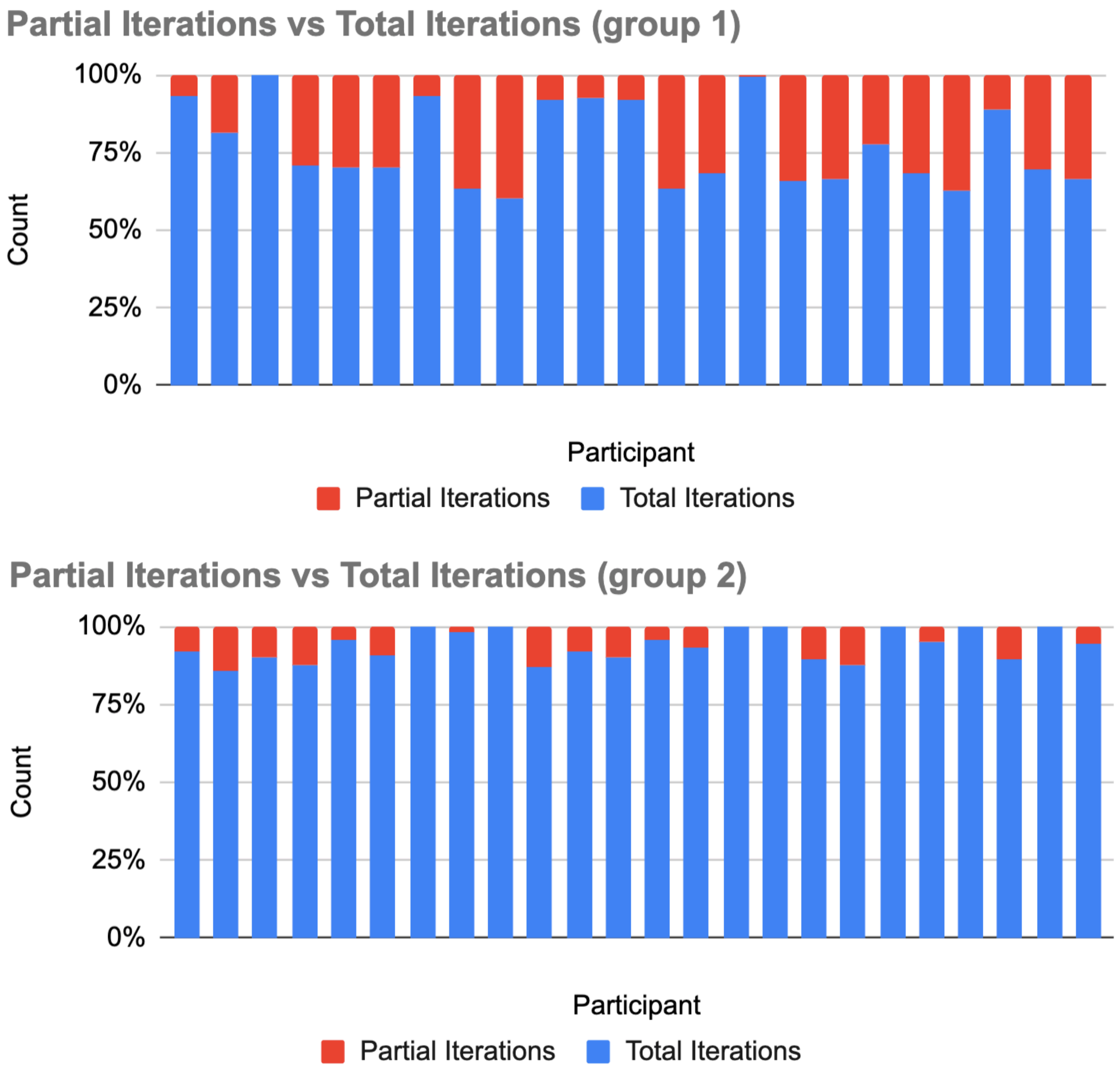


Figure 9.10.: Iteration ratio for each user.

9.5. Methodology Comparisons and Technical Difficulties

Despite the second study being similar to the first, there were notable (if unavoidable) methodological differences between the two, highlighting the opportunities and complications of running sessions remotely. The BBCU learning tool mostly proved adequate for presenting the workshop remotely.

Such studies would not have been possible without the high-quality video and audio stream afforded by high-speed internet, although not all participants had the same experience, as some reported that the transmitted system audio was lacking in quality, resulting in 'low-fidelity' sound in the music examples.

At the time of writing, the BBCU tool transmits only a mono stream (taking the left channel only) from an otherwise stereo audio application. Due to the different technologies and encoding of video/audio, audio drop-out is often more noticeable than video.

The chat window feature proved useful when setting up. For example, using the Windows 10 version of IGME requires the manual installation of a font, something that had been overlooked by the researcher. Fortunately, a participant was quick to realise the requirement and instructed others on a solution. Unlike other studies on IGME, the end-user's computer was not a controllable quality. In addition, the previous studies had exclusively used MacOS. Other issues included having to bypass gatekeeper on MacOS 10.15, due to Apple's increased security measures, something that had also been overlooked at the time.

Some participants were confused about the steps required to activate the software. Such issues could be attributed to the tools used. For the study and the software to meet ethical approval, consent had to be obtained and data had to be ring-fenced and protected, requiring several involved steps. In other in-person IGME studies, the researcher was able to ensure all users were properly set up, a luxury not afforded in this study. This might explain why users found it challenging to get started with IGME as part of the failed public beta (Chapter 7).

The primary benefit of running a user study remotely is that it opens it to a potentially much larger pool of participants, broadening access for users. Also, users would probably feel more comfortable working with their own computers in familiar surroundings, helping to further minimise the *Hawthorne* effect.

9.6. Conclusions

Similar survey results were collected for both groups. This suggests that being a participant in controlled or real-world conditions had minimal effect on the participants' survey response. This mitigates some of the concerns associated with the Hawthorne effect (McCarney *et al.*, 2007). However, the interaction data produced different results, for example, whereas all participants of group 1 remained in the session until the end and completed the survey, 71% of group 2 left and did not complete the survey. This might suggest that participants of group 1 were encouraged to stay because of remuneration, whereas group 2 were intrinsically motivated, given their attendance at the conference in which the study was held.

This chapter has shown that users would use CGM to inspire and motivate them to create initial music ideas. Users remained somewhat apprehensive about directly using the output of CGM in their own music. Participants agreed with the merits of the version control system, and two-stage editing process. Participants were quick to get started with IGME and produce iterations, likely the merits of retaining the familiar workflows and interface paradigms of existing music software permitted this.

One advantage of the controlled conditions study was that it was easy to ensure participants had installed the software correctly and that everyone completed the survey. As evidenced by discussions in section 9.2, remote study participants found it more difficult to get started, and it proved harder to engage them.

A limitation of the study was that most users spent less than one hour with the software. The next chapter discusses many of the same concepts as in this chapter but considers these with participants that engaged with IGME for longer periods. The findings of this chapter are extended through Chapter 10 before formal conclusions are drawn in Chapter 11.

Question		Group A		Group B		
		Median	Mode	Median	Mode	Median Dif
1	A: I feel that the VCS encourages me to experiment with ideas.	2	1	1	1	1
	B: I feel that the VCS allows me to easily compare different iterations.	2	2	2	1	0
	C: I feel that the VCS allows me to check my progress.	2	3	3	1	-1
	D: Using the VCS makes it easy to go back and make changes to the music.	2	2	2	1	0
2	A. The TSEP allows me to rapidly enter ideas.	2	2	2	2	0
	B. The TSEP allows me to make edits in any order.	2	2	3	3	-1
	C. The TSEP makes it easy to go back and make changes to the music.	2	1	3	3	-1
	D. I feel that the TSEP creates hidden dependencies.	3	3	3	3	0
3	A: The GP in IGME helped me to come up with new ideas.	2	1	2	2	0
	B: I find that the GP make annoying mistakes.	3	3	3	3	0
	C: I would like more control over the GP.	3	3	3	3	0
	D: I would like to be able to define my own GP.	2	2	2	2	0
	E: Using GP helped me to generate ideas that I would not have created myself.	2	2	2	2	0
4	A. I feel that it is easy to distinguish between human parts and computer parts.	2	1	2	2	0
	B. Is it easy to find the type of part I am looking for.	2	3	3	3	-1
	C. Breaking the music into parts makes it easy to try out new ideas.	2	1	2	1	0

Table 9.6.: Table of comparisons between both groups answers to the first 4 questions. *1 = strongly agree, and 5 = strongly disagree.*

Question		Group A		Group B		
		Median	Mode	Median	Mode	Median Dif
6	A. When writing music within IGME, there were difficult things to work out in my head.	3	3	2	3	1
	B. I feel that I have suitable control over the generative processes in IGME.	2	2	2	2	0
	C. I feel that the computer has taken some control of the composition process in IGME.	2	2	3	3	-1
	D. I feel that my knowledge of generative music has improved since using IGME.	2	2	2	2	0
	E. The interface provided by IGME allows me to easily explore new ideas.	2	1	2	2	0
	F. IGME's workflow is similar to other score editors and sequencers.	2	2	2	2	0
	G. The tutorial system inside IGME helped me to learn the software quickly.	1	1	2	1	-1
7	A: I would use CGM techniques to help me to come up with new ideas.	1	1	2	2	-1
	B: I would use CGM techniques to explore different permutations of my own material.	2	1	1	1	1
	C: I would use accompaniments created by the computer alongside my own composed music.	2	2	2	2	0
	D: I would use music the computer has generated in my own compositions.	2	2	1	1	1
	E: Using generative/algorithmic techniques helps to increase my productivity.	2	3	2	1	0

Table 9.7.: Table of comparisons between both groups answers to questions 6 and 7. *1 = strongly agree, and 5 = strongly disagree.*

10. Longitudinal Studies in End-User Computer-Generated Music

Previous studies of IGME were focused on short engagements, with larger sample sizes, whereas the work presented here involved more detailed use of IGME over a longer period. Screen recordings of participants were analysed and categorised into themes. Commonality between participants created discussion points on the mechanisms and techniques that enabled computer-generated music (CGM) workflows and their roles in computer-human composition.

10.1. Method

For this longitudinal study, (similar to those discussed in Chapter 9), people who had taken part in an earlier IGME study were contacted. Of those, 4 completed the study. Figure 10.1 shows an overview of how participants progressed from initial studies (Chapter 9) through to the longitudinal study. A pilot study was undertaken with an additional individual, to test and evaluate the entire process and subsequent methodology. Everyone who volunteered for the study received a £40 Amazon voucher.

The brief for the four participants was: *"compose music while considering the computer (IGME) as a collaborator"*. Each participant was given an anonymised name: Cameron, Tim, Gary, and Liam – to help with identification and general discussion.

Participants used IGME for 4 hours, split into two 2-hour sessions one week apart. They were able to reference tutorials supplied via a website[1]. The researcher was physically present throughout to answer questions but otherwise remained a passive observer[2], permitting further notes to be jotted down.

[1] The tutorial material can be viewed at Hunt (2021).

[2] Exceptions are discussed at the end of this chapter.

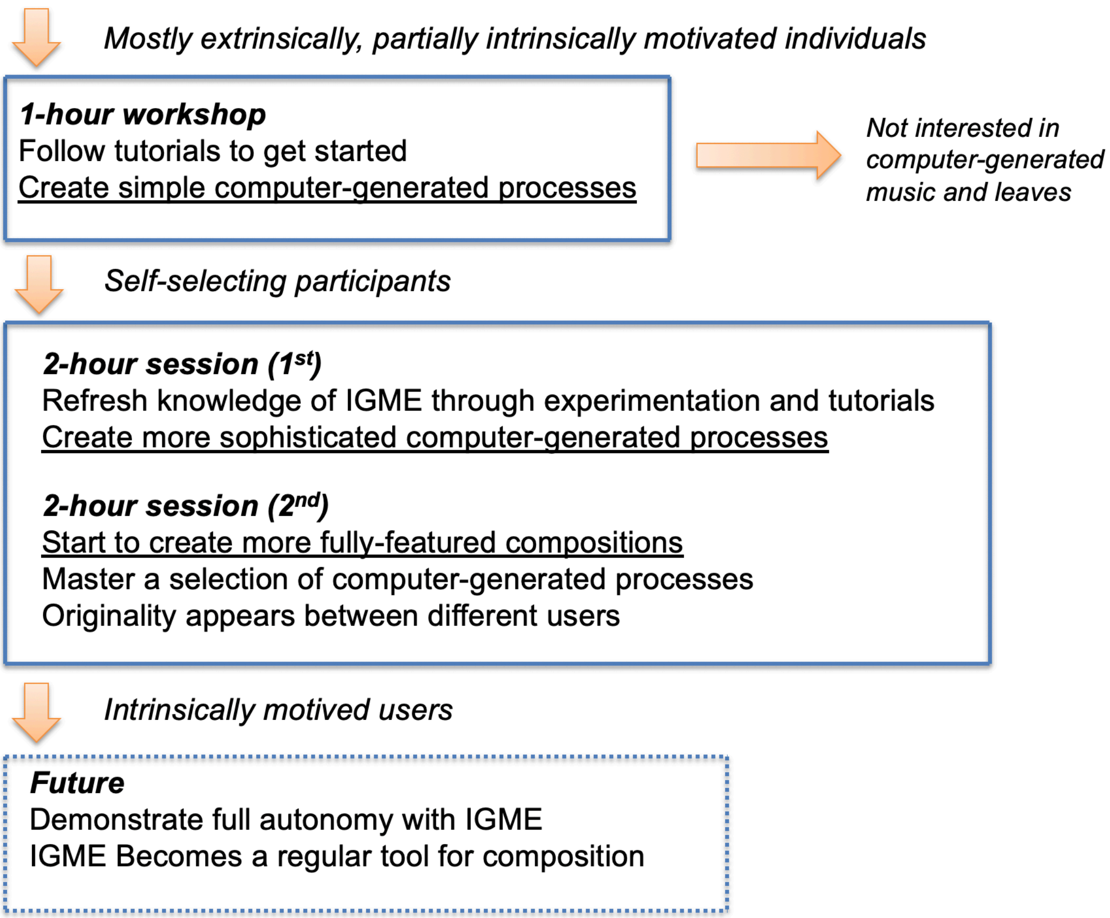

Mostly extrinsically, partially intrinsically motivated individuals
1-hour workshop
Follow tutorials to get started
Create simple computer-generated processes
Not interested in computer-generated music and leaves
Self-selecting participants
2-hour session (1st)
Refresh knowledge of IGME through experimentation and tutorials
Create more sophisticated computer-generated processes
2-hour session (2nd)
Start to create more fully-featured compositions
Master a selection of computer-generated processes
Originality appears between different users
Intrinsically motived users
Future
Demonstrate full autonomy with IGME
IGME Becomes a regular tool for composition

Figure 10.1.: A flowchart showing participant selection and motivation.

Screen-recording software, capturing internal audio output and microphone input, was used. As with other studies involving IGME, music and interaction data was logged and sent remotely. The music created by each participant was analysed and discussed. Cameron completed one composition across the two sessions. Tim and Liam each completed one composition per session while Gary created four smaller pieces overall. All produced smaller compositions at the start whilst learning IGME. However, due to space constraints, these are not discussed here.

10.1.1. Surveys

Participants were given survey C to complete before the study, and survey B2 at the end of each 2-hour session, revealing how their opinions modulate over time.

The results from survey C show the participants' backgrounds revealing they were experienced, music practitioners. All could play at least one instrument and had spent between 3-10 years composing music. Three of the four declared themselves *'expert'* in one or more types of music software and said they balanced their compositional activities between the computer and non-computer (i.e. playing an instrument). In contrast to the findings in section 10.2.4 (problem-solving theme), three said their composition was not guided by their knowledge of music theory. This statement is conflicting as users still need some knowledge of music theory (i.e. editing and reading scores) to use IGME. Similarly, the wording of this question implies that users follow music theory explicitly (i.e. no accidentals), but, as stated previously, music is as much about breaking *'rules'* as following them. Therefore, the interpretation of the question may provide some of the reasoning behind this disparity. Similar to the findings in Chapter 5, users had used arpeggiators, minimalist, and serialist processes, but had used neither neural networks nor Markov models. Figures supporting this discussion are shown in Appendix E (Figures 10 - 15).

10.1.2. Technology

Video recordings and screen recordings are particularly useful for logging human-computer iterations (Preece, Sharp, and Rogers, 2015) as they produce an objective record of events while they unfold, and permit the unobtrusive collection of intricate computer work (Tang *et al.*, 2006). FitzGerald (2012) introduces the field of video analysis, which is discussed in more depth by Erickson (2006). Video analysis is useful for studying interaction and performance in music contexts. For example, Xambó *et al.* (2013a) studied groups interacting with the *Reactable* instrument. The observation of recorded interaction can be quantified (how many times did this interaction happen). However, qualitative descriptors are more common.

Xambó *et al.* (2013b) notes that video analysis methodologies are not formally established. For example, Coorevits *et al.* (2015) analyses video footage of guitarists, applying themes to the musical expressions observed. Although the methodology is not a formal thematic analysis (discussed shortly) it has many parallels.

10.1.3. Thematic Analysis

Rosala (2019) defines thematic analysis as a *"systematic method of breaking down and organizing rich data from qualitative research by tagging individual observations and quotations with appropriate codes, to facilitate the discovery of significant themes"*. Thematic analysis is normally used for text-based data, such as interview transcripts, but can be applied to any qualitative data.

Braun and Clarke (2006) define 6 steps for such analysis:

1. Familiarising yourself with the data.
2. Generating initial codes.
3. Searching for themes.
4. Reviewing themes.
5. Defining and naming themes.
6. Producing the report.

Data can be analysed bottom-up (inductive) or top-down (theoretical or deductive). Inductive means themes are directly linked to the data, allowing themes to develop through the data itself. Deductive can result in mislabelled data, especially when data must fit within pre-selected themes, theories or hypotheses. Thematic analysis allows for the discovery of *'answers'* when the questions are not necessarily known. Tanaka *et al.* (2012) used thematic analysis to examine the results of a user survey on mobile music GUIs, to reveal themes not explicitly targeted by the survey.

The video interaction descriptions (discussed shortly) are qualitative and as such are examined under a thematic analysis framework. This research was intended to be as open as possible, to capture themes that the research may not have originally hypothesised.

The data collected from the video was mostly qualitative descriptions of interaction, with few, if any, spoken observations. This rules out two commonly used methodologies for qualitative research - discourse analysis and grounded theory. Discourse analysis is primarily used for the analysis of text or spoken language (Thompson, 2002). Likewise grounded theory (Grossoehme, 2014) requires primarily interview data and necessitates the data collection continues until hitting a saturation point (when no new information is being learned), subsequently requiring more participants than this study had. It also assumes from the outset the researchers don't have any existing theories[3].

10.1.4. Encoding Observations

Interactions observed in the video were logged (text) in a spreadsheet along with a timestamp, the corresponding IGME iteration (if applicable), and interaction type (e.g. spoken, physical). Between 720 and 780 observations were recorded for each participant and summarised as high-level user interactions. For example, *"user added a new part t1p1"*, rather than; *"user-selected*

[3] The research in this section builds upon the findings of the previous studies, so negates this.

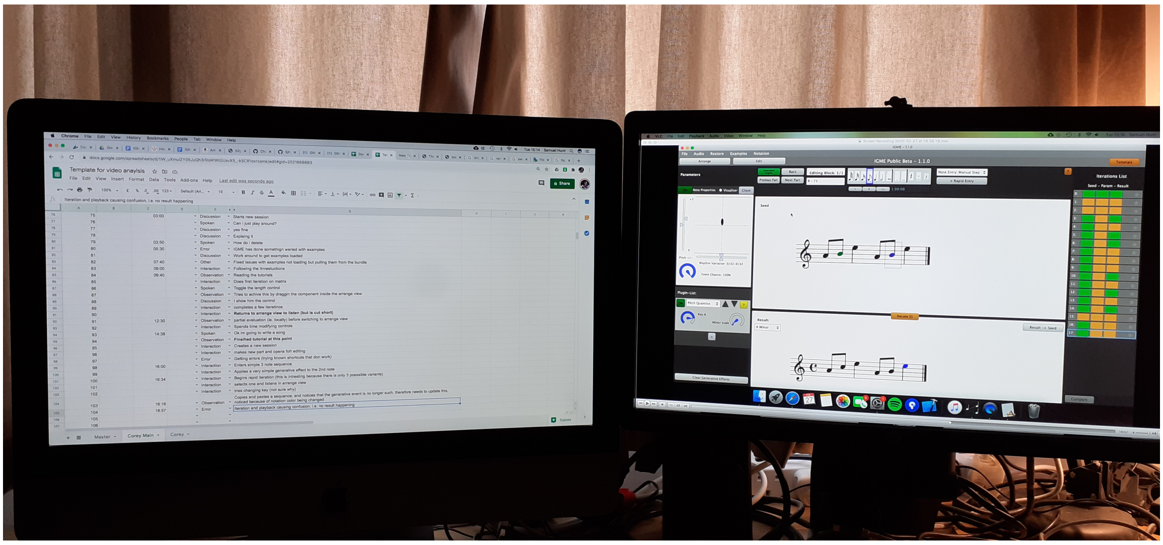


Figure 10.2.: A photo showing the physical set-up for analysing video footage, data sheet (left), video recording (right).

track 1, bar 1, right-clicked and selected add part". The spoken dialogue was similarly summarised[4]. The lower-level granularity of interaction was captured through the internal IGME logging system.

Each observation was evaluated and tagged with a mid-level theme summarising the user's actions. General themes were grouped and described the users' interaction routines with IGME. This bottom-up approach allowed for the discovery of themes, rather than for high-level themes to have been assumed at the outset.

10.2. Resultant Themes

The following section details each theme, with the discussion points creating a lens through which the interaction and music data were analysed. Although listed individually, the themes have many cross-over points. The seven themes to emerge from the study were:

A Human-computer composed music.

B Edit, iteration, and evaluation cycles.

C Familiar; plug-ins, workflows, and routines.

D Music Theory and problem-solving.

E Version control.

[4] Examining the entropy of verbatim dialogue was deemed unnecessary for this study and would have been excessively time-consuming.

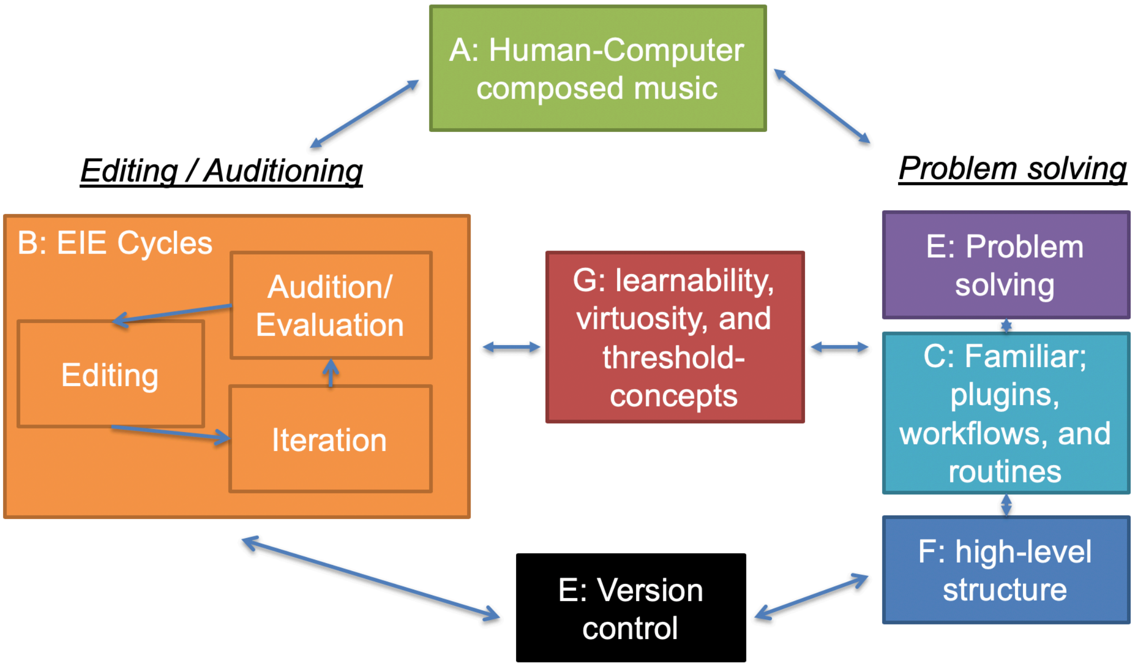


Figure 10.3.: A diagram showing a visualisation of the resultant themes.

F High-level structure.

G Learnability, virtuosity, and threshold concepts.

The work here is examined through a computer-generated music lens, however, the more general discussions around computer music composition might also be useful for the field. The themes are visualised in Figure 10.3.

10.2.1. A: Human-Computer Composed Music

"Users interacting with IGME use it to generate new ideas, or create variants of existing ones."

Processes employed by users for working with generative music were varied and nuanced, with some concepts common not just across this study but across studies discussed previously. The computer's main role, other than for music sequencing elements, was to act as a catalyst for creativity, aiding rather than replacing human creativity. The composition process was made more complex because, instead of just arranging notes, users had to arrange and debug computer-generated processes. They also had to juxtapose human and computer output, bouncing between the role of composer, curator, and music appraiser.

	Human	Human-computer	Computer	Reference	Iterator	Repeater
Median length	0.81	1.00	1.00	1.00	1.00	4.00
Mean length	0.89	1.07	2.37	1.19	0.96	3.06
Sample Size	501	1902	333	196	187	32

Table 10.1.: Average part length by type and sample size.

Using the computer to come up with ideas and, more frequently, using it to develop existing ideas were the two core workflows that emerged from examining the uses for CGM. Participants who worked with IGME found the computer was able to compose some of the lower-level (phrase) content, freeing them to evaluate those creations and then sequence different iterations together. One user said: *"I am enjoying not having to think about melody much because I am letting the computer do it"*. Users commonly chained up more than one computer-generated process. For example, they used the note parameter editor to vary a sequence and then applied (e.g. pitch quantiser) plug-ins to constrain it.

The varied roles for CGM were often materialised through different processes, Liam noted that, upon trying two plug-ins, the *wind chime* plug-in simply produced a rearrangement of what he had already composed, whereas the *transition table* created new material with similar musical characteristics. Understanding the different processes enabled participants to determine which should be used for the current composition task (discussed more in theme D).

The composition was developed in a mostly linear fashion, with users working on one musical part before moving on to another. Parts remained short in length (usually a single 4/4 bar), as illustrated in Figure 10.4 which shows the distribution for the length of the music grouped by part type, and Table 10.1 shows the median, mean, and sample size for each type. Human parts tended to be the smallest at around or just under a bar, while human-computer, reference and iterator parts were slightly longer. This is likely due to them being used to model high-level structures. The results for the computer part were significantly longer, but the sample size was considerably smaller. The role of the repeater part would appear to be for *'chunking'* together musical structure.

Of all the music created during the sessions, only 14 human-computer parts and 5 computer-parts were over 10 bars in length, accounting for just 0.63% of the total. These are not included in the results above as they are considered extreme anomalies.

The manual creation of simple sequences and the use of generative processes to modify them, most frequently with the note parameter editor (see theme C for plug-in use), were the most common workflows. Several of IGME's plug-ins, namely the transition table (TT) and distribution

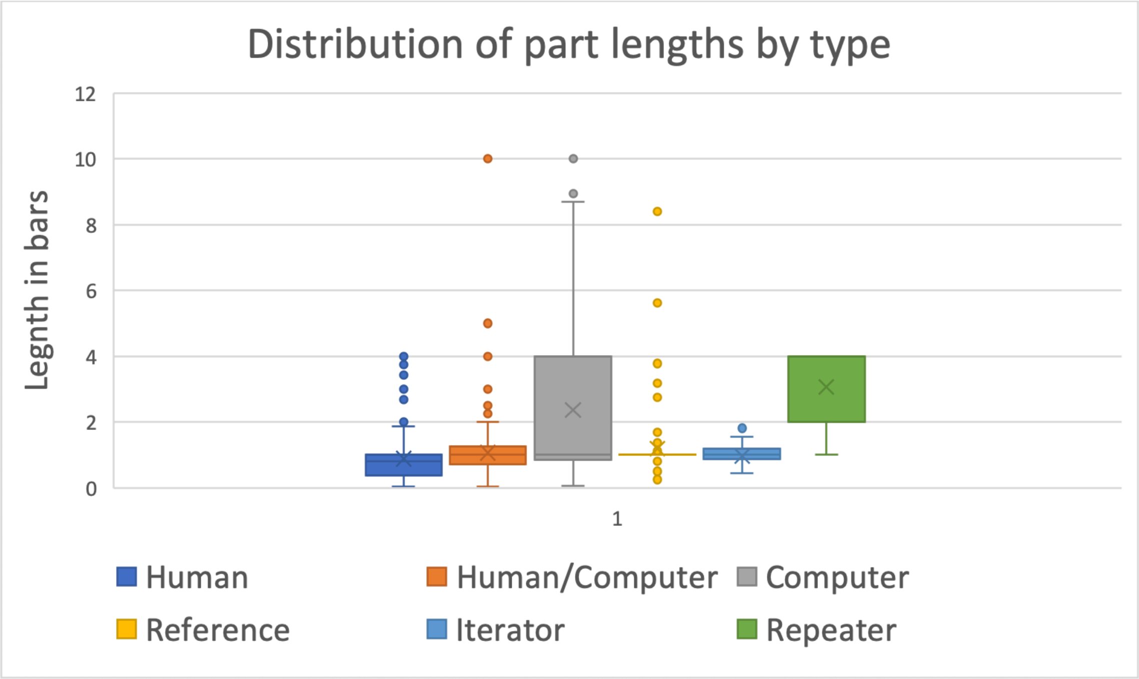


Figure 10.4.: A distribution of part result length, grouped by part type.

sample (DS), were used to produce music with similar characteristics to the user's music. By analysing existing music, either by specifying a range from a given track or by analysing the entire session, users could generate new music in a similar style, and still had the option of adjusting or tweaking the parameters.

Gary had been initially dismissive of the DS plug-in and had struggled to find a meaningful use for CGM. However, at the end of session 2, having composed a lot of music manually, he used the DS plug-in and, after tweaking various parameters, stated *"this is now very useful because it (the computer) knows what I want. I understand the IGME processes more, whereas before I found it hard to make use of ideas from nothing, but I find it useful when I already have ideas in the piece"*.

Gary's response was shared by other participants, suggesting that, of the two main uses for generative music, the ability to work with existing music is preferable to generating it from scratch. This finding is discussed again in Chapter 11. Using the computer to generate an idea from scratch was mainly achieved through the seed generator plug-ins. Cameron noted that this process was quite quick: *"you just fiddle with some knobs, you can get something decent quite quickly"*.

Cameron noted that trying a process such as retrograde in Sibelius meant applying the process manually, whereas with IGME it was possible to listen to the output of various processes and pick the one that sounded best rather than simply following the algorithm.

The part conversion was a process whereby a computer-generated, or reference part was converted back to a human part for more fine-grain editing. This process could be described as *taking ownership*, with the computer's compositional role no longer required. This process allowed any small mistakes by the computer to be easily rectified, rather than continuously re-iterating. Users expressed frustration at this process, often wanting to regenerate only a subdivision of a bar while retaining the fragments they were happy with, possibly suggesting that part conversion was the only option (problem-solving) and not the preferred approach.

Cameron stated *"I like to humanise a part"*, by converting a computer-generated idea into a human part. At this point, he remarked *"I guess I own it"*. At the end of the session, Cameron commented: *"it did end up sounding like my music"*. When asked, *"do you attribute this to yourself?"*, the answer was: *"it's definitely my music"*. These statements emphasise that authorship is not a concern for this user[5].

10.2.2. B: Edit, Iteration, and Evaluation Cycles

"Users composing music, configured effects and evaluated the output using an iterative process."

Edit-iteration-evaluation (EIE) cycles were a recurrent theme in which users edited either a sequence of notes and/or computer-generated model parameters, computed an iteration, and then evaluated this through auditioning it and visually inspecting the notation. Therefore this theme combined editing, iterating and evaluating. The differences between simple playback and iteration caused initial confusion to participants in all the studies and became an identified threshold concept (see theme G). Figure 10.5 shows an annotated visualisation of the interaction data, showing users editing notes, doing iterations, and listening to the output.

A rapid EIE cycle was a common workflow in which users rapidly iterated output, producing many variants of a generative model, often coupled with editing notes and/or parameters. This phenomenon was observed in all users. The rapid EIE cycles can be primarily attributed to the two-stage editing process and the VCS. Although playing mostly a passive role within EIE, the VCS was a safety net and also offered the ability to revisit the musical material, encouraging users to work rapidly, exploring the limits of a given generative process. As noted by Nash (2011), existing theories of flow emphasise the importance of rapid edit-audition cycles.

[5] Themes associated with authorship are explored again in Chapter 11.

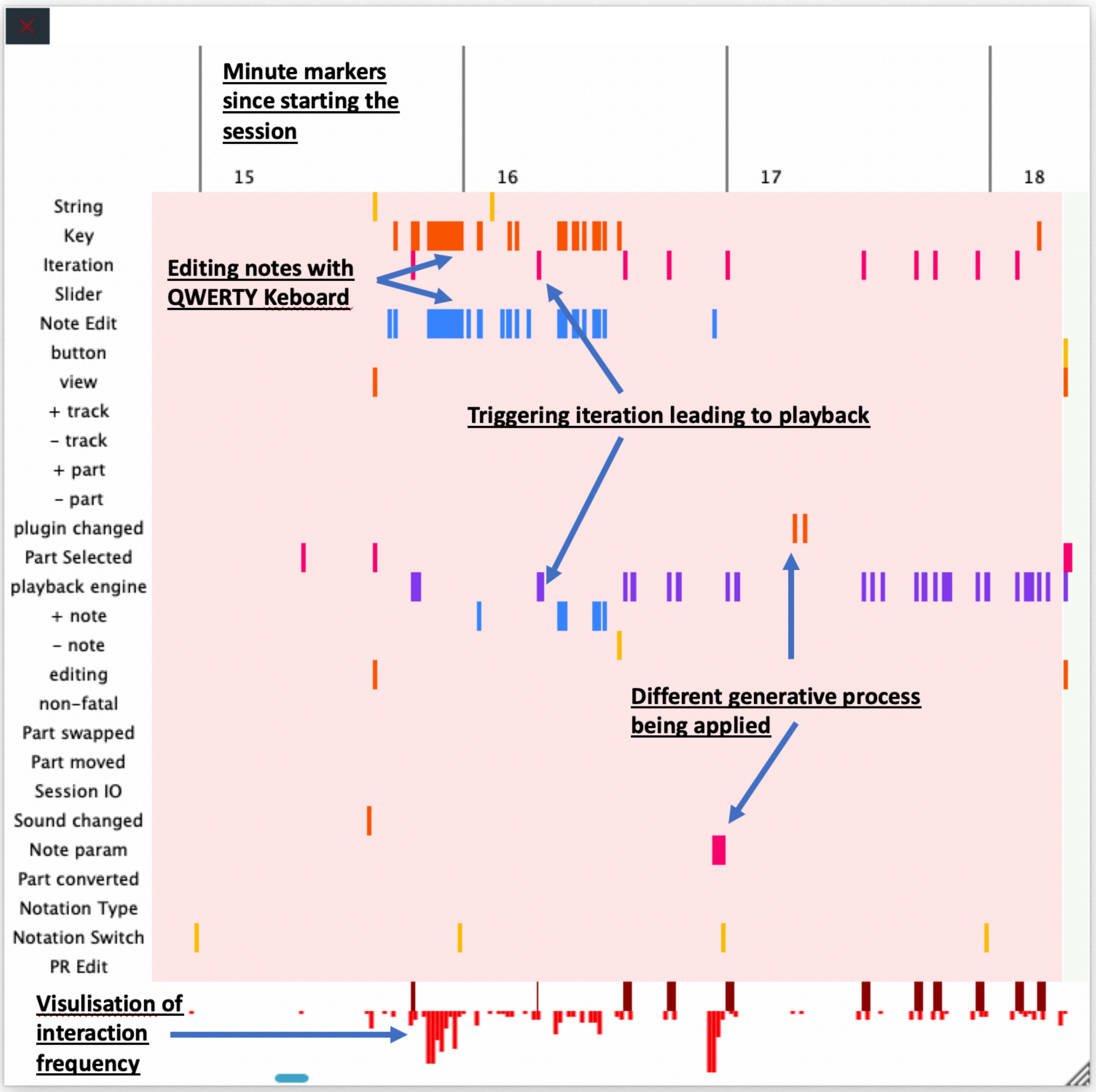


Figure 10.5.: Visualisation of a general edit-iteration cycle inside the IGME interaction analyser.

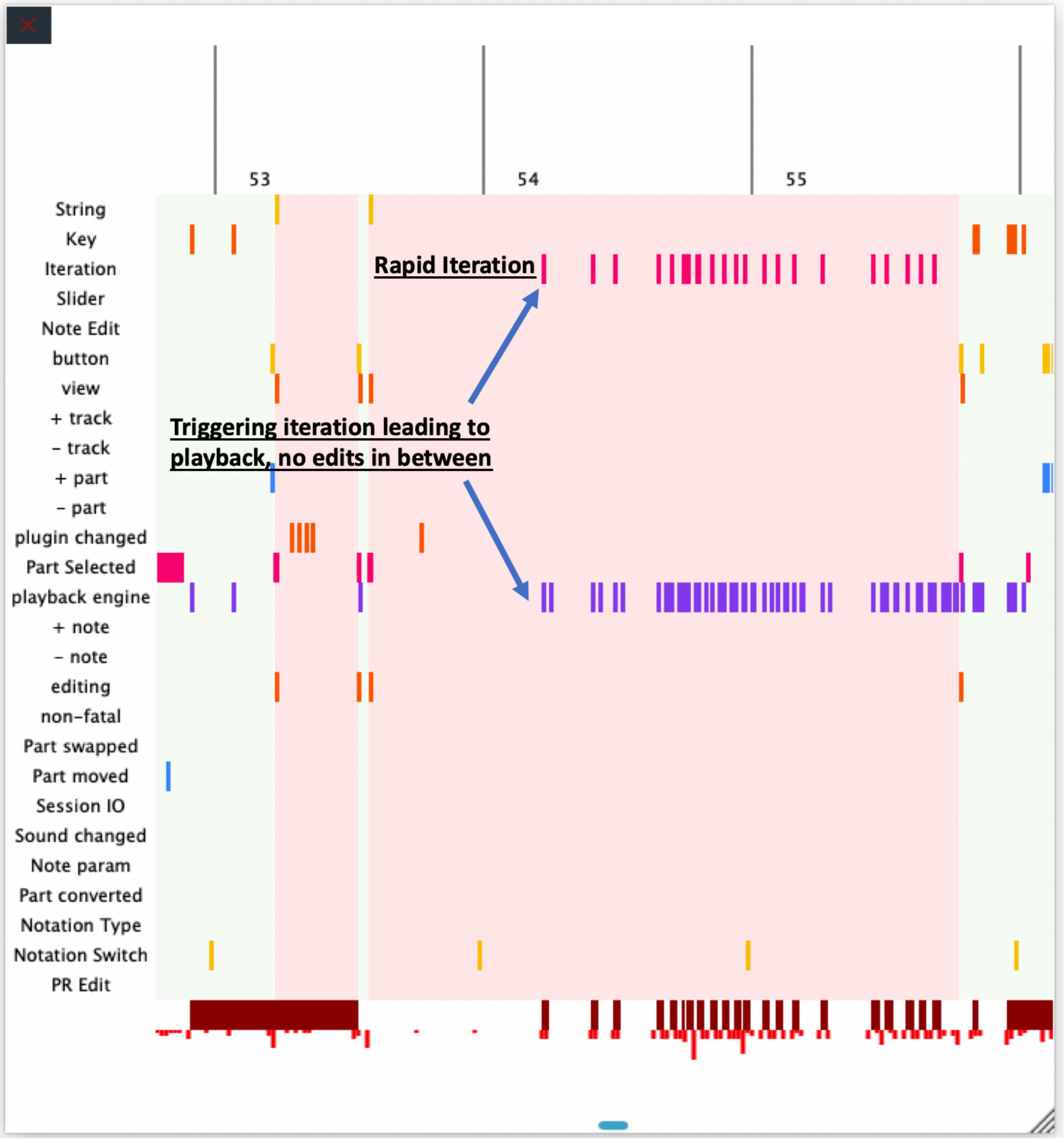


Figure 10.6.: Visualisation of a rapid iteration, few if any edits happen.

Occasionally, users triggered rapid iteration, as the output had not changed, either because an effect had not been applied or had been inadvertently disabled, requiring users to problem solve (see theme D), although this was an anomaly rather than a common reason for rapid iteration. As Cameron noted, it was not always possible to tell if there was any variance between iterations.

During EIE cycles, users would often terminate playback of output, because they had requested another iteration be produced, resulting in rapid iteration and partial evaluation (RIPE). There may be three possible reasons for this. The first is enjoyment, with users wanting to hear many different versions of the generative process they had configured. The second reason is that users could see issues with the score[6], such as a high occurrence of accidentals or large melodic leaps, and quickly deemed this unacceptable. The third reason was that users were unable to find a suitable iteration, although this is less likely, as they could just have returned to editing.

During Session 1, Tim generated 27 iterations in under a minute, which was unlikely to have happened with regular forms of composition. An example of this, seen in the IGME interaction analyser, is given in Figure 10.6. Tim later stated *"I just keep going through different iterations until I get something that I like"*. This was Possibly a criticism of CGM, highlighting the reliance on 'chance' in that the *'perfect'* musical content will possibly appear but is not guaranteed.

As the music developed, the frequency of iterations tended to tail off, resulting in a more contemplative EIE cycle, in which most musical output was auditioned in full. Users tended to jump backwards and forwards between the editor (auditioning solo) and the timeline (auditioning in context), sometimes generating new iterations locally. They were critical of this division of editing and arranging. Parts were developed in isolation, but additional steps were required to audition newly-created parts alongside material on the timeline. As the session neared its conclusion, longer periods of playback increased in frequency, probably an inherent side effect of having a more complete (longer) piece. As users progressed through iterations, parameter ranges become more constrained, narrowing the stochastic search space for computer-generated output. This was particularly observed with the note parameter editor[7].

[6] The ability to recognise issues with visual notation from existing knowledge of music theory and debug generative processes is discussed in theme D.

[7] This was observed throughout the video but exact patterns were harder to discover in the *'noisy'* interaction data.

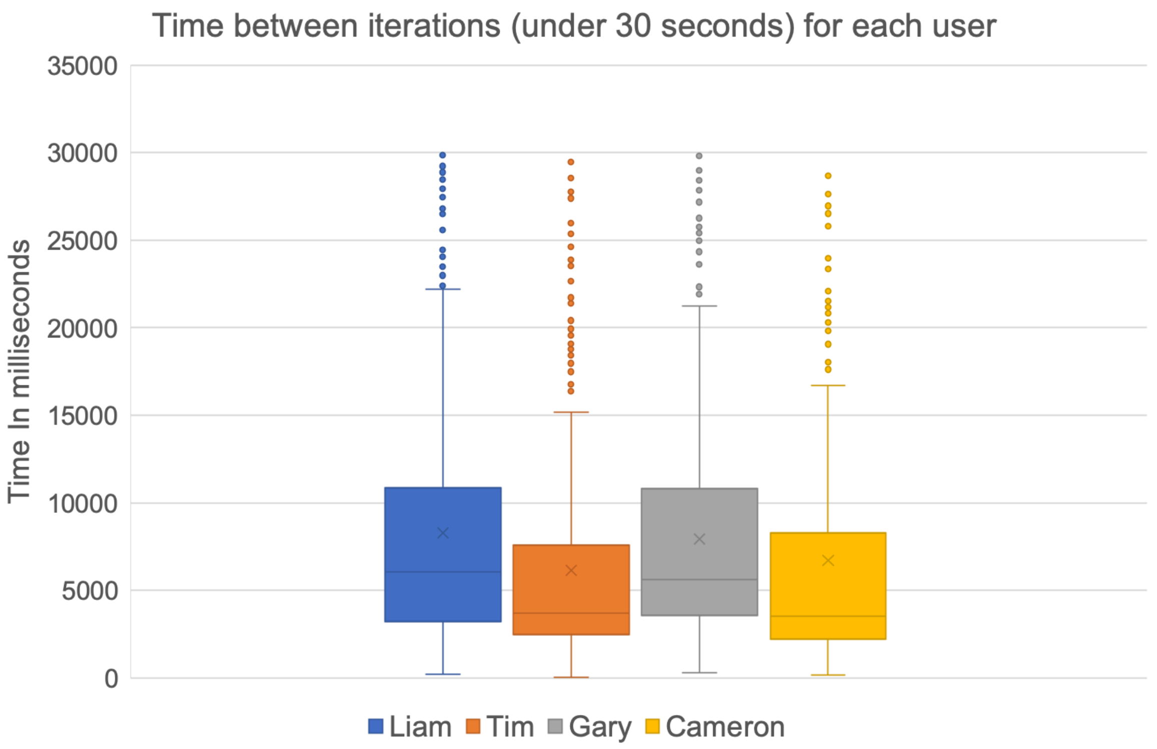


Figure 10.7.: Box and whisker plot of the difference in ms between iterations for each user. Excludes outliers above 30 seconds.

Users were often quick to audition edits. However, sometimes a series of edits or high-level processes that did not produce an immediate audible effect would be completed before auditioning. This suggested that either users were confident that processes would work without error, or that it was worthwhile to execute a pre-determined mental mind map (theme D) requiring multiple steps before an audition. Not all the evaluation was audible, as visualisation of the score, arrangement on the timeline, and explicit dependencies (theme F) were also mechanisms for evaluation.

The interaction data revealed the frequency of iterations, and also the amount of playback triggered by the user. Figure 10.7 shows a box and whisker plot for the interval in milliseconds between subsequent iterations occurring less than 30 seconds apart. Table 10.2 shows the total iterations for each participant, split into defined types. The median for all participants (Figure 10.7) was between 4 and 6 seconds, suggesting that rapid iteration was a common occurrence. Partial iteration was seen in all users, but especially for Liam and Tim, with 16% and 18% respectively, meaning they had triggered another iteration before fully auditioning the subsequent output. Arrange view iterations were recorded when the user iterated on the timeline, triggering all unlocked parts to be reiterated, although this was an uncommon occurrence.

		Name			
Group	**Observation**	Cam	Gary	Tim	Liam
Iterations	Total	392	569	540	676
	Partial	19	19	98	110
	Arrange	11	6	0	13
Time in (minutes)	Playback	44	53	29	48
	Arrange view	157	137	71	89
	Edit view	63	87	132	121
	Total	220	223	203	211
Tracks	Added	9	13	10	13
	Removed	1	1	0	4
Parts	Added	151	118	134	88
	Removed	72	60	44	28
	Converted	19	20	0	7
Edited	Sounds	403	446	72	249
	Note parameters	3741	496	16028	12716
	Notes	668	1444	2583	3365
View switched focus		343	416	200	213
Playback engine state changes		3077	4012	3582	3586

Table 10.2.: Various metrics captured from the interaction data (both session are summed).

Figure 10.8 illustrates how many iterations per part were needed to finalise or arrive at a desirable solution. Human parts have low iterations[8], compared with human-computer, and computer parts, i.e. fewer revisions are needed to achieve suitable content. This might suggest that the user is in control of human parts, leading to fewer iterations, whereas the stochastic processes of human-computer and computer parts leads to a higher frequency of iterations. This creates tension between a user's ability to control either human or computer parts. On average, iterator and reference parts have low total iterations, as these tend to develop rather than create new music (discussed in theme F).

[8] Iteration for a human part means users made a series of edits and then audition, registering a new iteration in the VCS, repeated auditions do not constitute a new iteration.

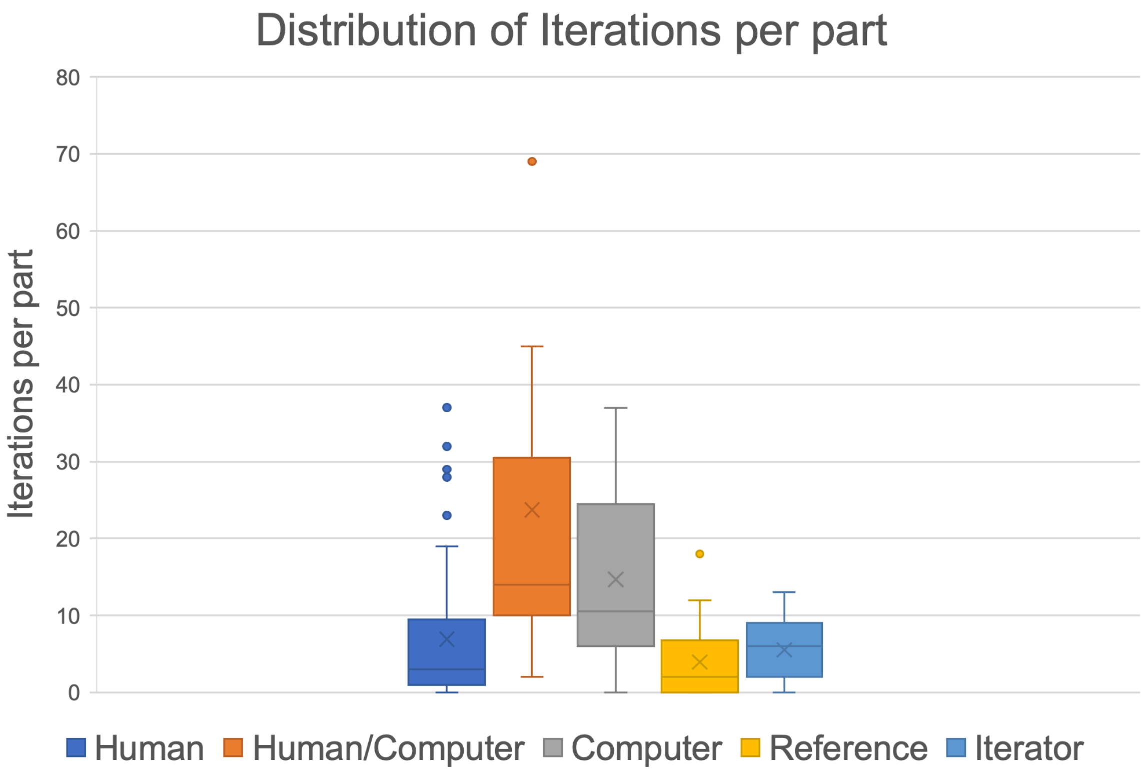


Figure 10.8.: Iterations per part type.

10.2.3. C: Familiar; Plug-ins, Workflows, and Routines.

"A user applied a sense of consistency across their choice and application of plug-ins, workflows, and routines."

All users had prior experience with digital music composition. Within IGME, they applied established routines for digital music composition and found ways to combine IGME's unique features and workflows. Although keen to explore the various plug-ins and effects inside IGME, they seemed to settle on a small number of these, focusing on mastering and exploring the limits of each. This resulted in the same set of procedures being applied across several parts. Although at first, this appeared to be limiting, it made the musical structure more cohesive. Sometimes, users would close a plug-in and then return to one they had already learnt, possibly suggesting they were less motivated to learn new processes.

CGM can be criticised for its lack of both high-level structure and overall musical coherence (discussed in Chapter 3). Users were careful to reuse compositional constructs, working with similar parameters across different generative effects and parts. The tendency to take a conscious approach to composition has many parallels with theme F (high-level structure).

Explicit examples of musical consistency include users making constant use of the *pitch quantiser* and *constrainer* plug-ins. The former ensured sequences stayed in key/scale while the latter limited the overall note range and ensured sequence lengths remained quantised to a bar. Often, when opening a new part, these plug-ins were added before anything else. This might have artificially restricted musical variety, but also highlighted how constrained CGM processes need to be effective. Such constraints allowed working with CGM to remain a manageable objective.

From analysing each of the nine compositions produced, the ratio of different computer-generated processes used to produce iterations is shown in Figures 10.9 and 10.10. The values indicate that for example, 45.8% of the iterations Gary produced used the note parameters process. The results emphasise that once a user understood a given process they preferred to master this rather than learn a new one.

Figure 10.11, summarises all the data and shows that the note parameter model was the most commonly used process for computer-generated music, followed by the pitch quantiser. The distribution sample was the most commonly used seed generator. While the note map, L-system, and rhythm quantizer were not used (excluded from the diagram). The note parameter process

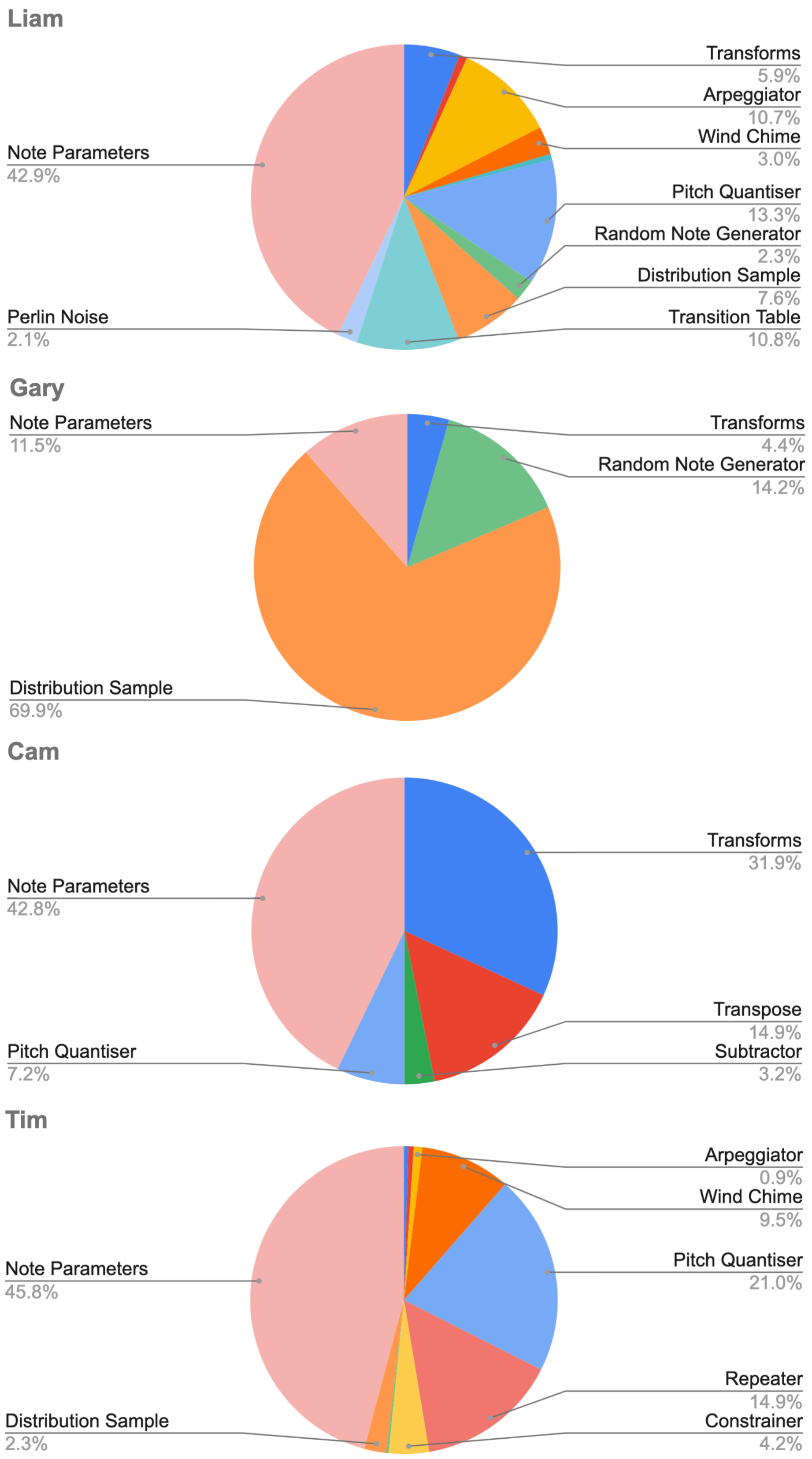


Figure 10.9.: Ratio of different computer-generated processes used by each participant.

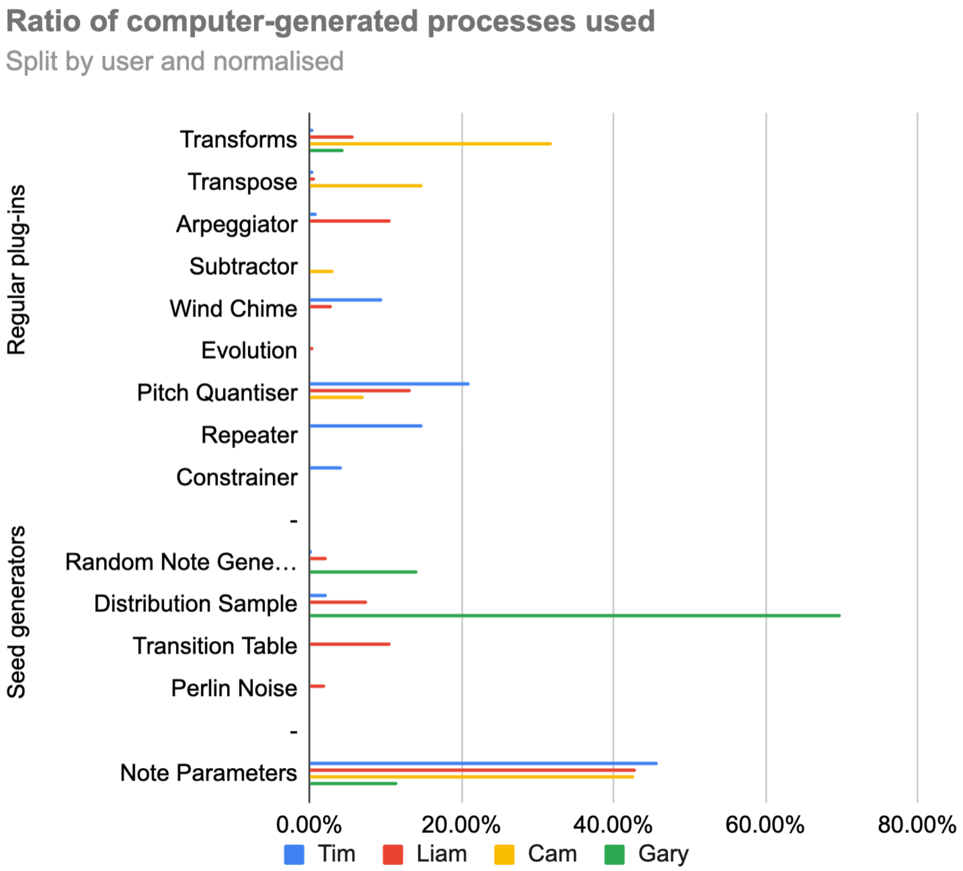


Figure 10.10.: Ratio of different computer-generated processes used by each participant (comparison).

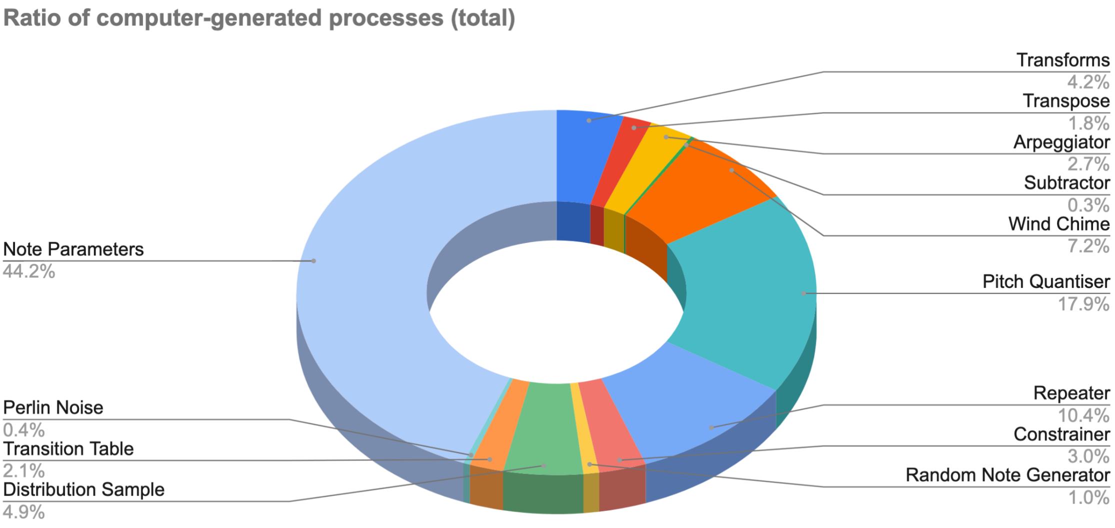


Figure 10.11.: Ratio of different computer-generated processes overall.

permitted per-note stochastic effects, unlike other plug-ins, suggesting users wanted more control than that afforded by other processes. As a result, this plug-in was used frequently, as it could also be easily be combined with other processes.

10.2.4. D: Music Theory and Problem Solving

“A user’s experience of IGME is dependent on their knowledge of music theory. It is used to debug, evaluate and critically analyse issues with CGM.”

The ability to solve problems was a key theme identified from the observation of users in both problem-seeking and problem-solving domains. Problem-solving was divided into three categories: debugging an undesirable generative output (or finding out why there was no output at all), working out how to construct a computer-generated process, and general usability issues. With all three, there was frustration when things did not work out.

Problem-solving and learnability (theme G) are interrelated, i.e. problems cannot be solved without learning the tools and relevant threshold concepts. Users solved high-level problems with various high-level tools (discussed in theme F).

A computer-generated process synthesising only a single note would be a trivial problem to solve. This was a common occurrence with some examples being; users turning off a computer-generated effect or applying a note-parameter effect with no notes selected, although both of these might be general usability issues. The resultant output was not always obvious, leading users to move parameters to the extreme to check the effect was actually working. This subsequently led to examples of unconventional music (discussed in section 10.3).

More notably, users relied on their music theory knowledge while working with generative processes, as evidenced by comments like *'the piece has too large melodic intervals (jumpy)'; 'the output is too simple'; 'notes are in an inappropriate range';* and *'this sequence has no tonal centre (key).'* Such observations came from both visual (inspecting the notation) and audible perspectives. Another observation concerned two tracks clashing as they occupied the same note register, leading to the user transposing one part.

Working with the DS plug-in, Cameron wanted to explicitly generate within a given key, stating the sequence: *"yeah tone, tone, semitone... emphasise the dominant and subdominant"* as he went, configuring the pitch profile accordingly.

Tim wanted to transpose by a value that exceeded the limits of the transposer plug-in (+24 semitones) so simply added two of them in a series to achieve a +48 semitone transposition. Although users found ways of working creatively around limitations, not all experiments were successful. For example, Liam added two arpeggiators in a series, having no meaningful effect (in this context). Unaware of this, Liam remained satisfied.

It was clear that users needed (either explicit or implicit) knowledge of music theory and that working with CGM is not suitable for those without such knowledge[9], as is sometimes true of more general music practice.

Mental Mind Maps

Users often had a mental mind map of what they wanted to create in IGME. They were able to realise plans through a combination of solving problems and their knowledge of music composition and IGME, as demonstrated by observations from the video, sometimes in the form of questions requiring answers, and sometimes thinking aloud. Cameron made a plan using an external drawing program and then worked his way through it in IGME (see Figure 10.12). Gary left empty parts across the timeline as placeholders, suggesting a delegation of parts of the song to be either human or computer. The pauses that were observed throughout the study indicated that users needed time to think.

[9] Although AI may be able to become a *"musical style spell checker"* in the future.

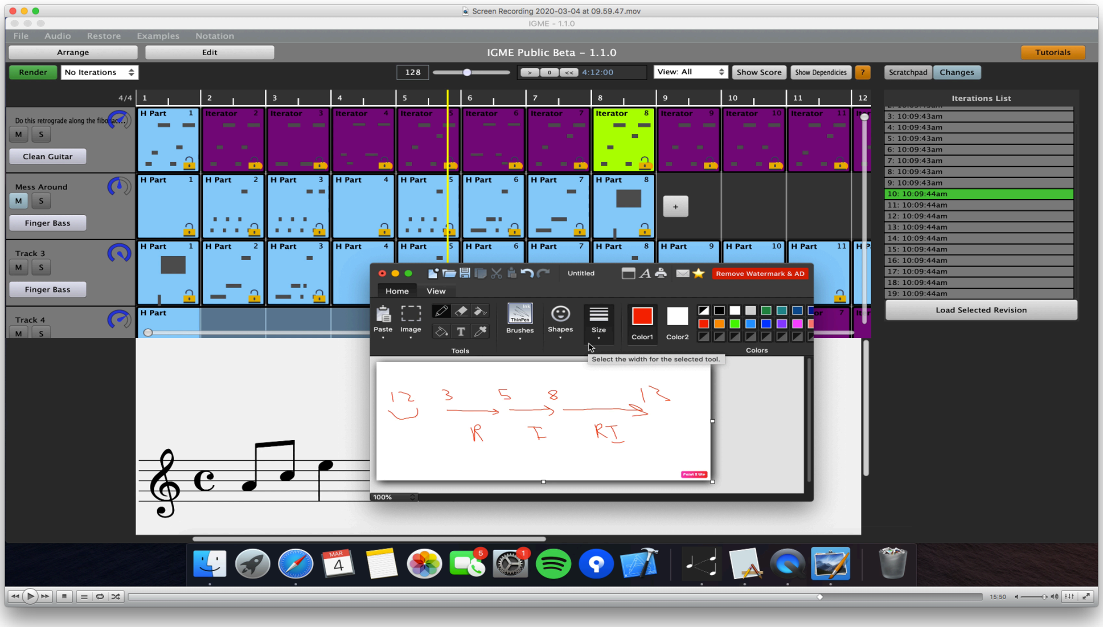


Figure 10.12.: Cameron's compositional plan - captured in paint.

10.2.5. E: Version Control

"Version control is used to cache existing material and recall previous versions."

The version control system (VCS) was considered as both a passive tool and an active tool. In a passive role, this created a virtual safety net, giving users confidence that everything was being retained, and a catalyst in enabling the rapid iteration discussed previously. Despite the safety net, the version history was sometimes lost completely, such as when converting from one part type to another.

As an active tool, VCS enabled users to recall versions, make comparisons, and check for duplicates. The inbuilt *'diff'* tool[10] was not utilised by any users (discussed in Chapter 6).

After a series of rapid iterations, and trying and failing to get a *desired* result, users sometimes revisited the list to check the suitability of a previous idea. The starring system enabled iterations to be marked (favouriting), although only Tim used this. He said he had marked iterations in order to highlight interesting ones (to be re-auditioned) and also to confirm what iteration was wanted[11].

[10] A tool used in existing version control systems for comparing the changes between two files.

[11] However, the selection of any iteration in the VCS would also have achieved this.

In another novel approach, Cameron used a single generative process to create different iterations and then copied and pasted variants from the VCS to form additional parts on the timeline.

Some participants used the scratchpad to cache previously created parts. They generally found this useful. Gary's statement that they *"had not wanted such a feature"* supported the argument that users do not always know what they want (Nielsen, 1994). Before learning the intricacies of IGME, Gary had moved parts into the future on the timeline (caching ideas), a rudimentary form of version control. Likewise, Cameron had created a track named *'mess around'*.

Although the video made it possible to see the VCS in action, no data was logged concerning, loading from or comparing iterations. The logging of such interactions was overlooked by the researcher. However, an inspection of each user's composition, to check that the final version in each part was not the last entry in the list, revealed if a previous version had been recalled. Four of the nine compositions created in the study had not used any version recall while three had used it just once. In Tim's 2nd composition, 9/66 and 8/36 parts used a recall. Although the sample size was too small to draw any meaningful conclusions, the study showed that the VCS was occasionally used in an active capacity.

10.2.6. F: High-Level Structure

"Users develop high-level structure using the features of IGME (i.e. reference parts)".

After developing music at the part (micro) level, users were required, as with other music sequencing workflows, to arrange and sequence parts into tracks across a timeline, producing high-level (macro) structure. Another set of IGME's unique features was prescribed. Specifically, a set of mechanisms, built on the idea of a *'reference part'*, encouraged the development and repetition of existing computer-generated processes.

Reference, iterator, and repeater parts are all detailed in Chapter 7. Reference parts take the output of one part and feed it into another; iterators produce another iteration of a previous part on the timeline, while repeater parts repeat the music of a given range.

Before understanding the references as a threshold concept (discussed in theme G), users relied upon features such as the ubiquitous copy and paste. Using copy and paste reduced once users had understood references. Participants in the shorter studies remained apprehensive throughout, but those in the longitudinal study eventually found the tools crucial for developing high-level

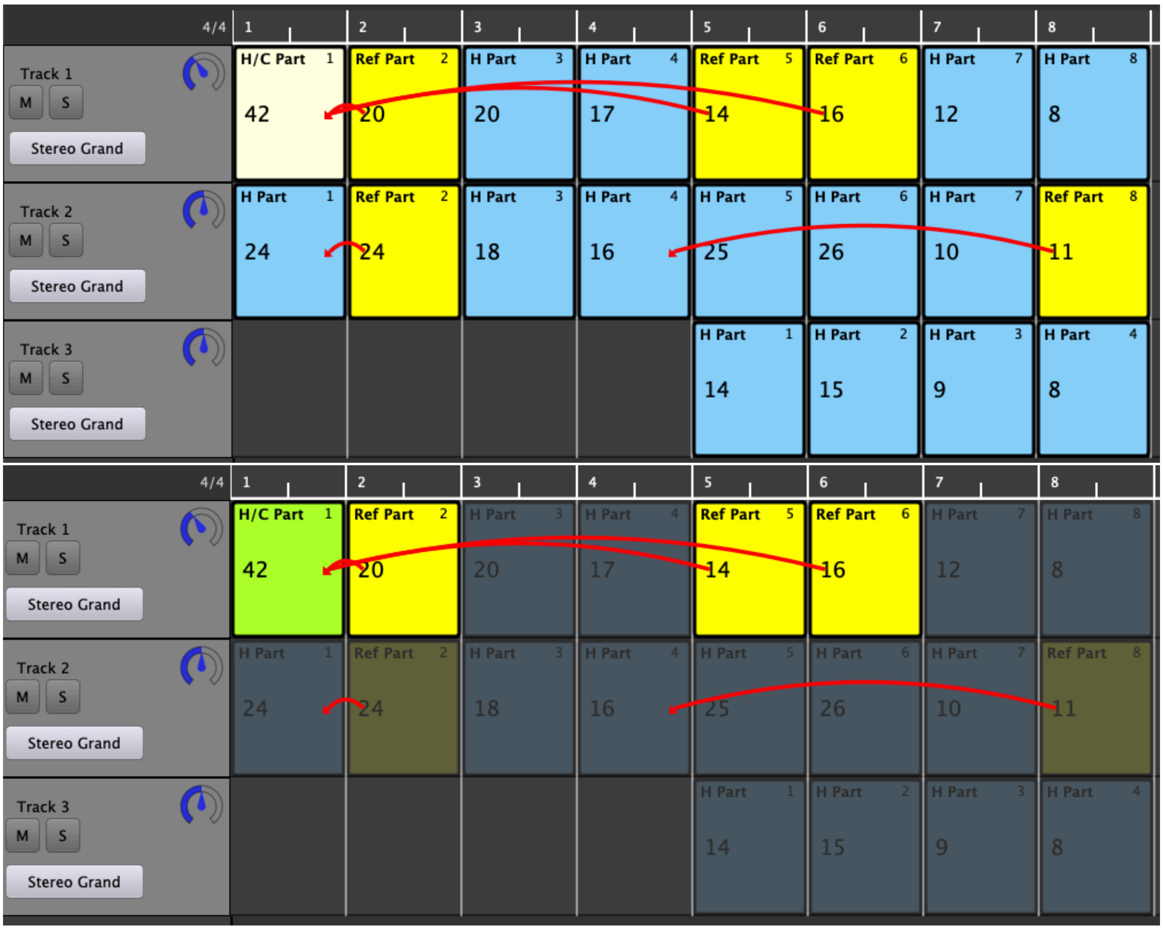


Figure 10.13.: Two methods for checking reference part dependencies in IGME. The top showing all dependencies, the bottom showing all the children attached to a given parent (track 1 part 1).

structure, reinforcing the consistency theme (C) discussed previously. For example, using iterators to produce several variants of the same computer-generated model resulted in each having similar musical properties (e.g. a motif or theme).

Users employed many techniques to achieve this high-level structure (discussed below). Once they had become familiar with the references (threshold concepts), they showed confidence with the techniques, sometimes setting up multiple effects and high-level mechanisms without auditioning them. This highlighted the execution of pre-existing plans (mental mind-map, theme D).

Reference parts were primarily used when a user wanted to develop a previously composed idea, applying edits or changes. Iterator parts were used when the user had set up a desirable generative process but wanted to create several variants across the timeline. Repeaters were used for repeating a large block of material that could contain an arbitrary arrangement of part types.

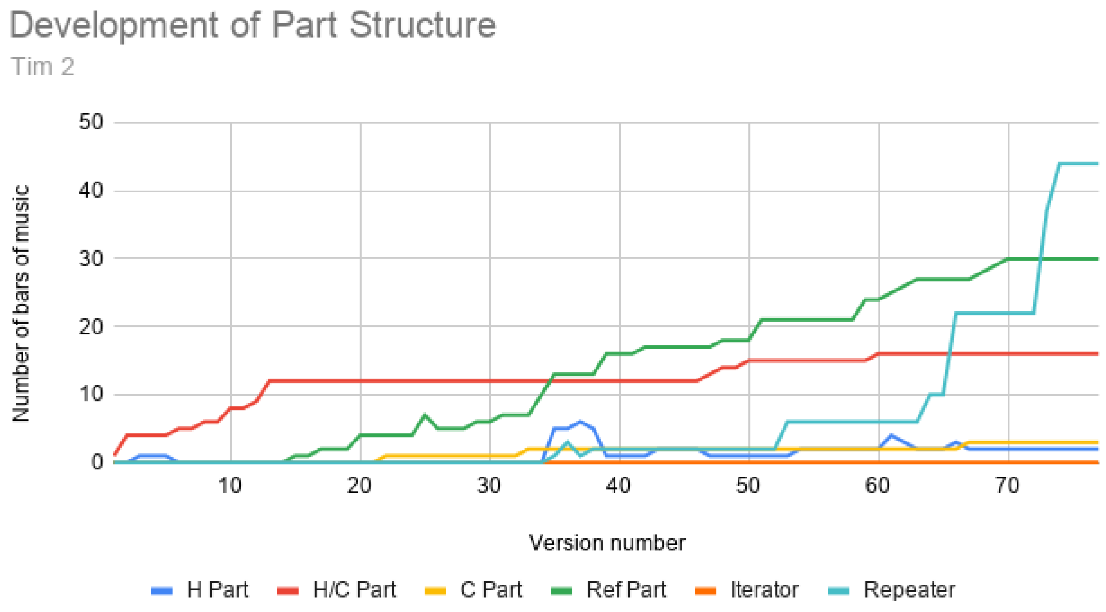


Figure 10.14.: Line graph showing the development of part types in Tim's 2nd composition throughout the piece.

Sometimes these tools were used simply to create symbolic repeats (copy-paste implicitly loses such links and dependencies).

Issues arose when users wanted to repeat or develop musical material. This meant they had to problem-solve and consider which process would allow them to do what they wanted. Sometimes they used a mechanism in an unorthodox way, such as repeating content with an *'iterator'* part. Alternatively, they would use a reference to take the output of one part (expected use case) and convert it back to a human part, this having the same effect as copying the result of one part into the seed of another. This too demonstrated problem-solving (theme D).

Users utilised the automatic dependency-checking options within IGME (Figure 10.13), as well as manually studying the arrange view, to determine structural dependencies. A design feature of reference parts is that updating the parent will automatically propagate changes through to the child parts. Participants in the studies rarely updated the parent. It is unclear whether they were aware of this implicit mechanism and subsequent dependences created by it. Users sometimes converted reference parts into normal parts, explicitly breaking this structure. At times, they seemed to distrust this visualisation and would additionally audition and manually compare two parts to ensure the effect had worked correctly.

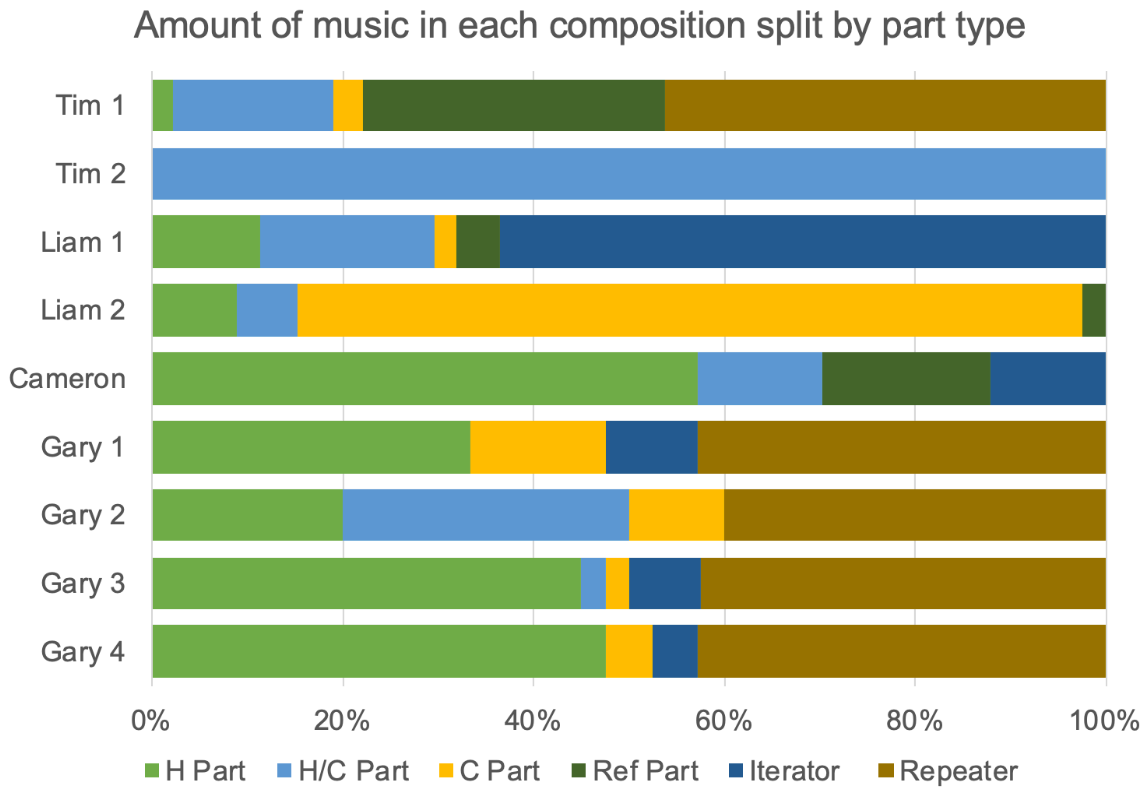


Figure 10.15.: Ratio of part types for each composition.

Although the mechanisms encouraged the development of high-level structure, the explicit breaking down of music into different roles (i.e. parts) placed an additional cognitive load on the user, who had to determine which part was suitable for what.

The development of high-level structure can be visualised through the music logged in the arrange-level VCS system. The distribution of part types can be reconstructed to show how they develop over time. For example, Tim's second composition (Figure 10.14) shows the creation of new material plateaus and instead the user focuses on developing existing sequences of music.

Figure 10.15 shows the distribution of part types in each of the nine compositions created in this study. Each of the nine compositions created exhibit a wide variety in the way musical structure is developed, additional figures for each composition, showing how part types develop over time is given in Appendix E (Figures 1 - 9). The large variance between different users and their compositions suggests that IGME supports *creativity* due to the *non-linear* way of working in IGME (further discussed in Chapter 11).

10.2.7. G: Learnability, Virtuosity, and Threshold Concepts

"Users learnt IGME through studying the tutorials and experimenting autonomously."

This research set out to create an end-user music system similar to other digital music sequencers. Participants' prior experiences of music software enabled users to transfer their skills and knowledge via core components inside IGME. All used computer-generated effects and produced compositions, suggesting they had learnt IGME sufficiently. However, they needed more hand-holding to grasp IGME's more complex features and build a cognitive model of CGM. This particular aspect was overlooked at the outset of the project.

As in other studies, participants in this project primarily learnt IGME through tutorials and independent exploration. The learnability aspect of previous studies was mostly assessed by the researcher, who observed the participants and drew on his own experience of teaching students in a similar workshop-led style. The participants' *'development'*, and the threshold concepts they needed to grasp, was captured more explicitly by the screen recordings.

Threshold Concepts

A threshold concept can be defined as opening a previously unknown subject - in other words, a learner cannot progress in a subject without understanding a specific concept (Meyer and Land, 2003). As highlighted in the video, the reoccurring concepts that users had to understand were:

- The difference between deterministic and stochastic processes.
- The difference between iterate and playback.
- Using reference parts and iterators.
- Plug-ins, seed generators, and note parameters.

The difference between playback, which simply auditions the seed, and iteration, which causes the generative effects to be computed, was one of the first threshold concepts for participants to grasp. Gary stated: *"so in this part, if I iterate it, it won't do anything unless there is a plug-in loaded?"*. Tim lost the progress he had made, as he had not iterated, and tried to reload a previous version from the VCS system. Although frustrated, Tim verbalised his discovery, demonstrating an understanding of this threshold concept.

Differentiating between deterministic and stochastic processes, i.e. knowing at what point an iteration produces a computer-generated output, and what causes this to stop, is another threshold concept. Highlighting notes in different colours helped users to quickly understand which events were fixed (black) and which were stochastic (green). On the other hand, locked parts, non-selection of notes in the editor, and not loading computer-generated processes all caused the termination of output. A null effect could be produced by pure chance. Likewise, sometimes there was no output, for example, because a stochastic process had removed all the notes.

As argued in theme F, reference parts are crucial for building a high-level structure in CGM. However, they often confuse, primarily as they are more complicated than copy-paste and because there are multiple mechanisms for repeating structure. Although these are all fixable usability issues.

Learning by Experimentation

All users learnt IGME through experimentation (trial and error) and often applied the same tenacity to problem-solving (previously discussed). Users often applied a scatter-gun approach to working with processes in which parameters were quickly and arbitrarily set up, allowing users to make something quickly. By contrast, users also carefully constructed a computer-generated process once they understood it in more detail.

In a novel example of this, Tim went through each control in the DS plug-in, mapping between the controls and output. Gary did something similar (also using the DS), adjusting each pitch profile control and octave range to find the notes that would trigger cymbals. After struggling to understand references (a threshold concept), Cameron created a variant of each type to explicitly observe the resultant effect.

Users quickly learned about the relationship between the note parameter editor and subsequent result, moving each control in isolation to observe the effect. Sometimes they verbalised their thought process, confirming they had understood the underlying models. Moreover, they created novel effects to underpin their understanding, which might explain why much of the created computer-generated output could be considered unmusical (discussed shortly). Each computer-generated process has its own miniature threshold concept, but these are not easy to compound into generalised concepts. Users had to build a cognitive understanding of the mapping between input (notes and parameters) and output, this is made more difficult with stochastic processes (discussed more in Chapter 11).

10.3. Musical Artefacts

The music created by users varied significantly. It remains doubtful whether they always took the task seriously, as evidenced by Tim's comment, *"before (in session 1) I was trying to see how weird I could make it and this time I am trying to create something sensible"*.

Creating sound effects is a valid use of CGM, and this was all that one user did. Liam stated: *"the output would have been difficult to compose manually"*. Similarly, Tim experimented with generating drum patterns. His subsequent efforts to do this manually took significantly more time, illustrating how generative music can sometimes increase productivity, even if the output is not always as *'pleasing'*. Although, from results in theme B, users generated more iterations when working with CGM than without, resulting in increased ideation.

As with existing research in generative music, the music output of the study attracted criticism and sometimes induced a sense of frustration. Although users seemed to enjoy their overall experiences with the software, as evidenced by them laughing, smiling, and by their comments, they all expressed disdain at some point, with some explicitly stating *"that they did not like what they had made"*. More light-hearted observations included *"this sounds like two aliens speaking to each other"*. Even when praising the output, users said it often did not fit into the song in which they were working. Despite not being completely satisfied with what they had produced, users accepted the *'not-so-bad'* iteration and moved on with the rest of the composition.

Sometimes users blamed themselves (usually out loud) for a bad iteration. After trying and giving up on several iterations, Gary replaced the generative part with a human-created one, stating *"my issue is now I know exactly what I want to happen, so I will just do that"*, additionally stating *"It is so hard to work with generative parts when I am so used to not working with them"*. As discussed in theme A, Gary did eventually find a novel use for IGME's processes.

10.3.1. Sound Editing

IGME's large bank of sounds resulted in users spending a long time experimenting with instruments. This was not the focus of the study, so some of these instruments could have been removed. The researcher overlooked this, leaving users free to load sound effects such as the comical MIDI *jetplane* effect. All users spent time configuring instruments, even though this is already an intrinsic part of digital music composition.

10.4. Survey Comparisons

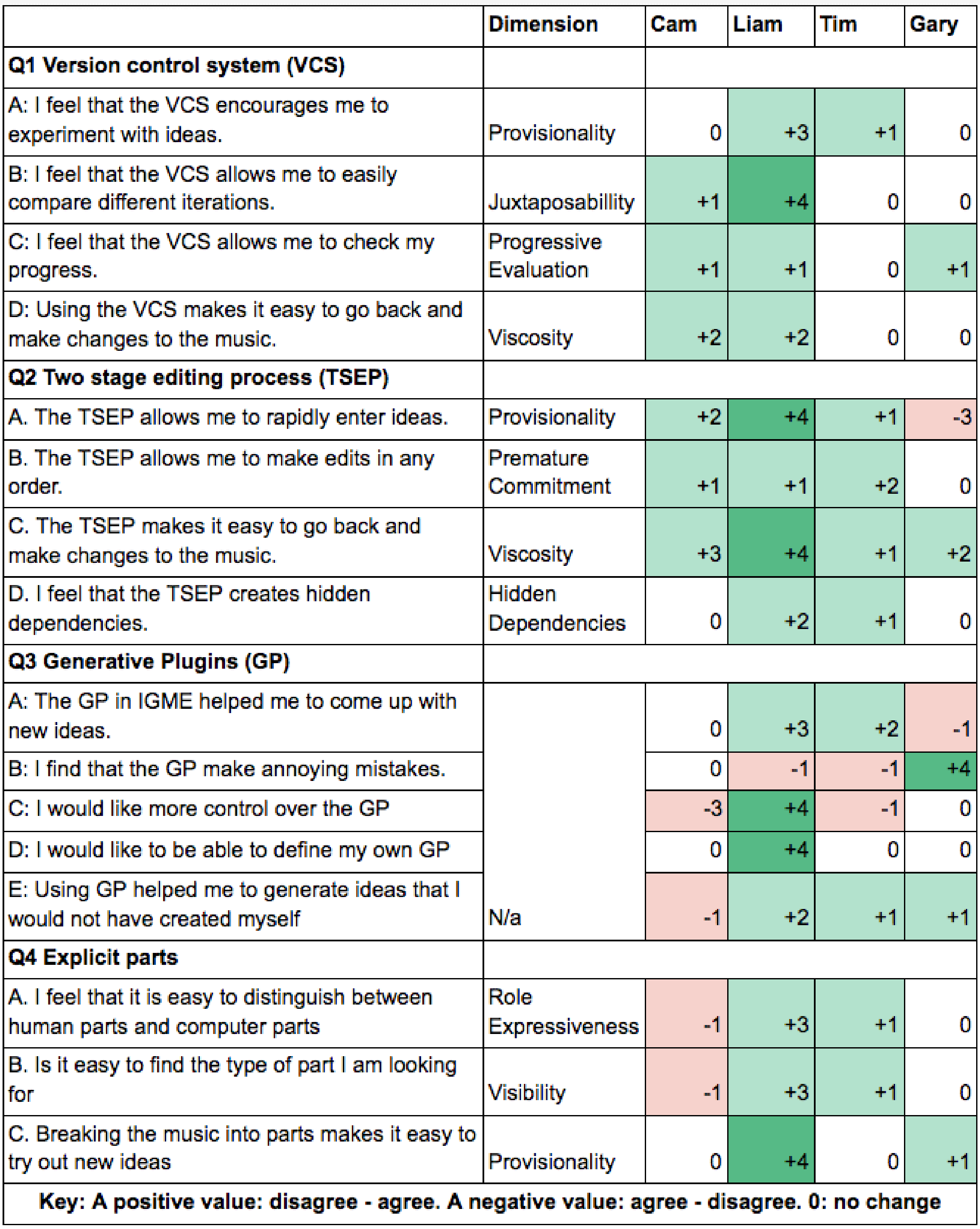
Survey B has been analysed in depth previously (Chapter 9), so the focus here is on whether users' responses after the first 2-hour session changed once they had completed the second 2-hour session. The full responses for both sessions are given in appendix E (Figures 16-19). The small sample size limited the choice of valid statistical tests, so the numerical difference between scores for each question was computed, this giving a coarse comparison of the results. A positive value indicated users moved from *'strongly disagree'* to *'strongly agree'*, and vice-versa. A score of +4 or -4 signified a flip of opinion from one extreme to the other.

The results in Figure 10.16 and 10.17 showed users' perceptions of the VCS and TSEP generally improved. Two users noted that the TSEP increased *hidden dependencies* and that this trade-off improved *provisionality*, reduced *premature commitment* and *viscosity*.

Question 3 elicited more varied responses. Gary, who was the least engaged with generative music, strongly agreed they made annoying mistakes (3B) whereas the responses of Liam and Tim became more positive as time lapsed. This suggested the more they learnt the processes, the more they were able to control them. Question 3C produced polarising responses, with Liam strongly agreeing he wanted more control and also to be able to define his own processes (3D). Answering question 4, Liam agreed with splitting music into parts. Other users' opinions did not do change significantly.

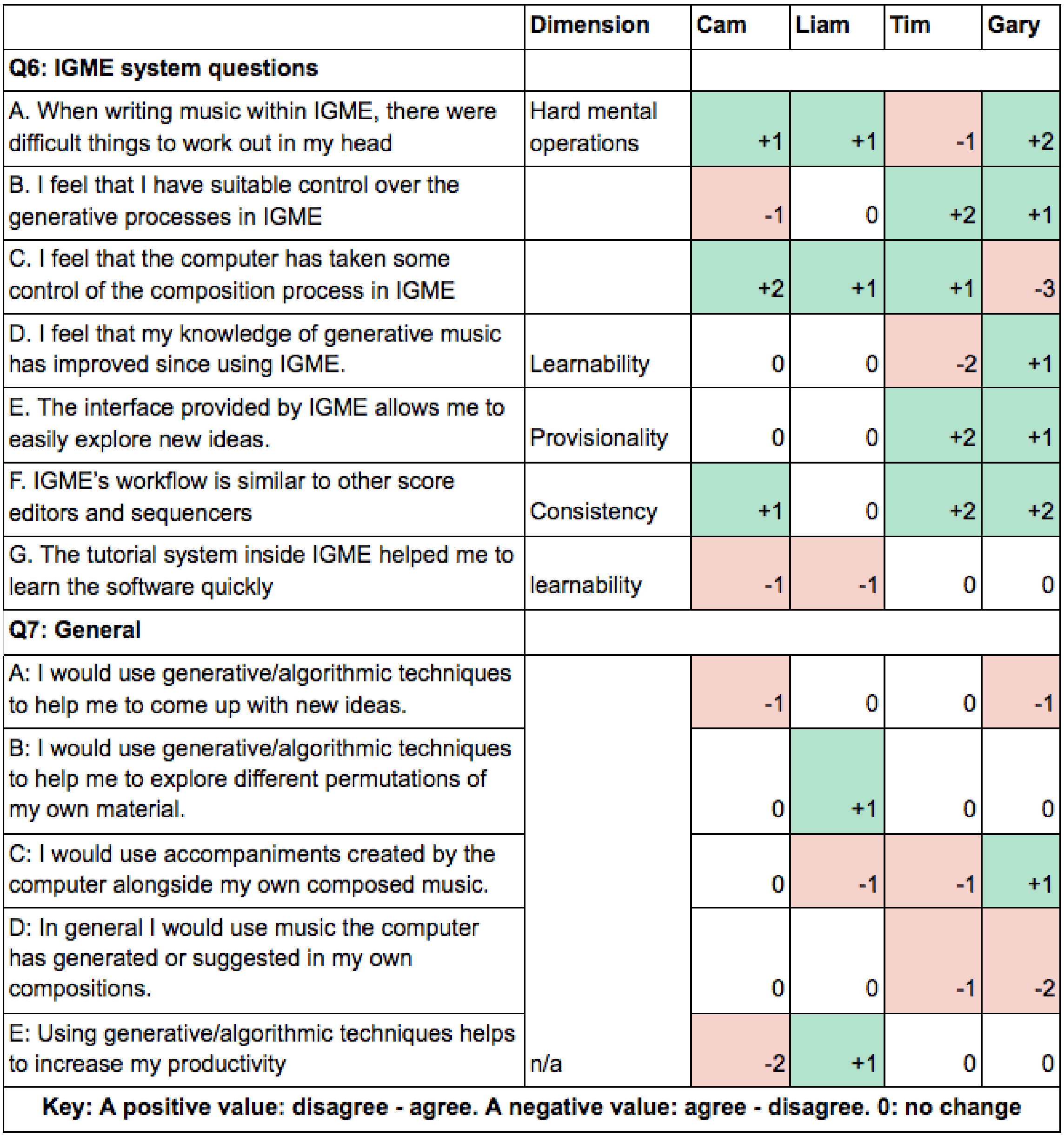
The range of results and anomalies for questions 6 and 7 make it difficult to draw definitive conclusions. In summary, participants agreed that *hard mental operations* had increased. They generally concurred that they had suitable control over IGME (6B), but also said the computer had taken away some control (6C). As time lapsed, they agreed that IGME's overall workflow was similar to other sequencers, confirming that one of the fundamental design requirements had been met[12]. Users were slightly less likely to use CGM to generate initial ideas (7A), but remained happy to use it to develop their own ideas (7B). Two participants signalled they were now less likely to use CGM in their own composition (7D). The remaining survey questions produced little to no variance in responses.

[12] IGME could easily be used by someone with familiar experiences of other digital music sequencing software, which IGME was designed to imitate.



	Dimension	Cam	Liam	Tim	Gary
Q1 Version control system (VCS)					
A: I feel that the VCS encourages me to experiment with ideas.	Provisionality	0	+3	+1	0
B: I feel that the VCS allows me to easily compare different iterations.	Juxtaposabillity	+1	+4	0	0
C: I feel that the VCS allows me to check my progress.	Progressive Evaluation	+1	+1	0	+1
D: Using the VCS makes it easy to go back and make changes to the music.	Viscosity	+2	+2	0	0
Q2 Two stage editing process (TSEP)					
A. The TSEP allows me to rapidly enter ideas.	Provisionality	+2	+4	+1	-3
B. The TSEP allows me to make edits in any order.	Premature Commitment	+1	+1	+2	0
C. The TSEP makes it easy to go back and make changes to the music.	Viscosity	+3	+4	+1	+2
D. I feel that the TSEP creates hidden dependencies.	Hidden Dependencies	0	+2	+1	0
Q3 Generative Plugins (GP)					
A: The GP in IGME helped me to come up with new ideas.	N/a	0	+3	+2	-1
B: I find that the GP make annoying mistakes.		0	-1	-1	+4
C: I would like more control over the GP		-3	+4	-1	0
D: I would like to be able to define my own GP		0	+4	0	0
E: Using GP helped me to generate ideas that I would not have created myself		-1	+2	+1	+1
Q4 Explicit parts					
A. I feel that it is easy to distinguish between human parts and computer parts	Role Expressiveness	-1	+3	+1	0
B. Is it easy to find the type of part I am looking for	Visibility	-1	+3	+1	0
C. Breaking the music into parts makes it easy to try out new ideas	Provisionality	0	+4	0	+1
Key: A positive value: disagree - agree. A negative value: agree - disagree. 0: no change					

Figure 10.16.: Survey B results indicating the difference in received survey response recorded between each sessions for all participants (questions 1 - 4).



	Dimension	Cam	Liam	Tim	Gary
Q6: IGME system questions					
A. When writing music within IGME, there were difficult things to work out in my head	Hard mental operations	+1	+1	-1	+2
B. I feel that I have suitable control over the generative processes in IGME		-1	0	+2	+1
C. I feel that the computer has taken some control of the composition process in IGME		+2	+1	+1	-3
D. I feel that my knowledge of generative music has improved since using IGME.	Learnability	0	0	-2	+1
E. The interface provided by IGME allows me to easily explore new ideas.	Provisionality	0	0	+2	+1
F. IGME's workflow is similar to other score editors and sequencers	Consistency	+1	0	+2	+2
G. The tutorial system inside IGME helped me to learn the software quickly	learnability	-1	-1	0	0
Q7: General					
A: I would use generative/algorithmic techniques to help me to come up with new ideas.	n/a	-1	0	0	-1
B: I would use generative/algorithmic techniques to help me to explore different permutations of my own material.		0	+1	0	0
C: I would use accompaniments created by the computer alongside my own composed music.		0	-1	-1	+1
D: In general I would use music the computer has generated or suggested in my own compositions.		0	0	-1	-2
E: Using generative/algorithmic techniques helps to increase my productivity		-2	+1	0	0
Key: A positive value: disagree - agree. A negative value: agree - disagree. 0: no change					

Figure 10.17.: Survey B results indicating the difference in received survey response recorded between each sessions for all participants (questions 6 - 7).

10.5. Methodology Review and Threats to Validity

The study permitted analysing interaction in greater detail than previous studies. The anomalies found previously within interaction data, could now be explained through the recorded video, providing context and resolving ambiguous findings. The video also showed interaction not captured by IGME's data logger, such as browsing tutorials that permitted discovering content associated with theme G (learnability) or understanding how the VCS was used (as no interaction data was captured for this). Generally, the discovery of themes guided the researcher for areas in which to *probe* the interaction data.

The study originally proposed that 10 participants would be studied, however as discussed in Chapter 8, the remaining 6 had to be cancelled. The small sample size would normally raise questions as to validity, repeatability and generalisability. However, they are extensions of those already discovered in studies involving larger numbers of participants. Furthermore, the small sample size permitted the researcher to spend time analysing the results in microscopic detail - more finely reviewing and coding the themes discussed. It is unclear within the scope of the research if it would have been feasible to complete more studies.

Undoubtedly, the invasive nature of participants being observed did not create a perfect real-world situation. A notable downside of the study was that users' questions often required a verbal explanation. The researcher aimed to be objective when answering questions based on content in the tutorials, as some problems could be solved in multiple ways. Although trying as hard as possible to be a passive observer, the researcher had to step in when IGME crashed and needed to be fixed. Despite all the testing, several feature-crippling bugs developed. Realistically, users could not have fixed these, or found workarounds, by themselves, so either the researcher had to explain what needed to be done or had to apply a manual fix. It is unclear what users would have done, or how they would have solved such problems if the researcher had not been present. On the other hand, having to sort out issues for themselves could have accelerated users' comprehension of IGME and its threshold concepts.

10.6. Conclusion

In summary, when working with generative music (theme A), a series of edit, iteration, evaluating (EIE) cycles were used to produce many versions of generative output (theme B). As time progressed, users moved from working rapidly to applying more detailed or fine-grained edits. They were quick to learn and apply workflows for CGM. A set of preferred plug-ins were mastered, facilitating musical consistency (theme C). Users of CGM were problem-solvers, debugging and fixing issues with computer-generated output using existing knowledge of music theory, and assembling processes to realise an envisioned compositional structure (theme D).

Version control (theme E) was mostly used as a safety net, encouraging users to ideate musical ideas. Recalling and comparing variants remained uncommon. Several tools for developing high-level structure were used, making use of IGME's inbuilt reference parts, emphasising repeating and developing structure (theme F).

IGME was learnt through a combination of existing knowledge of music sequencing, studying tutorials and experimenting with the software, requiring users to master several threshold concepts (theme G).

Although these themes were documented in this chapter, they are also echoed within the smaller studies, and many of them, despite not being formalised, were evident in previous studies.

Part V.

CONCLUSION

11. Conclusion

The thesis aimed to observe, through the IGME platform, the interaction between end-users and computer-generated music (CGM) composition systems. A range of methodologies, supported by an iterative research and development process, was employed. The methodologies included initial informal pilot studies, formal studies in both controlled and uncontrolled conditions, and a longitudinal study. This chapter reviews those methodologies and summarises the empirical findings of the user studies. It also presents a set of design heuristics for CGM software

11.1. Summary of Thesis

This thesis was informed by creativity and HCI. Both were explored in Chapter 2, while the introduction and categorisation of techniques for computer-generated music, and their primary evaluation as tools for end-users, were covered in Chapter 3. This theme continued in Chapter 4, which took a broad look at existing systems for CGM, attempting to find a *'middle ground'* for building an end-user CGM system.

Chapter 5 set out the design requirements for a CGM system, amalgamating primary and secondary research and developing formal models for supporting interaction. Chapter 6 analysed existing systems and interfaces using the CDN framework. This created a formal set of design hypotheses that helped to formalise requirements for building end-user CGM systems. These requirements evolved through the development of IGME, which was built for studying user interaction and discussed in Chapter 7.

Chapter 8 covered the methodologies for studying interaction and for improving IGME through informal pilot studies, resulting in a design, test, and evaluation cycle. Chapter 9 summarised and compared the results of two studies involving IGME, one in controlled conditions and one in uncontrolled conditions. The findings were extended throughout Chapter 10, which analysed

A: What tools do music practitioners require for working with computer-generated music?	Chapters
• The interfaces and tools provided should be similar to those found in common digital music sequencing software.	4
• Interfaces that abstract the complexity of CGM techniques democratise assess to them.	5, 6, 7
• A range of design heuristics are recommended for CGM systems (discussed further in this chapter).	6, 9, 10
B: How do these tools differ from existing systems and interfaces?	
• Existing systems are either too dissimilar to common music software interfaces, or require the appropriation of programming environments.	4, 6
• Music practitioners often have limited programming skills, and use the computer as one of many tools in composition (alongside instruments, and other forms of notion)	5
C: How can computer-generated music processes be appropriated in a co-collaborative creative process?	
• CGM is mostly used to influence and catalyse the composition processes.	9, 10
• CGM techniques are preferable at the micro rather than macro level of music structure.	9, 10

Table 11.1.: Summary of research questions.

the screen recordings of a longitudinal study, drawing out the themes associated with CGM composition.

11.1.1. Research Questions

The original research questions proposed in the introduction were:

- What tools do music practitioners require for working with computer-generated music?
- How do these tools differ from existing systems and interfaces?
- How can computer-generated music processes be appropriated in a co-collaborative creative process?

Table 11.1 shows each question and states in which chapters they were explored and answered. The proceeding section further explores these questions, which in turn leads to the contributions to knowledge.

11.2. Contributions to Knowledge

The principal themes of the research can be grouped as:

A : A set of design heuristics for building end-user computer-generated music systems.

B : How computer-generated music techniques are used in real-world practice.

C : The users' profile and backgrounds conducive to computer-generated music.

D : The role of the computer in co-collaborative music practice.

The findings are summarised in Table 11.2, and the above labels (A-D) are used to categorise the contributions. The remainder of this section further unpacks these findings.

11.2.1. Empirical Findings

Studying real-world interaction with computer-generated compositions revealed how the tools designed in this research were adopted by practitioners. Initial findings from the short studies (Chapter 9) were refined, clarified, and concluded in the longitudinal study (Chapter 10). The research demonstrated that participants could easily engage with CGM composition.

Computer-Generated Techniques

IGME users preferred to master a small set of computer-generated processes rather than experiment with all processes. The actual computer-generated technique underpinning each process became less important than how the technique was presented through a graphical interface. Users had to be able to connect action to mapping and debug what is fundamentally a *'black-box'*. Whereas some musical instruments have a direct mapping between an action and resultant sound, i.e. a piano (Maes *et al.*, 2014), most of IGME's processes are stochastic, and it is not immediately obvious how input and parameters are combined to produce output. All users appeared to enjoy the discovery process. Finding the limits became a creative endeavour, permitting the tool to be used in real-world challenges.

Theme	Summary of contribution	Page
Chapter 4		
A	• Existing commercial music software has limited capability to engage with computer-generated processes.	61
C	• Purpose built generative music systems require engagement with irregular interfaces and workflows.	61
Chapter 5		
C	• Music practitioners have limited exposure to, or knowledge of CGM techniques.	69
Chapter 9		
A	• The version control system is a peripheral interaction tool that supported ideation, and was occasionally used to recall previous versions.	135
A	• The two-stage editing process supported rapid note entry, again supporting ideation.	135
B	• Computer-generated processes helped users come up with ideas they would not have come up with on their own.	136
D	• Users said they would have liked a little more control and to be able to define their own processes within IGME, indicating the computer had taken away some control.	136
A	• Users agreed that IGME was similar to mainstream music software and that this had helped them to learn it and also to learn more about generative music.	137
B	• Users were happy to use computer-generated techniques to come up with ideas, and permute existing ones, as well as using the techniques in their own music.	138
D	• Participants remained apprehensive about the extent to which the computer was involved, with just under half expressing concern over authorship.	139
D	• Participants accepted that the computer was responsible for some of their musical output. None attributed all their musical output to the computer.	139
A	• On average, users took less than 20 minutes to produce their first CGM.	142
C	• Similar results were obtained under both controlled and uncontrolled conditions.	148
Chapter 10		
	Confirmed many findings from the earlier studies but in addition:	
B	• Users became less likely to use computer-generated processes for creating music from scratch but were more likely to use them for modifying, extending, and recomposing existing music.	156
A	• Rapid edit-iteration-evaluating cycles were commonly used.	159
B	• Users preferred to master a small number of computer-generated processes.	166
C	• Users needed some knowledge of music theory to work with generative output.	169
A	• Tools for developing high-level structure were essential.	172
C	• Several threshold concepts were evident and required for meaningful interaction.	176

Table 11.2.: Summary of findings.

Techniques that could transform, reinterpret and modify user-composed sequences were preferable to those designed to replace initial ideation. The CGM field focuses mainly on generating music from scratch. Little research has been done on music practitioners' preference for using computer-generated techniques to manipulate, extend and reinterpret music composed by themselves. Therefore, future research should focus on this.

With most techniques inside IGME outputting swiftly, future work should focus on ensuring that any computer-generated process supports rapid ideation. CGM relies heavily on *'chance'*, which means the search space has to be navigated repeatedly for satisfying musical output. The version control system (VCS) and two-stage edit process (TSEP) ensured participants in the studies could work quickly and that few restrictions were enforced by the user interface, this ensuring a state of flow could be obtained.

The chance of finding suitable music in stochastic processes diminishes when model complexity and the range of parameters is increased. As mentioned in Chapter 10, generative processes often had to be used alongside other processes that could limit or constrain the output. Reflecting on their experiences of IGME, one participant stated that, to use IGME *'properly'*, more constraints would be needed, emphasising that at times the system might have been too *'random'*. That individual said, *"it feels like I am throwing paint rather than applying it and I think I would rather paint with a brush than chuck paint at it"*.

Several study participants used the transition table plug-in (a first-order Markov model), proving that complex techniques for CGM can be appropriated by regular digital music practitioners without pre-existing knowledge of the technique. This specific process supports populating the initial parameters through the analysis of existing music, while still permitting the user to freely edit the controls. This not only allows users to get started quickly but eventually gives them the confidence to define their own models. Therefore, it is not the techniques that act as barriers to entry for CGM, but the interfaces designed around them.

High-Level Structure

Subjects preferred to generate small fragments of music and subsequently arrange them to form high-level structure rather than generate larger phrases. Working with smaller pieces of music ensured users were more in control and the local structure was less likely to spiral into chaos. Existing research in the field has had more success with creating small local structures (i.e. a bar), rather than high-level structures, although the gap as suggested by recent literature (Siphocly, Salem, and El-Horabty, 2021) is narrowing. However, a lack of high-level structure remains a commonly criticised characteristic of CGM. This partially confirms that simpler, easy to understand and implement processes are preferable to larger, more complex black-box algorithms, as speculated by Sorensen and Brown (2008). In addition, this research has shown that users prefer to work with CGM techniques that have a narrow *'breadth of context'* (Wooller *et al.*, 2005). Although this thesis did not explore many techniques that have a large *'breath of context'* it also may suggest that these may not have been preferable for the music practitioners studied here, which may or may not demonstrate a wider trend. Indeed, in reviewing music software more generally, Nash (2011) notated a strong correlation between flow (intrinsic motivation, enjoyment) and programs that offer smaller, more focused edit-audition workflows (patterns, loops), versus those designed around wider musical arrangements (linear timelines).

Working With Computer-Generated Music

When working with CGM, the computer's role might be that of a *'visionary'* and creator of a set of musical ideas to inspire the human, whose role remains that of *composer* (or curator), deciding which musical fragments can be best used and in what order they should be sequenced.

Users were mostly positive about CGM and welcomed the idea of using it, suggesting that the low widespread adoption was probably due to a lack of appropriate interfaces and workflows in existing systems. Another factor is that composers who are unaware of computer-generated tools will not seek them out. As with all commercial software, the need for additional features often correlates to a return on investment. If users are not requesting computer-generated features, and such features will not increase the sales of said software, then they remain assigned to the enthusiast and academic domains.

The choice of computer-generated processes, the development of high-level structure, and the variance of interaction data (including notable outliers) indicate that users did not work with IGME in a linear way, with each demonstrating *originality*. This observation and others already discussed, suggest that IGME elicits creativity, and supports numerous ways *('wide walls')* of composing and working with computer-generated processes (Resnick *et al.*, 2005). In their definition of creativity, Acar, Burnett, and Cabra (2017) note four ingredients: originality, value, surprise, and aesthetics, with *originality* being the strongest correlation to *creativity*.

Authorship questions arose throughout the studies but were not fully addressed. As seen in Chapter 9, one participant noted that *'AI'* was seen as a threat by *"people that usually know nothing about it"*. It could be argued that *AI* in music is misunderstood and that there is a need to emphasise to composers that music generated by an intelligent process is not a threat but a means of increasing creativity and efficiency. Moreover, computer-generated software users are not just composers of notes but also composers of parameters and constraints. Authorship can still be expressed through the arrangement of initial music material, selection of parameters and plug-ins and, finally, the curation of material from different output iterations. A degree of creative flair is needed when configuring computer-generated processes.

In the shorter studies, CGM users were apprehensive about who the author was. Participants said they would use CGM to inspire and motivate them to create initial music ideas, but they remained apprehensive about directly using the output of such music in their own compositions. However, survey responses discussed in Chapters 10 and 11 show that no one attributed music composed with IGME solely to the *computer*. This is probably because users remained in control of this division (between computer and human-created music).

It is clear that longitudinal studies cannot be replaced by running repeated shorter studies. For example, at first glance, it appeared that participants had not used the tools offered by IGME in a serious or measured way (in the short studies). However, it became apparent that this observation was incorrect and that participants had initially created extreme pieces of music in order to gain a cognitive understanding of the various processes, learning IGME through experimentation. If not for the screen recordings in the longitudinal study, such a finding might not have been made.

Using CGM to produce *'unmusical'* output or odd sound effects, is not deemed a negative characteristic but an observation. One user, specifically in Chapter 10, noted that these processes resulted in sound effects that would have been difficult to create manually. Similarly, trying to limit music composition to a task measured in terms of its productivity is reductionist and meaningless.

Music, as with other creative endeavours, needs intrinsic motivation. Not all music practitioners want to use computer-generated tools, just as some classical composers have no wish to write music for an electric banjo. For the field to advance, it must be acknowledged that CGM is not a tool sought out by every practitioner. Furthermore, should such computer-generated techniques become a more prominent feature of common music sequencing software, they should always remain optional, just as they are in the limited examples already discussed (Chapter 4).

A Computer-Generated Music Practitioner

The individuals[1] recruited for the studies had different levels of experience in music practice. All enjoyed their time with IGME but many remained apprehensive about adopting computer-generated techniques in their daily practice. Although some characteristics of the ideal CGM practitioner remain elusive, some common themes were evident. Notably, an individual needs enough knowledge of music theory (either explicitly or implicitly) to debug music that does not conform to inherent properties of tonal music - recognising *'tonal mistakes'*. Of course, creating atonal music remains a valid compositional activity. Users also need existing knowledge of music sequencing workflows.

11.2.2. Design Heuristics for Computer-Generated Music Systems

Limited advances have been made in integrating CGM techniques in existing music sequencing software and workflows. IGME was built as a means for exploring what such a system might look like. In evaluating the features of IGME through real-world testing, the following section discusses desirable characteristics for end-user CGM systems. Fundamentally, there should be a clear divide between systems that seek to replace or mimic human composition and those that aim to work alongside human composition, with a different set of requirements for each. This research looked to support human composition.

Two sets of design heuristics are given in this section. The first (Table 11.3) shows a summary of specific design elements evaluated through the user studies. The second set (Table 11.4) articulates further general design principles using the language of the cognitive dimensions of notations framework with lessons from both literature and the empirical work presented here. Where possible each heuristics provides a page number where the strongest evidence for its rationale was discussed. It is important to note that dimensions are not orthogonal, and have trade-offs, requiring that designers identify which dimensions are priorities for their intended user experience (Nash, 2012). Neither list is exhaustive and omits more general usability requirements already well documented, focusing on priorities specifically for CGM systems.

[1] The demographic for this study was primarily undergraduate music technology students. This limitation is discussed further in section 11.3.

Design requirement	Rationale	Evidenced by
Enable rapid edit-iteration-evaluation (EIE) cycles.	Rapid EIE cycles are integral to computer-generated processes.	Frequently observed in user studies (page 159), and used as a means to explore the limits of CGM processes, which often require many iterations for a result.
Contain a version control system.	Required for working with stochastic processes in which previous iterations are easily lost or overwritten, acting as a passive safety net.	The VCS was welcomed by participants (page 135), and primarily acted to support the above requirement as a (passive) peripheral interaction tool.
Built on familiar digital music composition design paradigms.	Aids with learnability and increases the likelihood of user adoption.	Chapter 9 & 10 revealed that users found IGME easy to learn and offered similar workflows to existing systems (page 137)
Contain minimal threshold concepts.	Decreases the barriers to entry and places focus on using rather than learning computer-generated processes.	Several threshold concepts were identified in Chapter 10 (176) which users needed to understand to create meaningful interaction with CGM.
Prioritise developing smaller well-contained measures of music.	Small measures are easier to create and less likely to spiral into chaos.	Users primarily created small fragments of music (page 156), focusing on narrow *'breadths of context'*.
Focus on small, easy to understand, processes that can be combined and sequenced together through formal mechanisms.	Easily allows the computer-generated output to be sequenced in and around human-composed content and permits high-level structure.	User preferred to master simple CGM processes and had limited engagement with more complex processes (page 166). The part referencing mechanism permitted the development of high-level structure (page 172)

Table 11.3.: Design heuristics for CGM systems.

Feature Summary

The VCS was primarily designed to make working with CGM practical. Given the inherently stochastic nature of such music, the ability to store and recall previous iterations was essential. Participants agreed that VCS had merits, although, since *'undo'* was never properly implemented inside IGME, they could have been overly reliant on it. Nevertheless, we strongly argue that such a feature is crucial for CGM systems, even though it could increase both ideation and rapid edit-audition cycles in existing music software. This research has advanced the topics discussed by Duignan (2010), who proposed the use of such systems in music compositions.

The two-stage editing process (initial input and resultant output) democratises access to CGM. Although unorthodox, it represents a novel attempt to solve the problem of creating an interface that enables access to CGM while retaining similarities to common music sequencing workflows. This feature was positively received by most study participants. The input editor and resultant output would also benefit from being decoupled in future systems, so the mappings created by a computer-generated effect can be discovered with the input and subsequent output of a computer-generated process shown explicitly.

We argue that a rapid edit-audition cycle should be integral to computer-generated processes, emphasising speed over other characteristics. If a more complex process can generate suitable music in fewer iterations, this is also desirable. Moreover, action sound latency (the delay between an action and sound) in digital music systems should always be minimal (McPherson, Jack, Moro, *et al.*, 2016). Almost all of IGME's computer-generated processes are instantaneous, but it remains unclear whether more complex processes could be equally instant.

Users commonly employed two of the more complex plug-ins: the distribution sampler, and the transition table. Both took over some of the hard work (as automation devices), allowing users to pre-populate parameters while still giving them a chance to fine-tune the model and at all times allowing them uncompromising control.

The part referencing system in IGME was primarily designed to alleviate issues associated with a lack of high-level structure, a common criticism of generative music already well documented. In splitting composition between human and computer, the human was in most cases left in charge of creating high-level structure, which is difficult for a computer to automate. These mechanisms could also be used to form non-generative structure in ordinary music composition software.

For a generative music system to be attractive, practitioners need to understand that music generated by an AI process need not be a threat but a means of increasing human creativity and efficiency. IGME does not explicitly do this, which could be a topic of future research.

Cognitive Dimensions of Notation Review

Chapter 6 proposed a set of design hypotheses (see Table 6.2) to use as guidelines for building the IGME software. These were aligned with each dimension in the Cognitive Dimensions of Notations framework. In summary, IGME mostly followed the design hypotheses originally listed. However, upon reviewing these, several dimensions needed to be redefined or amended as follows:

Hidden Dependencies

Although IGME used the part referencing mechanism to show hidden dependencies explicitly, it also encouraged users to create more of them. Additionally, this feature had several unforeseen side effects such as deleting a parent reference, leaving all the children orphaned (missing input music). This part referencing mechanism resulted in a trade-off between creating more dependencies and increasing high-level structure.

Consistency

The reliance on design metaphors in existing music software, led users to believe more features were supported (i.e. shortcuts). Little could be done to address this within the scope of the research, but arguably this would have made IGME both slightly harder to learn and work with efficiently.

Premature Commitment

If users wanted to make use of reference parts, they first had to determine which part would be the parent. This led to cases of premature commitment.

Error Proneness

Users were keen to explore CGM but were often left disappointed with low-quality music, suggesting that computer-generated models could have done more to produce higher-quality music, and remove 'obvious' mistakes.

Abstraction Management

Many users wanted to create their own effects, but it is unclear how this could have been supported. Furthermore, such limitations often encourage creative solutions (Chapter 2).

The table in Chapter 6 has been updated following evaluation of the research, with a final set of design heuristics shown in Table 11.4.

Despite limitations, the CDN framework helped design and critique the IGME software and in providing a vocabulary for comparing interfaces for CGM. However, trying to design software that rectifies an established issue within a field (in this case CGM having minimal high-level structure) leads to a CDN profile that suggests, for example, that the part referencing mechanism creates high amounts of hidden dependencies. Therefore, the CDN should not be the only mechanism for specifying design hypotheses but should be considered as guidelines rather than as a set of immutable design requirements.

Dimension	Design Heuristics	Page(s)
1. Visibility	Minimal layers of hierarchical views. Ensure that the role of each musical element is clearly defined visually. Elements of the music that the computer will compose need to be explicitly revealed.	176, 101
2. Juxtaposabillity	Allow different variants of the music to be swapped quickly and easily. Version control technologies can be used to support this. Ensure the timeline can grow and shrink as needed.	156, 171
3. Hard Mental Operations	Focus on using, rather than designing, computer-generated processes. Use a timeline to make control flow simple to predict.	156, 4.5
4. Progressive Evaluation	Support rapid edit-audition cycles and ensure music can be auditioned in any order.	159
5. Hidden Dependencies	Minimize hidden dependencies, and ensure that any required dependencies can be easily visualized. Consider how users inevitably create more dependencies.	172
6. Conciseness	Give controls verbose names. Consider what information a user will need at the surface level.	89
7. Provisionality	Ensure that the system can rapidly ideate. CGM should be computable in minimal time and not prevent rapid edit-iteration cycles. Consider adding a secondary timeline (scratchpad).	159, 171
8. Secondary Notation	Allow for informal notes to be added to various musical components (for example, track description).	170
9. Consistency	Use common design and interaction paradigms found in other music software (for example, clips, tracks, and timelines). Ensure that users can replicate the same (expected) functionality (i.e. short cuts).	137, 166
10. Viscosity	Allow users to lock down the position or final output of the music. Ensure that it is otherwise easy to go back and make changes. Consider how to limit the length of stochastic processes.	159, 166
11. Role Expressiveness	Make the role between computer-generated and human composed music distinguished.	176
12. Premature Commitment	Use an arrange view metaphor, and support part-by-part, bar-by-bar or top down arrangements. Consider how tools for supporting high-level structure may cause unforeseen premature commitment.	172
13. Error Proneness	The computer should not automatically fix musical errors, but instead, allow users to judge and fix these for themselves. Give the user the option of having obvious errors be fixed automatically.	178
14. Closeness of Mapping	Focus on graphical rather that textual controls.	61, 166
15. Abstraction Management	Permit the software to be used without creating abstractions, but allow advanced users to develop their own abstractions. Creating abstractions is likely a *'power-user'* feature.	136

Table 11.4.: A list of design heuristics for each dimension in the CDN that can be used as guidelines for building end-user CGM systems.

11.2.3. Methodology Review

A mixed-methods approach was used to evaluate interaction. The challenge of recruiting representative users caused delays at several stages of the research. A balance of broad, larger sample studies and longitudinal small sample studies were used to provide insight into real-world creative applications, although engagement within the scope of the research was limited. Directions for further development and further empirical studies are supported by the work presented here. The methodologies and study types for investigating user interaction inside IGME produced varying levels of success, and are discussed by type below.

Pilot Studies

Developing IGME to be representative of a complete end-user experience required an extensive development period. In hindsight, engaging users at the outset of the research would have been beneficial, although the pilot studies were invaluable for evaluating and shaping early IGME prototypes. Increasing the number of studies and decreasing the number of users in each could have been beneficial, as it would have resulted in more development-evaluation cycles.

Screen-Recording

The longitudinal study offered the best insight into a computer-generated composition workflow. The ability to re-watch the exact timeline of events executed by the user was invaluable. The longitudinal study took the most time to analyse and draw conclusions from. The study was also limited by sample size, with resource and time constraints preventing more studies from being conducted.

The researcher's presence in the study allowed for further notes to be made and for ad-hoc questions from participants to be answered, but at the risk of influencing subject responses. The information conveyed often helped to clarify novel actions executed by participants, although it remains unclear what effect this had on the study. Potentially, the longitudinal study could be run remotely. This would be difficult to do at present, for technical and ethical reasons, but is a possibility for the future.

User Surveys

User surveys were a primary method for collecting data across all studies. A mix of Likert scales and open-ended questions enabled a breadth of information to be collected. With Likert scales, topics could be quickly analysed and comparisons made quantitatively. The open-ended questions elicited qualitative responses beyond those targeted by the questionnaire. Regarding the remote study in Chapter 9, it was difficult to engage everyone. Only one-third of participants completed the survey, thereby highlighting that responses came from the engaged and the enthusiastic.

In all variants of survey B, question 5 was optional and few participants answered it. The questions specifically targeted the use of dependency arrows, which highlight structural dependencies when using reference parts. Even the participants who did use reference parts in the longitudinal study (Chapter 10), ignored it. In hindsight, the question should not have been marked optional. Instead, an option such as *"did not use"* could have been added, requiring users to acknowledge they did not engage with this feature.

Although the CDN framework was useful for designing the software, it was perhaps not as useful for designing questions in the user survey. Some of the questions, for which the researcher was most interested in obtaining answers did not align with a dimension. For example, many topics sought to understand the role of the computer as an author, which is fundamentally not what the CDN is useful for.

Interaction Logging

Interaction logging was selected as a means of unobtrusively collecting data. In practice, this produced thousands of data points. It became increasingly difficult to find patterns. In the end, other methodologies became the primary data sources, supplemented by the interaction data. Brown (2020) made similar observations when employing mixed methods for studying interaction in end-users of digital music instruments. Interaction logging is more useful when the sample size is increased, and/or users spend longer periods with the software (Nash, 2011), making other time-invasive methodologies less feasible.

As noted by Gerken *et al.* (2008), users' behaviour cannot always be explained by interaction data, and often leaves more questions than answers. The various unexplainable anomalies and outliers found in the uncontrolled conditions study (Chapter 9, page 146) broadly support this statement. Observing users of third party commercial music composition software is difficult (if not impossible) with interaction logging as this would require modifying the underlying source code (and gaining access to it).

11.3. Research limitations and Future Directions

The research was limited by its reliance on finding individuals interested in participating in the user studies. Specifically, finding experienced music practitioners who could dedicate time to learning all of IGME's intricacies, proved difficult.

The onus on the researcher to develop and test a complex interactive environment to be robust enough to support real-world usage limited the scope for further development. However, the novelty of IGME and its enablement of computer-generated composition gave the project sufficient scope that it was deemed unnecessary to develop more complex computer music models. The tools offered by the IGME environment are still quite rudimentary. Given IGME's modular architecture, there is ample opportunity to develop more complex plug-ins and effects in future research. There is also an opportunity to open-source the IGME software.

Generally, IGME was well-received throughout the studies, notably from Likert scale data. However, research from Dell *et al.* (2012) suggests that respondents are about 2.5x more likely to prefer technology developed by the interviewer (or researcher), than an alternative (although the users of IGME were not given an alternative software to work with). In addition, it is well understood by psychologists that study participants often modify their answers or behaviour by trying to second guess what the researcher is trying to achieve (McCambridge, De Bruin, and Witton, 2012). It is therefore worth noting that the findings presented here may indicate overly positive participant bias. Upon reflection, little could be done to compensate for this in the scope of the research.

11.3.1. Musical Affordances and aesthetics

All music technology products and tools dictate the music they afford the user. For example, sheet music editors (i.e. Sibelius) encourage the user to develop formal scores, whereas pattern-based sequencers (i.e. Ableton Live) encourage the development of loop-based music. When individuals use IGME software, they create *'IGME'* music. The software provides a set of constraints and limitations, giving the music that is created a certain aesthetic and a stylistic fingerprint.

The software mostly enforced the development of music into small fragments isolated from the surrounding musical context. IGME's inability to support music outside a 4/4 meter created rigidity in the musical artefacts. The heavy emphasis on computer-generated processes somewhat forced participants into using them, meaning perhaps fewer of their own musical signatures were evident in the resultant musical artefacts. The ratio of human vs computer-generated music would likely be higher, as discovered in this thesis, than if such features were already an integral part of mainstream music composition software.

Reflecting on the above, McPherson and Lepri (2020) state that *"Although a tool may theoretically be capable of anything, it will still have certain idiomatic patterns, making some structures and concepts easier or more obvious to the designer than others."* In summary, IGME has made using CGM easier, at the expense of making regular composition harder and pre-loading users with an expectation that they use CGM.

The majority of IGME's development and evaluation was provided via feedback from participatory design studies. Most participants were music technology students. Having had minimal exposure to professional music practice, or to postgraduate education (most were undergraduates), their inputs undoubtedly influenced how IGME evolved from initial conception to real-world software. In summary, the software design and features became tailored to the demographic using it. As already discussed, IGME had limited use and feedback from highly experienced music practitioners, so it is unclear how the design of such a tool would have differed if those practitioners had been involved. In addition, Dahlbäck and Karsvall (2000) notes that *"voluntary participants are not a representative sample of the general population, but are likely more extrovert and have more open personalities."*

IGME's reliance on the concepts of Western music limits its scope. Although the barrier to entry has been dropped, IGME has only democratised CGM tools for a computer-literate Western music-focused demographic. Little could be done to address this within the scope of the research. Such an observation is a common criticism of many musical interfaces in communities such as NIME (Morreale *et al.*, 2020). Future research should look deeper into this.

11.3.2. Future Directions

The range of computer-generated techniques integrated into IGME only scratches the surface of the field. Future research could focus on this aspect and on the automatic arrangements of high-level structure. A supplementary review of algorithmic representations of high-level structure (Hunt, Mitchell, and Nash, 2019), and the study of repetition (Hunt, 2020), was conducted by the researcher but is not discussed in the boy of this thesis (see Appendix item G).

To recap, most studies were conducted in controlled conditions with comparably small numbers of users. The resultant compositions were short and experimental in nature. Commissioning a series of longer pieces by an experienced IGME practitioner would offer further insight into the juxtaposition of human and computer-composed music. Moreover, future work could focus on analysing human-computer music and comparing it with existing styles, forms, and genres of music.

11.4. Closing Words

This thesis began with the observation: computer music researchers dream of the perfect algorithm, in which the music generated is indistinguishable from, or even superior to, that composed by the world's most talented practitioners. Although many researchers are still motivated to fulfil this ambition, many music practitioners are dismissive of the idea, as this thesis shows. As previously stated, we propose that computer-generated music techniques be used as tools that work alongside human composers, acting as a catalyst for human creativity rather than replacing it.

Although computers can automate many activities and can disenfranchise humans, we argue that computer-generated music and human composition are not at odds with each other. The world still needs exceptional composers, and computer music researchers still need to solve impossible problems. For now, at least, composers can sleep soundly knowing their practice is still safe from novel automation, but might yet be able to further harness the mysterious, intricate, complex, and open-ended 'technology' that is the *computer*.

Bibliography

Aaron, S. (2016) Sonic Pi–performance in education, technology and art. *International Journal of Performance Arts and Digital Media*. 12 (2), pp. 171–178.

Aaron, S. and Blackwell, A. F. (2013) From Sonic Pi to Overtone: Creative Musical Experiences with Domain-Specific and Functional Languages. In: *Proceedings of the first ACM SIGPLAN workshop on Functional art, music, modeling and; design*. Boston Massachusetts USA, Sept. 2013. ACM. pp. 35–46.

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., *et al.* (2016) Tensorflow: A system for large-scale machine learning. In: *12th Symposium on Operating Systems Design and Implementation*. 2016. Google. pp. 265–283.

Ableton. (2020) *Ableton Live*. Available From: `https://www.ableton.com/en/` [Last Accessed: 23/04/2020].

Acar, S., Burnett, C., and Cabra, J. F. (2017) Ingredients of creativity: Originality and more. *Creativity Research Journal*. 29 (2), pp. 133–144.

Adamatzky, A. (2010) *Game of life cellular automata*. New York: Springer.

Almeida, D. A., Murphy, G. C., Wilson, G., and Hoye, M. (2017) Do software developers understand open source licenses? In: *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*. 2017. pp. 1–11.

Alpern, A. (1995) Techniques for algorithmic composition of music. *Citeseer*. 95 pp. 120–137.

Alty, J. (1995) Navigating Through Compositional Space: The Creativity Corridor. *Leonardo*. 28 (3), pp. 215–219.

Amabile, T. M., Hadley, C. N., and Kramer, S. J. (2002) Creativity under the gun. *Harvard business review*. 80 pp. 52–63.

Ames, C. (1989) The Markov process as a compositional model: A survey and tutorial. *Leonardo*. pp. 175–187.

AmperMusic. (2020) *Amper Music*. Available From: `https://www.ampermusic.com/`.

Ampify-Music. (2020) *Launchpad: Make and Remix Music*. Available From: `https://apps.apple.com/gb/app/launchpad-make-remix-music/id584362474` [Last Accessed: 01/07/2020].

Antoine, A. and Miranda, E. (2016) A User-Centric Algorithmic Composition System. Antoine, A. and Miranda, E. In: *Proceedings of the 2016 International Conferences on New Music Concepts*. Treviso, Italy, Mar. 2016. ICNMC.

Apple. (2020) *GarageBand for mac*. Available From: `https://www.apple.com/uk/mac/garageband/` [Last Accessed: 01/07/2020].

Apple. (2020) *Logic Pro X*. Available From: `https://www.apple.com/uk/logic-pro/` [Last Accessed: 23/04/2020].

Ariza, C. (2005) *An Open Design for Computer-Aided Algorithmic Music Composition*. Boca Raton, Florida: Universal-Publishers.

Ariza, C. (2009) The interrogator as critic: The turing test and the evaluation of generative music systems. *Computer Music Journal*. 33 (2), pp. 48–70.

Arobas-Music. (2020) *Guitar Pro 7.5*. Available From: `https://www.guitar-pro.com/en/index.php` [Last Accessed: 23/04/2020].

Assayag, G., Rueda, C., Laurson, M., Agon, C., and Delerue, O. (1999) Computer-assisted composition at IRCAM: From PatchWork to OpenMusic. *Computer Music Journal*. 23 (3), pp. 59–72.

AudioTool. (2020) *Audio Tool*. Available From: `https://www.audiotool.com/` [Last Accessed: 23/04/2020].

Avid. (2020) *Pro Tools*. Available From: `https://www.avid.com/Audio` [Last Accessed: 23/04/2020].

Avid. (2020) *Sibelius*. Available From: `https://www.avid.com/sibelius` [Last Accessed: 23/04/2020].

Bellingham, M., Holland, S., and Mulholland, P. (2014) A cognitive dimensions analysis of interaction design for algorithmic composition software. Boulay, B. and Good, J. In: *Proceedings of Psychology of Programming Interest Group Annual Conference*. Old Ship Hotel, Brighton, 25-27th June 2014. University of Sussex. pp. 135–140.

Bellingham, M., Holland, S., and Mulholland, P. (2014) An analysis of algorithmic composition interaction design with reference to cognitive dimensions. *Technical report: The Open University*.

Berg, P. (2018) *AC Toolbox*. Available From: `http://www.actoolbox.net/` [Last Accessed: 23/04/2020].

Bertram, D. (2007) *Likert scales are the meaning of life*. Available From: `https://www.academia.edu/8160815/Likert_Scales_are_the_meaning_of_life` [Last Accessed: 01/07/2020].

Biewald, L. (2015) *Why human-in-the-loop computing is the future of machine learning*. Available From: `https://www.computerworld.com/article/3004013/why-human-in-the-loop-computing-is-the-future-of-machine-learning.html` [Last Accessed: 23/07/2020].

Biles, J. (1994) GenJam: A Genetic Algorithm for Generating Jazz Solos. In: *Proceedings of the International Computer Music Conference*. Denmark, Dec. 1994. Michigan Publishing. pp. 131–131.

Biles, J. A. (2002) GenJam: Evolution of a jazz improviser. *Creative evolutionary systems*. 168 p. 2.

Bitwig. (2020) *Bitwig Studio*. Available From: `https://www.bitwig.com/en/home.html` [Last Accessed: 23/04/2020].

Blackwell, A. F. and Collins, N. (2005) The Programming Language as a Musical Instrument. In: *Proceedings of 17th Psychology of Programming Interest Group*. Brighton, UK, 29-1 June 2005. Psychology of Programming Interest Group. p. 11.

Boden, M. A. (2004) *The creative mind: Myths and mechanisms*. unknown: Psychology Press.

Bouche, D., Nika, J., Chechile, A., and Bresson, J. (2017) Computer-aided composition of musical processes. *Journal of New Music Research*. 46 (1), pp. 3–14.

Boyd, J. (2013) *It's Not Only Rock 'n' Roll*. 2nd. London, England: John Blake Publishing Ltd.

Brabazon, A., O'Neill, M., and McGarraghy, S. (2015) *Natural computing algorithms*. Berlin, Germany: Springer.

Braun, V. and Clarke, V. (2006) Using thematic analysis in psychology. *Qualitative research in psychology*. 3 (2), pp. 77–101.

Bresson, J., Agon, C., and Assayag, G. (2011) OpenMusic: Visual Programming Environment for Music Composition, Analysis and Research. In: *Proceedings of the 19th ACM International Conference on Multimedia*. Scottsdale, Arizona, USA, Nov. 2011. ACM. pp. 743–746.

Briot, J.-P., Hadjeres, G., and Pachet, F.-D. (2020) *Deep learning techniques for music generation*. Berlin, Germany: Springer.

Brooks Jr, F., Hopkins Jr, A., Neumann, P. G., and Wright, W. (1957) An Experiment in Musical Composition. *Electronic Computers, IRE Transactions on*. 6 (3), pp. 175–182.

Brown, A. R. (2004) An aesthetic comparison of rule-based and genetic algorithms for generating melodies. *Organised Sound*. 9 (2), pp. 191–197.

Brown, A. R. and Kerr, T. (2009) Adaptive music techniques. In: *Improvise: The Australasian Computer Music Conference 2009*. Queensland University of Technology Brisbane, Australia, Feb. 2009. Australasian Computer Music Association. pp. 26–31.

Brown, D. (2020) *End-user action-sound mapping design for mid-air music performance*. PhD thesis, The University of the West of England (UWE).

Brown, D., Nash, C., and Mitchell, T. (2017) A User Experience Review of Music Interaction Evaluations. In: *17th International Conference on New Interfaces for Musical Expression*. Aalborg University, Copenhagen., 15-18 May 2017. NIME. pp. 370–375.

Bryden, K. (2006) Using a Human-in-the-Loop evolutionary algorithm to create Data-Driven music. In: *2006 IEEE International Conference on Evolutionary Computation*. Vancouver, Canada, 16-21 July 2006. IEEE. pp. 2065–2071.

Budgen, D. (2003) *Software design*. 2nd. Boston Massachusetts, USA: Addison-Wesley.

Burraston, D. and Edmonds, E. (2005) Cellular automata in generative electronic music and sonic art: a historical and technical review. *Digital Creativity*. 16 (3), pp. 165–185.

Burraston, D., Edmonds, E., Livingston, D., and Miranda, E. R. (2004) Cellular automata in MIDI based computer music. In: *Proceedings of the 2004 International Computer Music Conference*. Frost School of Music, University of Miami, Jan. 2004. International Computer Music Association.

Buxton, W. A. (1977) A composer's introduction to computer music. *Journal of New Music Research*. 6 (2), pp. 57–71.

Card, S. K. (2018) *The psychology of human-computer interaction*. Florida, United States: Crc Press.

Carroll, J. M. (2003) *HCI models, theories, and frameworks: Toward a multidisciplinary science*. Massachusetts, United States: Morgan Kaufmann.

Chapel, R. H. (2003) Some projects and reflections on algorithmic Music. In: *International Symposium on Computer Music Modeling and Retrieval*. Montpellier, France, 26-27 May 2003. Springer. pp. 1–12.

Choi, K., Fazekas, G., and Sandler, M. (2016) Text-based LSTM networks for Automatic Music Composition. In: *Conference on Computer Simulation of Musical Creativity*. Huddersfield. UK, 17-19 June 2016. University of Huddersfield.

Chomsky, N. (1957) *Syntactic structures.* unknown: Mouton.

Cockos. (2020) *Repear*. Available From: `https://www.reaper.fm/` [Last Accessed: 23/04/2020].

Collins, D. (2005) A synthesis process model of creative thinking in music composition. *Psychology of music*. 33 (2), pp. 193–216.

Collins, M. A. and Amabile, T. M. (1999) Motivation and creativity. *Handbook of creativity*. 297 pp. 1051–1057.

Collins, N. (2018) '... there is no reason why it should ever stop': large-scale algorithmic composition. *Journal of creative music systems.* 3 (1), pp. 1–25.

Pd-community. (2018) *Pure Data*. Available From: `https://puredata.info/` [Last Accessed: 29/08/2018].

Coorevits, E., Moelants, D., Östersjö, S., Gorton, D., and Leman, M. (2015) Decomposing a composition: On the multi-layered analysis of expressive music performance. In: *International Symposium on Computer Music Multidisciplinary Research*. Plymouth, UK, 16-19 June 2015. Springer. pp. 167–189.

Cope, D. (1991) *Computers and Musical Style*. Middleton, Wisconsin: A-R Editions.

Cope, D. and Mayer, M. J. (1996) *Experiments in musical intelligence*. Madison, WI: AR editions.

Craft, B. and Cairns, P. (2005) Beyond guidelines: what can we learn from the visual information seeking mantra? In: *Ninth International Conference on Information Visualisation (IV'05)*. London, Uk, June 2005. IEEE. pp. 110–118.

Crutchfield, R. S. (1962) *Conformity and creative thinking.* unknown: Atherton Press.

Csikszentmihalyi, M. (1990) *Flow: The psychology of optimal experience*. Australia: Harper Collins.

Csikszentmihalyi, M. (1990) The domain of creativity. *Sage focus editions*. 115 pp. 190–212.

Csikszentmihalyi, M. (1996) *Flow and the psychology of discovery and invention*. New York: Harper Perennial.

Dahlbäck, N. and Karsvall, A. (2000) A.: Personality Bias in Volunteer Based User Studies. In: *In: Proceedings of HCI 2000 the 14th Annual Conference of the British HCI Group*. May 2000. Citeseer.

Davis, P. (2020) *ARDOUR*. Available From: `https://ardour.org/` [Last Accessed: 23/04/2020].

Dell, N., Vaidyanathan, V., Medhi, I., Cutrell, E., and Thies, W. (2012) " Yours is better!" participant response bias in HCI. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Austin Texas USA, May 2012. Association for Computing Machinery. pp. 1321–1330.

Diaz-Jerez, G. (2011) Composing with Melomics: Delving into the computational world for musical inspiration. *Leonardo Music Journal*. 21 (21), pp. 13–14.

Doraisamy, S. (2004) *Polyphonic music retrieval: The n-gram approach*. PhD thesis, University of London London.

Duignan, M. (2010) Computer mediated music production: A study of abstraction and activity. *Computer Music Journal*. 34 (4), pp. 22–33.

Duignan, M., Noble, J., Barr, P., and Biddle, R. (2004) Metaphors for Electronic Music Production in Reason. In: *Computer Human Interaction: 6th Asia Pacific Conference, APCHI 2004, Rotorua, New Zealand, June 29-July 2, 2004, Proceedings*. Rotorua, New Zealand, 29-2 June 2004. Springer. p. 111.

Eck, D. and Schmidhuber, J. (2002) Finding temporal structure in music: Blues improvisation with LSTM recurrent networks. In: *Neural Networks for Signal Processing, 2002. Proceedings of the 2002 12th IEEE Workshop on*. Martigny, Switzerland, June 2002. IEEE. pp. 747–756.

Edwards, M. (2011) Algorithmic composition: computational thinking in music. *Communications of the ACM*. 54 (7), pp. 58–67.

Eiben, A. E. and Schoenauer, M. (2002) Evolutionary computing. *Information Processing Letters*. 82 (1), pp. 1–6.

Electronics, P. A. (2020) *Studio One*. Available From: `https://www.presonus.com/products/studio-one/` [Last Accessed: 23/04/2020].

Eno, B. and Chilvers, P. (2017) *Apps by Brian Eno and Peter Chilvers*. Available From: `http://www.generativemusic.com/` [Last Accessed: 29/08/2020].

Erickson, F. (2006) Definition and analysis of data from videotape: Some research procedures and their rationales. *Handbook of complementary methods in education research*. 3 pp. 177–192.

Fernández, J. D. and Vico, F. (2013) AI Methods in Algorithmic Composition: A Comprehensive Survey. *Journal of Artificial Intelligence Research*. 48 pp. 513–582.

Finke, R. A., Ward, T. B., and Smith, S. M. (1992) *Creative cognition: Theory, research, and applications*. Cambridge, MA: MIT press.

FitzGerald, E. (2012) Analysing video and audio data: Existing approaches and new innovations. In: *Surface Learning Workshop*. Bristol, UK, 18-20 March 2012.

Five12. (2020) *Numerology*. Available From: `http://www.five12.com/` [Last Accessed: 23/04/2020].

Föllmer, G. (2005) Electronic, aesthetic and social factors in net music. *Organised Sound*. 10 (3), pp. 185–192.

Freed, A. (1997) Open sound control: A new protocol for communicating with sound synthesizers. In: *International Computer Music Conference (ICMC)*. Thessaloniki, Greece, 25-30 September 1997. ICMC.

Gartland-Jones, A. (2002) Can a genetic algorithm think like a composer. Soddu, C. In: *In Generative Art 2002: 5th International Generative Art Conference*. Milan, Italy, Nov. 2002. Citeseer. pp. 1–12.

Geiger, C., Reckter, H., Paschke, D., Schulz, F., Poepel, C., and Ansbach, F. (2008) Towards Participatory Design and Evaluation of Theremin-based Musical Interfaces. In: *8th International Conference on New Interfaces for Musical Expression, NIME 2008*. Genova, Italy, May 2008. NIME. pp. 303–306.

Gerken, J., Bak, P., Jetter, C., Klinkhammer, D., and Reiterer, H. (2008) How to use interaction logs effectively for usability evaluation. In: *Position Paper presented to CHI 2008 Workshop BELIV'08: Beyond time and errors - novel evaLuation methods for Information Visualization*. Florence, Italy, Apr. 2008. ACM.

Gill, S. (1963) A Technique for the Composition of Music in a Computer. *The Computer Journal*. 6 (2), pp. 129–133.

Gillick, J., Tang, K., and Keller, R. M. (2010) Machine learning of jazz grammars. *Computer Music Journal*. 34 (3), pp. 56–66.

Green, T. R. G. and Petre, M. (1996) Usability analysis of visual programming environments: a 'cognitive dimensions' framework. *Journal of Visual Languages &amp; Computing*. 7 (2), pp. 131–174.

Grossoehme, D. H. (2014) Overview of qualitative research. *Journal of health care chaplaincy*. 20 (3), pp. 109–122.

Guzdial, M. (2010) *Why is it so hard to learn to program*. Massachusetts, United States: O'Reilly Media.

Hanna, W. (2007) The new Bloom's taxonomy: Implications for music education. *Arts Education Policy Review*. 108 (4), pp. 7–16.

Henningsson, D. and Team, F. (2011) FluidSynth real-time and thread safety challenges. In: *Proceedings of the 9th International Linux Audio Conference, Maynooth University, Ireland*. Maynooth, Ireland, June 2011. pp. 123–128.

Hill, L. (2019) Blackboard Collaborate Ultra: An Online, Interactive Teaching Tool. *Academy of Management Learning &amp; Education*. 18 (4), pp. 640–642.

Hiller, L. A. and Baker, R. A. (1964) Computer Cantata: A study in compositional method. *Perspectives of New Music*. 3 (1), pp. 62–90.

Hiller, L. A. and Isaacson, L. M. (1979) *Experimental Music; Composition with an electronic computer (republished)*. New York: McGraw-Hill.

Hiller, L. A. and Isaacson, L. M. (1959) *Experimental music: composition with an electronic computer*. unknown: McGraw-Hill.

Holland, S. (2000) Artificial Intelligence in Music Education: A Critical Review. *Readings in Music and Artificial Intelligence. Contemporary Music Studies*. 20 pp. 239–274.

Holtzman, S. (1981) Using generative grammars for music composition. *Computer Music Journal*. 5 (1), pp. 51–64.

HookPad. (2020) *HookTheory*. Available From: `https://hookpad.hooktheory.com/` [Last Accessed: 01/08/2020].

Hoos, H. H., Hamel, K. A., Renz, K., and Kilian, J. (1998) *The GUIDO Notation Format–A Novel Approach for Adequately Representing Score-Level Music*. Citeseer.

Horner, A. and Goldberg, D. E. (1991) *Genetic algorithms and computer-assisted music composition*. Michigan Publishing, University of Michigan Library: Ann Arbor.

Hsu, W. T. and Sosnick, M. H. (2009) Evaluating Interactive Music Systems: An HCI Approach. In: *NIME*. Pittsburgh, Pennsylvania, Apr. 2009. NIME. pp. 25–28.

Hunt, S. (2020) An analysis of repetition in video game music. Strum, B. In: *Proceedings of the 2020 Joint Conference on AI Music Creativity*. Stockholm, Sweden, 19-23 October 2020. Royal Institute of Technology (KTH).

Hunt, S. (2021) *IGME the Interactive Generative Music Environment*. Available From: `http://samhunt.panel.uwe.ac.uk/TEHome.html` [Last Accessed: 01/03/2021].

Hunt, S., Mitchell, T., and Nash, C. (2018) A cognitive dimensions approach for the design of an interactive generative score editor. Bhagwati, S. and Bresson, J. In: *Proceedings of the International Conference on Technologies for Music Notation and Representation – TENOR'18*. Montreal, Canada, 24-26 May 2018. Concordia University. pp. 119–127.

Hunt, S., Mitchell, T., and Nash, C. (2019) Automating algorithmic representations of musical structure using IGME: The Interactive Generative Music Environment. Paterson, J. In: *Innovation In Music 2019*. London, Uk, May 2019. University of West London.

Hunt, S., Mitchell, T., and Nash, C. (2020) Composing computer generated music, an observational study using IGME the Interactive Generative Music Environment. Michon, R. and Schroeder, F. In: *NIME 2020*. Royal Birmingham Conservatoire, UK, 21-25 July 2020. Birmingham City University. pp. 61–66.

Hunt, S., Nash, C., and Mitchell, T. (2017) Thoughts on Interactive Generative Music Composition. Laney, R. In: *2nd Conference on Computer Simulation of Musical Creativity*. Milton Keynes, Uk, Nov. 2017. Open University.

Image-Line. (2020) *FL Studio 20*. Available From: `https://www.image-line.com/flstudio/` [Last Accessed: 23/04/2020].

Intermorphic. (2018) *Generative Music*. Available From: `https://intermorphic.com/sseyo/koan/generativemusic1/` [Last Accessed: 12/07/2020].

Intermorphic. (2018) *SSEYO Koan*. Available From: `https://intermorphic.com/sseyo/koan/` [Last Accessed: 12/07/2020].

Intermorphic. (2021) *Generative Music Composer Plugin by Intermorphic*. Available From: `https://www.kvraudio.com/product/noatikl-by-intermorphic`.

Jackendoff, R. and Lerdahl, F. (1996) *A generative theory of tonal music*. 2nd. Cambridge, Massachusetts: MIT Press.

Jacob, B. (1995) Composing with genetic algorithms. In: *Proceedings of the International Computer Music Conference 1995*. Canada, Mar. 1995. International Computer Music Association.

Jacob, B. L. (1996) Algorithmic Composition as a Model of Creativity. *Organised Sound*. 1 (03), pp. 157–165.

Jeong, J., Kim, N., and In, H. P. (2020) GUI information-based interaction logging and visualization for asynchronous usability testing. *Expert Systems with Applications*. pp. 113–189.

Kantosalo, A., Toivanen, J. M., Xiao, P., and Toivonen, H. (2014) From Isolation to Involvement: Adapting Machine Creativity Software to Support Human-Computer Co-Creation. Colton, S., Ventura, D., Lavrač, N., and Cook, M. In: *Proceedings of the Fifth International Conference on*

Computational Creativity. Ljubljana, Slovenia, Sept. 2014. Institute Jozef Stefan, Ljubljana. pp. 1–7.

Kartika, G. (2010) *Midi Composition Tools Using JFugue Java API*. Available From: `https://www.academia.edu/2176488/MIDI_CompoSITion_Tools_dengan_menggunakan_JFugue_Java_API` [Last Accessed: 11/05/2020].

Katz, M. (2010) *Capturing sound: how technology has changed music*. Berkeley, California, United States: University of California Press.

Keller, R. M. and Morrison, D. R. (2007) A grammatical approach to automatic improvisation. In: *Proceedings, Fourth Sound and Music Conference*. Athens, Greece, Nov. 2007. pp. 330–337.

Koenig, G. M. (1970) PROJECT 2: A programme for musical composition. *Electronic Music Reports*. 3 (970), pp. 4–6.

Koenig, G. M. (1971) The use of computer programmes in creating music. In: *Music and Technology (Proceedings of the Stockholm Meeting organized by UNESCO)*. 1971. pp. 93–115.

Kohonen, T. (1989) A Self-Learning Musical Grammar, or ”Associative Memory of the Second Kind”. In: *Proceedings of International Joint Conference on Neural Networks*. Washington, DC, USA, 1989. IEEE. pp. 1–5.

Kolokolova, A., Billard, M., Bishop, R., Elsisy, M., Northcott, Z., Graves, L., Nagisetty, V., and Patey, H. (2020) *GANs and Reels: Creating Irish Music using a Generative Adversarial Network*. Available From: `https://arxiv.org/abs/2010.15772` [Last Accessed: 1/4/2020].

Krathwohl, D. R. (2002) A revision of Bloom's taxonomy: An overview. *Theory into practice*. 41 (4), pp. 212–218.

Langkjær-Bain, R. (2018) Five ways data is transforming music. *Significance*. 15 (1), pp. 20–23.

Leinonen, T., Toikkanen, T., and Silfvast, K. (2008) Software as hypothesis: research-based design methodology. Hakken, D. In: *Proceedings of the tenth anniversary conference on participatory design 2008*. Indianapolis, United States, Jan. 2008. Indiana University. pp. 61–70.

Lettner, F. and Holzmann, C. (2012) Automated and unsupervised user interaction logging as basis for usability evaluation of mobile applications. In: *Proceedings of the 10th International Conference on Advances in Mobile Computing &amp; Multimedia*. Bali, Indonesia, Mar. 2012. ACM. pp. 118–127.

Liang, F. (2016) *Bachbot: Automatic composition in the style of bach chorales*. PhD thesis, University of Cambridge.

Lidov, D. and Gabura, J. (1973) A melody writing algorithm using a formal language model. *Computers in the Humanities*. 3 pp. 138–48.

Liikkanen, L., Amos, C., Cunningham, S. J., Downie, J. S., and McDonald, D. (2012) Music interaction research in HCI: let's get the band back together. Konstan Joseph, A. In: *CHI'12 Extended Abstracts on Human Factors in Computing Systems*. Austin Texas USA, May 2012. ACM. pp. 1119–1122.

Lim, C. K., Tan, K. L., Yusran, H., and Suppramaniam, V. (2017) LSound: An L-System Framework for Score Generation. *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*. 9 (2-11), pp. 159–163.

Liu, C.-H. and Ting, C.-K. (2017) Computational Intelligence in Music Composition: A Survey. *IEEE Transactions on Emerging Topics in Computational Intelligence*. 1 (1), pp. 2–15.

Liu, I. and Ramakrishnan, B. (2014) Bach in 2014: Music Composition with Recurrent Neural Network. Yoshua, B. and Yann, L. In: *3rd International Conference on Learning Representations*. San Diego, USA, July 2014. arXiv.

Loeliger, J. and McCullough, M. (2012) *Version Control with Git: Powerful tools and techniques for collaborative software development*. Massachusetts, United States: O'Reilly Media, Inc.

Lorrain, D. (1980) A panoply of stochastic'cannons'. *Computer Music Journal*. 4 (1), pp. 53–81.

Loughran, R. and O'Neill, M. (2020) Evolutionary music: applying evolutionary computation to the art of creating music. *Genetic Programming and Evolvable Machines*. 21 pp. 55–85.

Lubart, T. (2005) How can computers be partners in the creative process: classification and commentary on the special issue. *International Journal of Human-Computer Studies*. 63 (4-5), pp. 365–369.

Madden, C. (1999) *Fractals in Music: Introductory Mathematics for Musical Analysis*. London: High Art Press.

Maes, P.-J., Leman, M., Palmer, C., and Wanderley, M. (2014) Action-based effects on music perception. *Frontiers in psychology*. 4 p. 1008.

MAGIX. (2020) *Acid Pro 10*. Available From: `https://www.magix.com/gb/music/acid/acid-pro/` [Last Accessed: 23/04/2020].

Makemusic. (2018) *Finale*. Available From: `https://www.finalemusic.com/` [Last Accessed: 03/09/2018].

Manzo, V. J. (2016) *Max/MSP/Jitter for Music: A Practical Guide to Developing Interactive Music Systems for Education and More*. Oxford, Uk: Oxford University Press.

McAlpine, K., Miranda, E., and Hoggar, S. (1999) Making music with algorithms: A case-study system. *Computer Music Journal*. 23 (2), pp. 19–30.

McCambridge, J., De Bruin, M., and Witton, J. (2012) The effects of demand characteristics on research participant behaviours in non-laboratory settings: a systematic review. *PloS one*. 7 (6), e39116.

McCarney, R., Warner, J., Iliffe, S., Van Haselen, R., Griffin, M., and Fisher, P. (2007) The Hawthorne Effect: a randomised, controlled trial. *BMC medical research methodology*. 7 (1), p. 30.

McCormack, J. (1996) Grammar based music composition. *Complex systems*. 96 pp. 321–336.

McCormack, J., McIlwain, P., Lane, A., and Dorin, A. (2007) Generative composition with Nodal. Almeida Costa, F. In: *Workshop on music and artificial life (part of The 9th European Conference on Artificial Life)*. Lisbon, Portugal, Oct. 2007. Citeseer.

McPherson, A. P., Jack, R. H., Moro, G., *et al.* (2016) Action-sound latency: Are our tools fast enough? In: *Proceedings of the International Conference on New Interfaces for Musical Expression*. Brisbane, Australia, Nov. 2016. Queensland Conservatorium Griffith University. pp. 20–25.

McPherson, A. and Lepri, G. (2020) Beholden to our tools: negotiating with technology while sketching digital instruments. Michon, R. and Schroeder, F. In: *Proceedings of the International Conference on New Interfaces for Musical Expression*. Birmingham, UK, 21-25 July 2020. Birmingham City University. pp. 434–439.

Meyer, J. and Land, R. (2003) Threshold concepts and troublesome knowledge: Linkages to ways of thinking and practising within the disciplines. *Improving Student Learning - Ten Years on*. 1 pp. 412–424.

Miranda, E. R. (2001) *Composing music with computers*. Oxford, Uk: Focal Press.

Miranda, E. R. (1993) Cellular automata music: An interdisciplinary project. *Journal of New Music Research*. 22 (1), pp. 3–21.

Miranda, E. R. (2021) *Handbook of Artificial Intelligence for Music: Foundations, Advanced Approaches, and Developments for Creativity*. New York, United States: Springer.

Mochihashi, D. and Sumita, E. (2008) The infinite Markov model. In: *Twenty-Second Annual Conference on Neural Information Processing Systems*. Vancouver, Canada, Aug. 2008. NIPS. pp. 1017–1024.

Morreale, F., Bin, S. A., McPherson, A., Stapleton, P., and Wanderley, M. (2020) A nime of the times: Developing an outward-looking political agenda for this community. In: *Proceedings of the International Conference on New Interfaces for Musical Expression*. Birmingham, UK, 21-25 July 2020. Birmingham City University. pp. 160–165.

MOTU. (2020) *DP10*. Available From: `https://motu.com/en-us/products/software/dp/` [Last Accessed: 23/04/2020].

Mozer, M. C. (1994) Neural Network Music Composition by Prediction: Exploring the Benefits of Psychoacoustic Constraints and Multi-Scale Processing. *Connection Science*. 6 (2-3), pp. 247–280.

Muller, M. J. and Kuhn, S. (1993) Participatory design. *Communications of the ACM*. 36 (6), pp. 24–28.

Myhill, J. (1979) Controlled Indeterminacy A First Step Towards a Semi-Stochastic Music Language. *Computer Music Journal*. 3 (3), pp. 12–14.

Nash, C. (2011) *Supporting Virtuosity and Flow in Computer Music*. PhD thesis, University Of Cambridge.

Nash, C. (2014) Manhattan: End-user programming for music. In: *14th International Conference on New Interfaces for Musical Expression*. Goldsmiths, University of London, 30 June 2014. Goldsmiths, University of London. pp. 221–226.

Nash, C. (2015) The Cognitive Dimensions of Music Notations. In: *Proceedings of the International Conference on Technologies for Music Notation and Representation - TENOR2015*. 28-30 May, 2015. Institut de Recherche en Musicologie.

Nash, C. and Blackwell, A. F. (2012) Liveness and Flow in Notation Use. In: *Proceedings of the International Conference on New Interfaces for Musical Expression*. Ann Arbor, Michigan, 21-23 May 2012. University of Michigan.

Nevels, D. L. (2013) Using music software in the compositional process: A case study of electronic music composition. *Journal of Music, Technology & Education*. 5 (3), pp. 257–271.

Nielsen, J. (1994) *Usability engineering*. Massachusetts, United States: Morgan Kaufmann.

Nodal. (2021) *Nodal 2.0*. Available From: `https://nodalmusic.com/` [Last Accessed: 01/03/2021].

Ohm-studio. (2020) *Ohm-studio*. Available From: `https://www.ohmstudio.com/` [Last Accessed: 07/07/2020].

Olson, H. F. and Belar, H. (1961) Aid to music composition employing a random probability system. *The Journal of the Acoustical Society of America*. 33 (9), pp. 1163–1170.

Pachet, F. (2003) The Continuator: Musical Interaction With Style. *Journal of New Music Research*. 32 (3), pp. 333–341.

Papadopoulos, G. and Wiggins, G. (1999) AI methods for Algorithmic Composition: A survey, a Critical View and Future prospects. Curry, B. In: *AISB Symposium on Musical Creativity*. Edinburgh, UK, Dec. 1999. AISB. pp. 110–117.

Pearce, M. T. (2007) *Early Applications of Information Theory to Music*. Available From: `https://www.semanticscholar.org/paper/Early-Applications-of-Information-Theory-to-Music-Pearce/1c91aa7074a2e9a68530f29eb0f8a1505c284b89#related-papers` [Last Accessed: 25/3/2021].

Pearce, M. T. and Wiggins, G. A. (2007) Evaluating cognitive models of musical composition. In: *Proceedings of the 4th international joint workshop on computational creativity*. Goldsmiths, University of London, 17-19 June 2007. Goldsmiths, University of London. pp. 73–80.

Peters, M. (2010) From Strange to Impossible: Interactive Attractor Music. *Contemporary Music Review*. 29 (4), pp. 395–404.

Potter, K. (2002) *Four Musical Minimalists: La Monte Young, Terry Riley, Steve Reich, Philip Glass*. Cambridge, UK: Cambridge University Press.

Preece, J., Sharp, H., and Rogers, Y. (2015) *Interaction design: beyond human-computer interaction*. 4th. New Jersey, United States: John Wiley.

Prusinkiewicz, P. (1986) Score generation with L-systems. In: *1986 International Computer Music Conference*. Den Haag, The Netherlands, 20-24 October 1986. Michigan Publishing. pp. 455–457.

Prusinkiewicz, P. and Lindenmayer, A. (2012) *The algorithmic beauty of plants*. Berlin, Germany: Springer.

Quintana, C. S., Arcas, F. M., Molina, D. A., Rodríguez, J. D. F., and Vico, F. J. (2013) Melomics: A case-study of AI in Spain. *AI Magazine*. 34 (3), pp. 99–103.

Rahn, J. (1993) Repetition. *Contemporary Music Review*. 7 (2), pp. 49–57.

Raines, R. (2015) *Composition in the Digital World: Conversations with 21st Century American Composers*. Oxford, Uk: Oxford University Press.

Reason-Studios. (2020) *Reason 11*. Available From: `https://www.reasonstudios.com/en/reason` [Last Accessed: 23/04/2020].

Reiners, P. (2004) *Cellular automata and music*. Available From: `https://www.ibm.com/developerworks/library/j-camusic/` [Last Accessed: 09/02/2019].

Renoise. (2020) *Renoise 3.2*. Available From: `https://www.renoise.com/` [Last Accessed: 23/04/2020].

Resnick, M., Myers, B., Nakakoji, K., Shneiderman, B., Pausch, R., Selker, T., and Eisenberg, M. (2005) Design principles for tools to support creative thinking. In: *NSF Workshop Report on Creativity Support Tools*. Washington DC, USA, June 2005. Carnegie Mellon University.

Rittel, H. W. and Webber, M. M. (1973) Dilemmas in a general theory of planning. *Policy sciences*. 4 (2), pp. 155–169.

Rivest, R. and Dusse, S. (1992) *The MD5 message-digest algorithm*. Available From: `https://tools.ietf.org/html/rfc1321` [Last Accessed: 1/12/2020].

Roberts, A., Engel, J., Mann, Y., Gillick, J., Kayacik, C., Nørly, S., Dinculescu, M., Radebaugh, C., Hawthorne, C., and Eck, D. (2019) Magenta studio: Augmenting creativity with deep learning in ableton live. *Goolge Research technical report*. 1.

Rosala, M. (2019) *How to Analyze Qualitative Data from UX Research: Thematic Analysis (Nielsen Norman Group)*. Available From: `https://www.nngroup.com/articles/thematic-analysis/` [Last Accessed: 24/3/2021].

Ruttkay, Z. (1997) Composing Mozart variations with dice. *Teaching Statistics*. 19 (1), pp. 18–19.

Sanders, E. B.-N. (2002) From user-centered to participatory design approaches. *Design and the social sciences*. pp. 18–25.

Service, T. (2012) *Thomas Adès: Full of Noises: Conversations with Tom Service*. New York, United States: Macmillan.

Sethi, R. (2018) *Top 12 Most Popular DAWs*. Available From: `https://ask.audio/articles/top-12-most-popular-daws-you-voted-for` [Last Accessed: 30/06/2020].

Shannon, C. E. (1948) A mathematical theory of communication. *Bell system technical journal*. 27 (3), pp. 379–423.

Shannon, C. E. (1953) Computers and automata. *Proceedings of the IRE*. 41 (10), pp. 1234–1241.

Sheikholharam, P. and Teshnehlab, M. (2008) Music composition using combination of genetic algorithms and recurrent neural networks. In: *Eighth International Conference on Hybrid*

Intelligent Systems, 2008. HIS'08. Universitat Politècnica de Catalunya, Oct. 2008. IEEE. pp. 350–355.

Shneiderman, B. (1996) The eyes have it: A task by data type taxonomy for information visualizations. In: *Proceedings of the IEEE Symposium on Visual Languages, 1996.* Boulder, USA, Mar. 1996. IEEE. pp. 336–343.

Siphocly, N. N. J., El-Horbaty, E.-S. M., and Salem, A.-B. M. (2021) Top 10 Artificial Intelligence Algorithms in Computer Music Composition. *International Journal of Computing and Digital Systems*. 10 (01), pp. 373–394.

Siphocly, N. N., Salem, A.-B. M., and El-Horabty, E.-S. M. (2021) Applications of Computational Intelligence in Computer Music Composition. *International Journal of Intelligent Computing and Information Sciences*. 21 (1), pp. 59–67.

Soddell, F. and Soddell, J. (2000) Microbes and music. In: *Pacific Rim International Conference on Artificial Intelligence*. Melbourne, VIC, Australia, 28-1 August 2000. Springer. pp. 767–777.

Sorensen, A. C. and Brown, A. R. (2008) A Computational Model for the Generation of Orchestral Music in the Germanic Symphonic Tradition: A Progress Report. Hood, A. and Wilkie, S. In: *Proceedings of Sound : Space - The Australasian Computer Music Conference*. Sydney, Australia, Oct. 2008. ACMA. pp. 78–84.

Sorensen, A. C. (2018) *Extempore: The design, implementation and application of a cyber-physical programming language*. PhD thesis, Australian National University.

Sorensen, A. and Gardner, H. (2010) Programming With Time: Cyber-Physical Programming With Impromptu. *ACM Sigplan Notices*. 45 (10), pp. 822–834.

Splice.com. (2021) *Splice Studio*. Available From: `https://splice.com/features/studio` [Last Accessed: 01/03/2021].

SSEYO. (1995) *Koan Pro User's Guide*. unknown: SSEYO Ltd.

Steinberg. (2020) *Cubase 10.5*. Available From: `https://www.steinberg.net/en/home.html` [Last Accessed: 23/04/2020].

Steinberg. (2020) *Dorico*. Available From: `https://new.steinberg.net/dorico/` [Last Accessed: 23/04/2020].

Sternberg, R. J. (2003) *Wisdom, intelligence, and creativity synthesized*. Cambridge, UK: Cambridge University Press.

Sturm, B. (2020) *The 2020 Joint Conference on AI Music Creativity*. Available From: `https://boblsturm.github.io/aimusic2020/` [Last Accessed: 24/12/2020].

Sturm, B. L., Ben-Tal, O., Monaghan, U., Collins, N., Herremans, D., Chew, E., Hadjeres, G., Deruty, E., and Pachet, F. (2019) Machine learning research that matters for music creation: A case study. *Journal of New Music Research*. 48 (1), pp. 36–55.

Sturm, B. L., Santos, J. F., Ben-Tal, O., and Korshunova, I. (2016) Music transcription modelling and composition using deep learning. In: *1st Conference on Computer Simulation of Musical Creativity*. Huddersfield. UK, 17 – 19 June 2016. University of Huddersfield.

Sukumaran, S. and Thiyagarajan, D. (2009) Generation of fractal music with Mandelbrot set. *Global Journal of Computer Science and Technology*. 9 (4), pp. 127–131.

Sulyok, C., McPherson, A., and Harte, C. (2016) Evolving the process of a virtual composer. *Natural Computing*. 18 pp. 47–60.

Takagi, H. (2001) Interactive evolutionary computation: Fusion of the capabilities of EC optimization and human evaluation. *Proceedings of the IEEE*. 89 (9), pp. 1275–1296.

Tanaka, A., Parkinson, A., Settel, Z., and Tahiroglu, K. (2012) A Survey and Thematic Analysis Approach as Input to the Design of Mobile Music GUIs. In: *Proceedings of NIME conference 2012*. University of Michigan, Ann Arbor., 21-23 May 2012. NIME.

Tang, J. C., Liu, S. B., Muller, M., Lin, J., and Drews, C. (2006) Unobtrusive but invasive: using screen recording to collect field data on computer-mediated interaction. Hinds, P. and Martin, D. In: *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*. Banff, Alberta Canada, Nov. 2006. ACM. pp. 479–482.

Temperley, D. (2007) *Music and probability*. Cambridge, Massachusetts: The MIT Press.

Thompson, K. (2002) A Critical Discourse Analysis of World Music as the Other in Education. *Research Studies in Music Education*. 19 (1), pp. 14–21.

Todd, P. M. (1989) A Connectionist Approach to Algorithmic Composition. *Computer Music Journal*. 13 (4), pp. 27–43.

Todd, R. L. (1978) Retrograde, inversion, retrograde-inversion, and related techniques in the masses of Jacobus Obrecht. *The Musical Quarterly*. 64 (1), pp. 50–78.

Togelius, J., Shaker, N., and Dormans, J. (2016) Procedural Content Generation in Games. In: *Grammars and L-systems with applications to vegetation and levels*. Springer. pp. 73–98.

Tracktion-Software-Corporation. (2020) *Waveform Pro*. Available From: `https://www.tracktion.com/products/waveform-pro` [Last Accessed: 23/04/2020].

Unehara, M. and Onisawa, T. (2003) Music composition system based on subjective evaluation. In: *IEEE International Conference on Systems, Man and Cybernetics, 2003.* Washington, DC, USA, Aug. 2003. IEEE. pp. 980–986.

Van Der Merwe, A. and Schulze, W. (2010) Music generation with markov models. *IEEE MultiMedia*. 18 (3), pp. 78–85.

Vercoe, B. and Ellis, D. (1990) Real-time CSound: Software Synthesis with Sensing and Control. In: *ICMC*. Glasgow, Scotland, Oct. 1990. Michigan Publishing. pp. 209–211.

Vlaskovits, P. (2011). *Henry Ford, Innovation, and That "Faster Horse" Quote*. URL: `https://hbr.org/2011/08/henry-ford-never-said-the-fast`.

Walker, R. (2008) *Tunesmithy*. Available From: `http://robertinventor.com/software/tunesmithy/music.htm` [Last Accessed: 15/07/2020].

Wang, G., Cook, P. R., and Salazar, S. (2015) Chuck: A strongly timed computer music language. *Computer Music Journal*. 39 (4), pp. 10–29.

Watershed. (2019) *Composing Music with Humans and Computers*. Available From: `https://www.watershed.co.uk/whatson/10180/composing-music-with-humans-and-computers/?fbclid=IwAR3Ie7FM62VHqaJfuSevefCTF9A8XfwXug_RNXcWjXQeRoumlQKjg4o71Vs#` [Last Accessed: 11/11/2020].

Watson, M. (2018) MuseScore. *Journal of the Musical Arts in Africa*. 15 (1-2), pp. 143–147.

Webster, P. (2002) Historical perspectives on technology and music. *Music Educators Journal*. 89 (1), pp. 38–43.

Wessel, D. and Wright, M. (2002) Problems and prospects for intimate musical control of computers. *Computer music journal*. 26 (3), pp. 11–22.

Wiggins, G., Papadopoulos, G., Phon-Amnuaisuk, S., and Tuson, A. (1998) Evolutionary Methods for Musical Composition. In: *AISB Symposium on Musical Creativity*. Jyväskylä, Findland, 1998. University of Edinburgh.

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

Wooller, R., Brown, A. R., Miranda, E., Diederich, J., and Berry, R. (2005) A Framework for Comparison of Process in Algorithmic Music Systems. In: *Generative Arts Practice 2005 - A Creativity and Cognition Symposium*. Sydney, Australia, May 2005. Creativity and Cognition Studios. pp. 109–124.

Worth, P. and Stepney, S. (2005) Growing music: musical interpretations of L-systems. In: *Proceedings of the Conference on Applications of Evolutionary Computation*. Lausanne, Switzerland, 30-1 March 2005. Springer. pp. 545–550.

Xambó, A., Hornecker, E., Marshall, P., Jordà, S., Dobbyn, C., and Laney, R. (2013) Let's jam the reactable: Peer learning during musical improvisation with a tabletop tangible interface. *ACM Transactions on Computer-Human Interaction (TOCHI)*. 20 (6), pp. 1–34.

Xambó, A., Laney, R., Dobbyn, C., and Jordà, S. (2013) Music and Human-Computer Interaction. In: Holland, S., Wilkie, K., Mulholland, P., and Seago, A. *Video analysis for evaluating music interaction: musical tabletops*. Springer. pp. 241–258.

Xenakis, I. (1971) *Formalized music thought and mathematics in composition*. Stuyvesant NY: Pendragon Press.

Zanzotto, F. M. (2019) Human-in-the-loop Artificial Intelligence. *Journal of Artificial Intelligence Research*. 64 pp. 243–252.

Zulić, H. (2019) How AI can Change/Improve/Influence Music Composition, Performance and Education: Three Case Studies. *INSAM Journal of Contemporary Music, Art and Technology*. 1 (2), pp. 100–114.

Part VI.

APPENDICES

Summary

A: Music Practitioner Questionnaire

Discusses the results of a user survey given to music practitioners. Summarised in chapter 5, but discussed at length here.

B: Surveys

Copies of the 3 surveys used in the user studies discussed in chapters 7, 8 and 10.

C: Existing Music Software Plug-in Summary

An extension of the work summarised in chapter 4. This section discusses the range of computer-generated plug-ins and processes already found in common music software in detail.

D: IGME System Documentation

Details all of the computer-generated processes within IGME.

E: Additional Figures

Additional figures excluded from the main body.

F: IGME Tutorial Sheet

A copy of the tutorial given to participants in the first pilot study.

G: Additional Publications

Publications produced as part of doing this PhD that are not discussed in the main body.

A: Music Practitioner Questionnaire

This chapter originally formed a larger part of the main body of work. Although it was removed due to space constraints, key findings were summarised in Chapter 5. The original work is included here for the interested reader.

1. Introduction

Evaluating existing music systems through simply exploring them in the literature provides only a single-dimensional view of the topics this research aimed to address. Studying the methods used by composers to create music helped to identify workflows that could be automated and roles for which computer-generated music might be useful. An additional aim of the research was to identify the type of composer who would, not only use but benefit from a computer-generated music system.

The methodology employed an online questionnaire (based on work by Nash (2011), which was advertised to the university's music technology cohort as well as through an online composer forum[1]. A range of music students, amateur composers, and professional composers responded. 24 complete responses were gathered.

2. Survey

This section summarises the results for each question. The main findings are given a cursory look, as an in-depth discussion was beyond the scope of this research. To aid the narrative, the questions are not explored consecutively. A template of the questionnaire is supplied at the end of the chapter.

[1] http://www.compositiontoday.com/blog/431.asp

2.1. Participant Background

Table 1 shows the musical background of the participants, with the majority being music technology students. Table 2 summarises the participants' experience of music. Important observations include that 21/24 participants had composed some form of music and 9/24 were professional music practitioners. Results in Figure 1 show that participants have a broad knowledge of musical activities. More notably, just over half stated they had *above average* experience of composing music using a computer.

Figure 2 shows a box and whisker plot for the length of experience each participant had under *'general musicianship', 'composing/songwriting' and 'professional composition'*, with the results showing wide variance. The distribution for *professional composition* is skewed as several participants entered 0 years.

Q1: Which of these best describes your background?

Music Technology Student	Music Student	Recreational Composer	Professional Composer	Other
11	4	2	5	2

Table 1.: The musical background of each participant.

Q2: What current musical experiences do you have? (tick any that apply)

Question	Count	Percetage
I play piano.	13	54.2%
I play guitar.	13	54.2%
I play another acoustic instrument.	13	54.2%
I play several musical instruments.	16	66.7%
I listen to a lot of music.	20	83.3%
I can read music.	16	66.7%
I have had music lessons.	19	79.2%
I have studied music theory (scales, etc.)	21	87.5%
I have performed live	15	62.5%
I have composed music/songs/tunes.	21	87.5%
I practise a lot.	8	33.3%
I am a professional performer.	5	20.8%
I am a professional composer/songwriter.	9	37.5%
I have performed with friends	17	70.8%

Table 2.: The participant's experience of music.

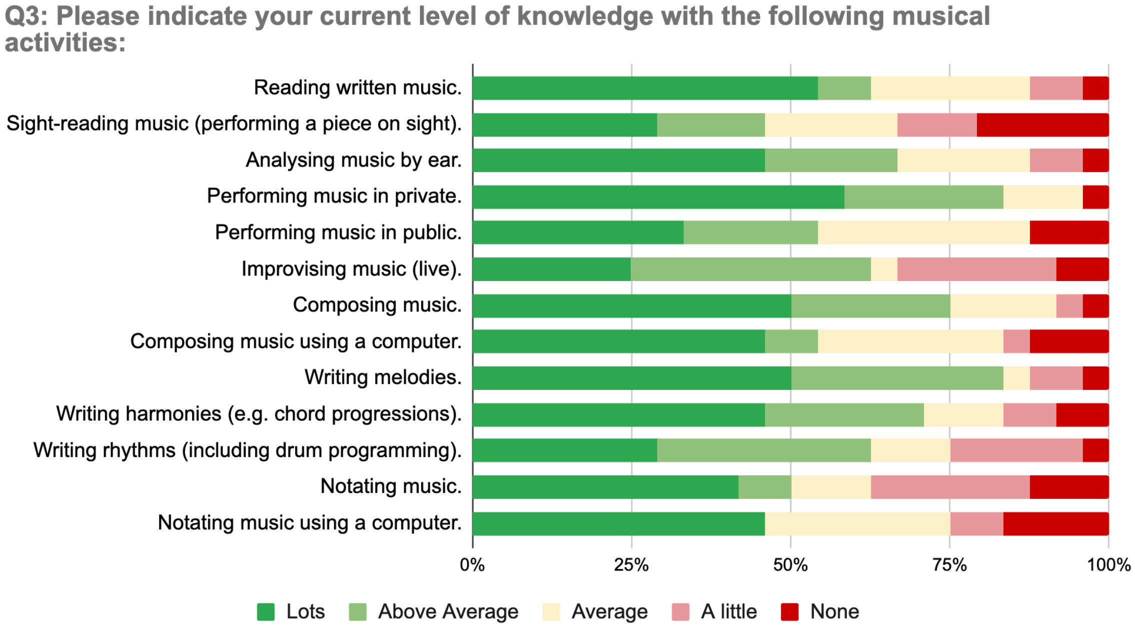


Figure 1.: Distribution of responses for participants' knowledge of musical activities.

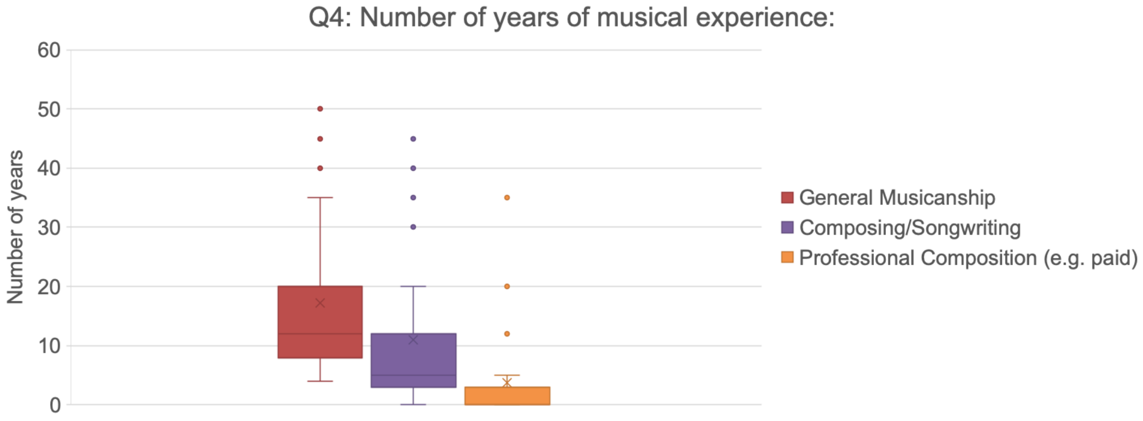


Figure 2.: Box and whisker plot of the number of years of musical experience, by type.

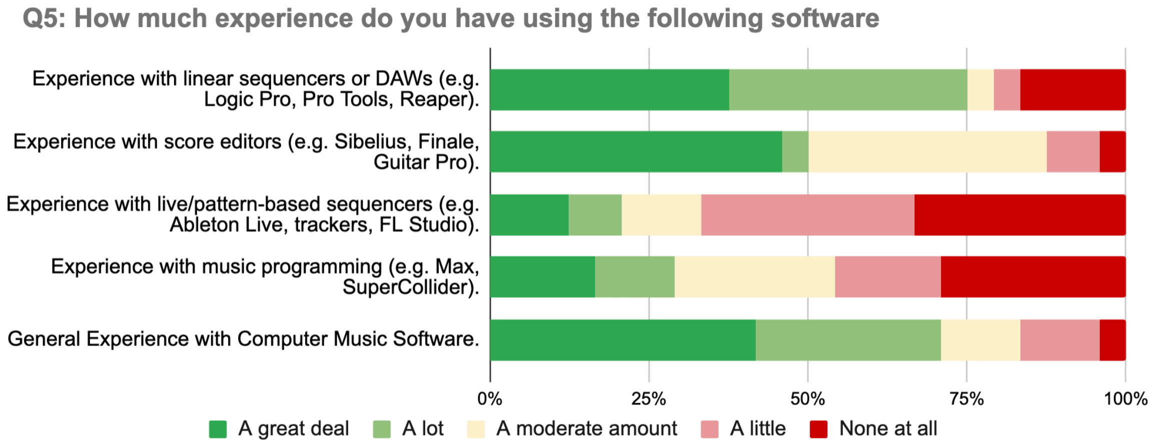


Figure 3.: Summary of results for experience of different types of music software.

2.2. Music Composition Software

Figure 3 shows the distribution of results when asking about experiences of using different types of music software. The results suggest that *'linear sequencers'* and *'score editors'* remain the most commonly used software types. *'pattern-based sequencers'* and *'music programming'* show relatively little use by comparison. Question 6 asked *"what is your preferred composition/DAW package."*. A range of responses was given, with the two most popular being Logic Pro X and Sibelius.

2.3. Music Ideation

Question 7 was *"briefly reflect on how your preferred software allows you to explore new ideas?"*. A recurring theme was the software features that permitted the participant to experiment with different arrangements of their music. One noted that *"Ableton Live also has some great presets, which allow me to sketch out musical ideas before fleshing them out"*. Another stated that music software allows them to experiment with ideas they cannot perform on the piano, and the playback of such sequencers allows them to *proofread*.

Several participants said the workflow afforded by their chosen software permitted ideas to flow easily from their minds to the computer. One noted that strong knowledge of the software's limitations and user interfaces meant it was possible to concentrate on composing.

Question 8 complemented the previous question by asking, *"is there a specific feature in any of the above software that helps you come up with new ideas?"*. Two recurring responses were *'using the inbuilt loops as a source of inspiration'*, and *'using inbuilt audio effects/processes to create new sounds'*.

In a counter-response, one participant stated: *"I don't think Finale especially promotes the formation of new musical ideas"*. Finale (Makemusic, 2018) like Sibelius is primarily a score engraving program, and not designed for 'coming up with ideas'. Similarly, another participant noted there are such *features*, but preferred not to use them.

Question 9 asked *"are there composition techniques that you are interested in, which are not well supported in your current DAW/composition package?"*. Several participants agreed that their needs were already being met. Two wanted to use generative and algorithmic music tools, while another wanted to create 'avant-garde' compositions. In essence, most composers' needs are already being met.

2.4. Workflows and Approaches to Composition

Question 10 asked *"are there specific technique(s) that you use to improve workflow (e.g. work faster, more efficiently)?"*. Many participants said they made use of an external device, such as a MIDI controller, keyboard, or an iPad, to input material. A similar number indicated they used keyboard shortcuts. One participant noted that being able to quickly audition clips, and see how they fitted in with different loop points and effects, improved *workflow* in their choice of software (Ableton Live).

Question 11 asked *"when using any of the above software, are there certain issues that restrict your workflow?"*. More responses were received for this than for the previous question. Several participants complained that inputting notes, either through initial note-by-note input or by cleaning up MIDI data, was often tedious. Similar responses were that the computer could not work very fast and that it was easier to work on paper. One participant observed that score editing software packages are unable to generate core musical structures, such as a chord sequence, during the initial sketching phase of composition. Sketching is an activity afforded by a paper medium (Nash, 2015) and is often more difficult in software.

Question 14 asked *"For each of the following statements, indicate how well the description agrees with your own approach to composing music?"*. Answers are discussed below, in reference to Figure 4.

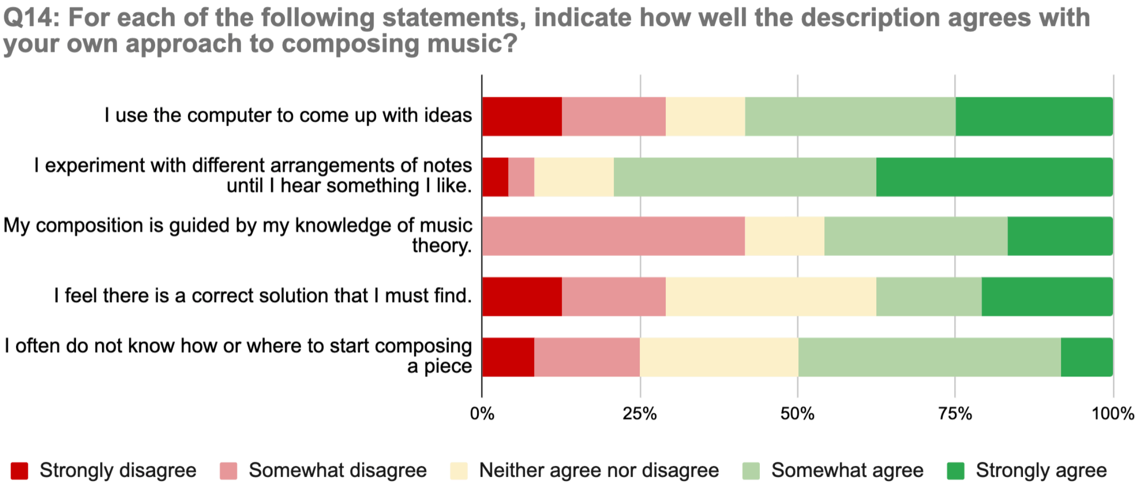


Figure 4.: Approaches to Composition

"I use the computer to come up with ideas."

Most participants *agreed*, suggesting the computer facilitates the user in creating new ideas. Computers can assist in many ways, including auditioning a full orchestra, applying transformations (e.g. retrograde), and generating content. However, solely based on this answer, it is unclear what features of the computer or software facilitate idea generation.

"I experiment with different arrangements of notes until I hear something I like."

The majority of people *agreed* with this, which indicates the benefit of using computer tools that make it easy to go back and make changes and of working with few constraints.

"My composition is guided by my knowledge of music theory."

Slightly more people *disagreed* with this statement than *agreed*, perhaps because strictly following music theory severely limits the scope of music. This possibly suggests that embedding music theory in a way that tries to *correct* compositional choices would be undesirable. Music is as much about respecting the rules as breaking them.

"I feel there is a correct solution that I must find."

This question generated a similar distribution of answers between *agree* and *disagree*. Many algorithms and machine learning techniques can be seen as problem solvers, in that they search through a problem space for a solution. This idea has parallels with music (Alty, 1995). The

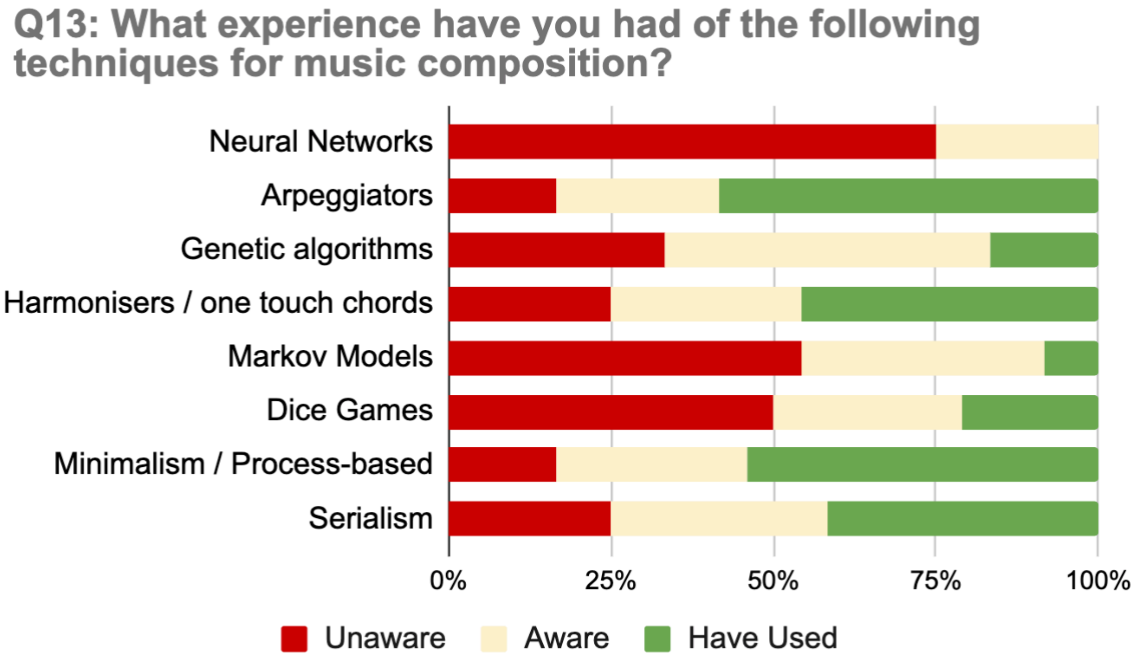


Figure 5.: Experience of composition techniques.

difficulty in finding a correct solution in ordinary music composition is that multiple possibilities are evaluated based on subjective preference.

"I often do not know how or where to start composing a piece."

The majority of participants *agreed* with this statement, indicating that using computer-generated techniques for initial idea generation could be proposed as a solution for aiding composers.

2.5. Techniques for Music Composition

Question 13 asked *"what experience have you had of the following techniques for music composition?"*, with results given in Figure 5. The most *used* technique is the arpeggiator, which along with harmonisers is commonly found on consumer keyboards and features widely in music composition software. Minimalism and serialism techniques also showed frequent *'usage'* and *'awareness'*, perhaps unsurprisingly as these are often taught on music education courses. Neural networks and Markov models have been commonly explored techniques for computer-generated music in the literature, although the majority of participants were *'unaware'* of them and hardly any had *'used'* them. Many survey participants marked that they were *'aware'* of genetic algorithms, and three marked they had *'used'* them.

Q21: If the comptuer was able to suggest musical ideas, would you find this feature:	
Annoying	5/24
Intrusive	9/24
Crucial	0/24
Useful for ’composer’s block’	13/24
Interesting	13/24
Novel	1/24
Inspiring	3/24
None of the above	0/24

Table 3.: Summary of responses for above question.

It could be suggested that the reason composers have not used certain techniques is they are simply unaware of their existence, or because they are satisfied with their existing experiences of music composition. Many of the more complex computer-generated music techniques remain firmly in the academic domain and are absent from existing mainstream music composition software.

Table 3 shows the results of asking *“if the computer was able to suggest musical ideas, would you find this feature...”*, with participants able to make multiple selections. One- third found the suggestion *‘interesting’* and/or *‘useful for composer’s block’*, while roughly the same number found this too *‘intrusive’*. It is evident that there needs to be a clear separation between making such a feature *‘interesting’* to use and not making it *‘intrusive’*. Few people found this to be *‘novel’* or *‘inspiring’* and no one found it to be *‘crucial’*.

Question 15 asked *“would you consider using music the computer has generated in your current music practice?”*. 12 participants answered *‘yes’*, 9 answered *‘it depends’* and 3 answered *‘no’*. Those who answered *‘it depends’*, additionally wrote that they would need to have control over the process and that it would depend on what it sounded like. Another noted they would use computer-aided composition, such as fractals, but would not use AI-generated material. Several stated that it would depend on the context, with one suggesting they would consider using computerisation only for a commission.

Among those who answered ‘no’, one said *“I consider composition exclusively from my own ideas and experience to be a satisfying challenge”*. Another said, *“the emotions in music are not susceptible to computerisation”*. It should be asserted and respected that generative music techniques are simply not of interest to every composer.

Q12: Where does most of your musical creativity take place?	
Entirely at the computer	0
Mostly at the computer	11
Mostly away from the computer	6
Entirely away from the computer	4
Other	3

Table 4.: Where does creativity happen.

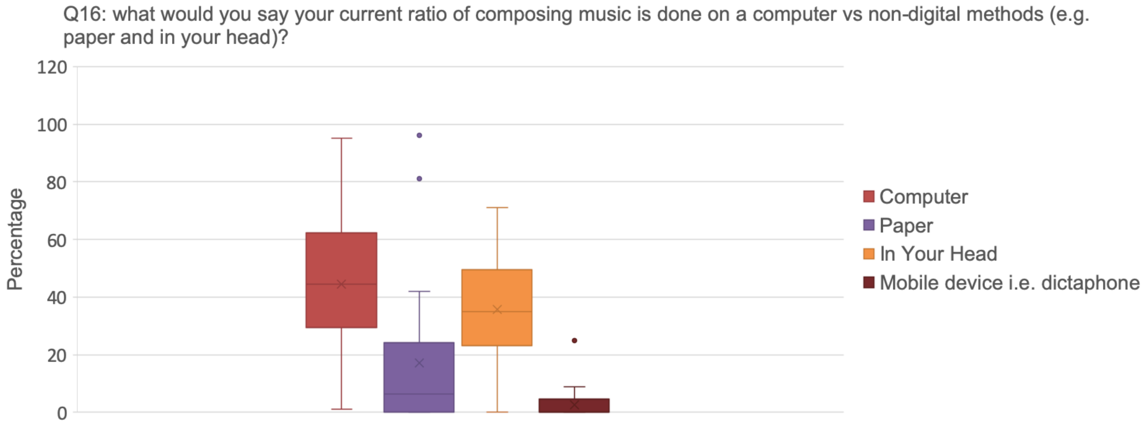


Figure 6.: Ratio of composing music on a computer vs non-digital methods.

2.6. Digital and non-Digital Creativity

Question 12 asked *"where does most of your musical creativity take place?"*. The results, as set out in Table 4, were mixed. One participant stated *"I generally don't have musical ideas when working with a computer, but rather use it as a way to create or discover something that I would never have thought of myself - I like it to surprise me"*. The results show that no participant uses the computer in isolation but makes use of other forms of music practice (i.e. playing an instrument).

Question 16 asked *"what would you say your current ratio of composing music is done on a computer vs non-digital methods"*. Participants manipulated a set of sliders to give a percentage for each, totalling 100[2]. The results in Figure 6 showed wide variance for each category. On average, half of all compositional activity was completed on a computer. The outliers for the paper distribution showed that for two individuals at least 80% of composition was completed on paper. Finally, a large proportion of composition happens in the *'mind'* of the composer.

Question 17 asked: *"when composing, are there particular tasks that you prefer to complete on the computer; and/or on paper; and/or using a different method?"*. Several participants made informal notes on paper, or via a mobile device. Many respondents noted that playing a physical instrument

[2] the user interface ensured that the results totalled exactly 100

Q20: Do you feel music composition software supports or impedes creativity?

Mostly Supports	Somewhat Supports	Neither	Somewhat Impedes	Mostly Impedes
8	13	3	0	0

Table 5.: Opinions on creativity.

is crucial for creating music, something that the computer cannot replicate. Many participants stated that finishing a final score is often more suited to being completed on a computer (instead of paper).

In summary, many participants use instruments to come up with ideas and then use software as the main method for preparing finalised compositions, while using other devices or forms of *'secondary notation'* to capture informal ideas. It is therefore crucial that the computer is not considered the only 'tool' in a composer's toolkit.

Question 19 asked *"how important a role does technology play in your composition process (if at all)?"*. One notable response was: *"without computer-aided techniques, I wouldn't be able to explore many of my ideas in more depth at the speed I can."*, reinforcing the idea that the computer can be used to facilitate creativity. Another response was *"for a lot of people it's the only way people can create the music they hear, and hear the music they create!"*.

Finally, question 20 asked *"do you feel music composition software supports or impedes creativity?"*. The results from this question (Table 5) indicated that computers do support the creative process but that this could be improved further. Nobody found computers to *'impede'* creativity, although a few answered *'neutral'*.

3. Summary and Methodology Review

The questionnaire covered topics relating to composition, software, workflows, creativity, and computer-generated music techniques. The questionnaire was lengthy for an online survey, resulting in several abandoned responses - 60 people started the survey but only 24 finished. The questions were broad and focused on general music practice rather than specifically looking at computer-generated music themes.

One hypothesis from studying the results is that understanding computer-generated processes increases the likelihood of them being considered for use in composition. For example, composers are unlikely to use serialist techniques if they have never been taught them. Therefore, it is suggested that any proposed software should explain techniques (through inbuilt tutorials or documentation) such as a Markov model before interfaces for playing with them are given[3].

3.1. Research Questions

The research identified three areas in which computer-generated music could be used:

- Generating new ideas
- Exploring existing ideas
- Creating automatic accompaniments

Each objective is defined in more detail below and presented with a research question that was later used in the participant studies.

Generating new ideas:

This refers to techniques that can create new material, with the user specifying the parameters of a given generative model.

Research question: *"I would use CGM techniques to help me to come up with new ideas"*.

Exploring existing ideas:

Allow the user to utilise CGM techniques for experimenting with different arrangements and support this in a suitable interface.

Research questions: *"I would use CGM techniques to help me to explore different permutations of my own material"* and *"the interface provided allows me to effortlessly explore new ideas"*.

Creating automatic accompaniments:

Allow the user to create automatic accompaniments alongside their existing music.

Research question: *"I would use the automatic accompaniments as a starting point in my composition practice"*.

[3] This was not explored in depth in the research or user studies and could be an area for future research.

4. Conclusion

To conclude, the following design requirements should be factored into any computer-generated music system, and specifically for the system built for this research. The topics discussed in this chapter informed the requirements listed in Table 6.

Requirement	Description
Workflow	The system should ensure that ideas can be entered quickly, and that informal notes and sketches can be made in appropriate alternative notations.
Education	The system should seek to educate users on generative techniques.
Experiment	The system should use algorithmic techniques to augment existing ideas and to explore alternative arrangements with ease.
Learnability	The system should be familiar in nature to existing music sequencing applications.
Idea Generation	The system should allow the user to use novel generative music techniques for creating new material.
Unobtrusive	A user should be able to use the system firstly as a music sequencer and, if the user so wishes, to have inbuilt tools for experimenting with generative and algorithmic techniques.

Table 6.: Some initial design requirements for building an end-user computer-generated music system.

In summary, the data gathered here was used in conjunction with the background research to inform the initial design requirements for an end-user computer-generated music system. The system could have been designed based on the secondary research detailed in Chapters 3 and 4. However, the design was supplemented and reinforced by the research presented in this chapter.

5. Survey

Listed on the next page is the survey given to participants for the work discussed in this chapter.

Default Question Block

Composer Workflow Study

This survey looks to gain your insight into software based methods that can assist composers. The research looks at how generative music applications can be embedded inside existing systems such as Sequencers, Score Editors and Digital Audio Workstations, and how these can be integrated with existing musical practices.

The questions included in this survey are subjective, there are no right or wrong answers.

Data gathered as part of this questionnaire will be stored in the UK and will be anonymised. It is envisioned that the data will be used for academic publications and theses. You are free to withdraw from the survey at any time until 1st June 2017, after this the data collected will have been collated and published.

If you have any questions prior to taking this survey you can email the following people:

Samuel Hunt - Principle Researcher: Samuel.hunt@uwe.ac.uk.

Dr Chris Nash - Project Supervisor: Chris.Nash@uwe.ac.uk

Information about the project and consent can be downloaded from the following link:
Composer survey information and consent form

Please note consent for the online survey is given in the section bellow.

Please confirm that you give consent to provide the data required by this survey by clicking and highlighting the following three statements.

My participation in this survey is entirely voluntary, I am free to withdraw at any time up until the 1st of June 2017 without reason. If I choose to withdraw any information provided will be

securely discarded.

Given the nature of the questions relating to sometimes personal experiences. I have the right to decline answering any question without reason.

I have downloaded a copy of the Information and consent for participants document, and have read and understood it.

Music

Which of these best describes your background?

What current musical experiences do you have? (tick any that apply)

I play piano.
I play guitar
I play another acoustic instrument
I play several musical instruments.
I listen to a lot of music.
I can read music.
I have had music lessons.
I have studied music theory (scales, etc.)
I have performed live.
I have composed music/songs/tunes.
I practice a lot.
I am a professional performer.
I am a professional composer/songwriter.
I have performed with friends

Please indicate your current level of knowledge with the following musical activities:

	None	...	Some	...	Lots
Reading written music.	○	○	○	○	○
Sight-reading music (performing a piece on sight).	○	○	○	○	○
Analysing music by ear.	○	○	○	○	○
Performing music in private.	○	○	○	○	○
Performing music in public.	○	○	○	○	○
Improvising music (live).	○	○	○	○	○

	None	...	Some	...	Lots
Composing music.	○	○	○	○	○
Composing music using a computer.	○	○	○	○	○
Writing melodies.	○	○	○	○	○
Writing harmonies (e.g. chord progressions).	○	○	○	○	○
Writing rhythms (including drum programming).	○	○	○	○	○
Notating music.	○	○	○	○	○
Notating music using a computer.	○	○	○	○	○

Number of years of musical experience:

General Musicanship	
Composing/Songwriting	
Professional Composition (e.g. paid)	

Software

How much experience do you have using the following software.

	A great deal	A lot	A moderate amount	A little	None at all
Experience with linear sequencers or DAWs (e.g. Logic Pro, Pro Tools, Reaper).	○	○	○	○	○
Experience with score editors (e.g. Sibelius, Finale, Guitar Pro).	○	○	○	○	○
Experience with live/pattern-based sequencers (e.g. Ableton Live, trackers, FL Studio).	○	○	○	○	○

	A great deal	A lot	A moderate amount	A little	None at all
Experience with music programming (e.g. Max, SuperCollider).	○	○	○	○	○
General Experience with Computer Music Software.	○	○	○	○	○

What is your preferred composition/DAW package?

Briefly reflect on how your preferred software allows you to explore new ideas?

Is there a specific feature in any of the above software that helps you come up with new ideas? (Feel free to list as many as you want.)

Are there composition techniques that you are interested in, which are not well supported in your current DAW/composition package? (Feel free to list as many as you want.)

Are there specific technique(s) that you use to improve work flow (e.g. work faster, more efficiently)? (Feel free to list as many as you want.)

When using any of the above software, are there certain issues that restrict your work flow? (Feel free to list as many as you want.)

Where does most of your musical creativity take place?

	Entirely at the computer	Mostly at the computer	Mostly away from the computer	Entirely away from the computer	Other
I generate most of my ideas...	○	○	○	○	○

If you selected other please briefly explain:

Music Composition

What experience have you had of the following techniques for music composition?

	Unaware	Aware	Have used
Neural Networks	○	○	○
Arpeggiators	○	○	○
Genetic algorithms	○	○	○
Harmonisers / one touch chords	○	○	○
Markov Models	○	○	○
Dice Games	○	○	○
Minimalism / Process-based	○	○	○
Serialism	○	○	○

For each of the following statements, indicate how well the description agrees with your own approach to composing music?

	Strongly disagree	Somewhat disagree	Neither agree nor disagree	Somewhat agree	Strongly agree
I use my instrument to experiment with music before notating/entering it into a computer.	○	○	○	○	○
Most of the music I compose is recorded live (audio or MIDI)	○	○	○	○	○
I use the computer to come up with ideas.	○	○	○	○	○
I experiment with different arrangements of notes until I hear something I like.	○	○	○	○	○
I know exactly what I want to write before sitting down at the computer.	○	○	○	○	○
My composition is guided by my knowledge of music theory.	○	○	○	○	○
I feel there is a correct solution that I must find.	○	○	○	○	○
I work a track at a time; producing all the music for one part before moving to the next (e.g. writing a whole melody before adding accompaniment).	○	○	○	○	○
I work a segment at a time; building all tracks simultaneously, working incrementally from the start of the piece to the end.	○	○	○	○	○

	Strongly disagree	Somewhat disagree	Neither agree nor disagree	Somewhat agree	Strongly agree
I rely on how the music sounds to guide my composing/editing.	○	○	○	○	○
I often do not know how or where to start composing a piece.	○	○	○	○	○
I could write a piece of music without listening to it.	○	○	○	○	○

Would you consider using music the computer has generated in your current music practice?

Yes

It depends (please specify):

No (please briefly explain):

Technology

what would you say your current ratio of composing music is done on a computer vs non-digital methods (e.g. paper and in your head)?

	0	10	20	30	40	50	60	70	80	90	100	
Computer												69
Paper												31
In Your Head												0
Mobile device i.e. dictaphone												0
Total:												**100**

When composing, are there particular tasks that you prefer to complete on the computer; and/or on paper; and/or using a different method? Please briefly describe the tasks and methods used.

How important a role does technology play in your composition process (if at all)?

Have you ever forgotten a new musical idea because of the lack of supporting technology? (if so please explain briefly)

Do you feel music composition software supports or impedes creativity?

Mostly Supports

Somewhat Supports

Neither Supports or Impedes

Somewhat Impedes

Mostly Impedes

If the comptuer was able to suggest musical ideas, would you find this feature:

Annoying	Interesting
Intrusive	Novel
Crucial	Inspiring
Useful for 'composer's block'	None of the above

Conclusion

Thank your for your valuable input into this survey. We are also looking for composers who are willing to discuss their composition process and use of technology in more detail. If this is something you would like to be part of, please leave your name and email address.

Contact Details

Name:

Email:

Are there further comments you would like to add, regarding the research or any of the survey questions in general?

Powered by Qualtrics

B: Surveys

1. Survey A: Preliminary Evaluation

This survey was used for the first pilot study, discussed in chapter 7.

IGMSE Questionnaire sample: Document ID:………………

Please answer all questions, there are no right or wrong answers.

Q1. How much experience do you have of using the following software:
(please note only tick one box for each row)

	A great deal	A lot	A moderate amount	A little	None at all
Linear Sequences and DAWs (e.g. Logic, Pro tools)					
Score editors (e.g. Sibelius, finale)					
Live pattern-based sequencers (Ableton, trackers, FL studio)					
Music Programming (e.g. Max, Supercolider)					
General Experience with computer music software					

Q2. What experience have you had of the following techniques for music composition?
(please note only tick one box for each row)

	Unaware	Aware	Have used
Neural Networks			
Arpeggiators			
Genetic Algorithms			
Harmonisers/one touch chords			
Markov Models			
Dice Games			
Minimalism/process based			
Serialism			

Q3. What is you preferred composition/DAW package?

Questions continued overleaf

Q4. What did you find particularly *enjoyable* about using IGMSE?

Q5. What did you find particularly *frustrating* about using IGMSE?

Q6. What part(s) of IGMSE could be improved (not including the synthesised audio quality)?

Q7. Would you be interested in using a similar tool again in the future? (if so please leave email)

Optional email:

Q8. Do you have additional comments?

Thank you again for your participation in this workshop.
By Samuel Hunt, 2017

2. Survey B-3: Cognitive Dimensions Evaluation

This survey was used for the second pilot study in chapter 7, and the studies in chapter 9 and 10.

Default Question Block

Having now spent some time using IGME please fill out the following survey. It should take between 5 - 10 minutes to complete

Please enter your email address

Please enter your product key (optional)

Have you used generative music techniques before this project?

○ Yes
○ No

Briefly in your own words describe your musical background:

Question 1

Version Control System

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
I feel that the version control system encourages me to experiment with ideas.	○	○	○	○	○
I feel that the version control system allows me to easily compare different iterations.	○	○	○	○	○
I feel that the version control system allows me to check my progress	○	○	○	○	○
Using the version control system makes it easy to go back and make changes to the music	○	○	○	○	○

Question 2

Two stage editing process

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The two stage editing process allows me to rapidly enter ideas.	○	○	○	○	○
The two stage editing process allows me to make edits in any order.	○	○	○	○	○
The two stage editing process makes it easy to go back and make changes to the music.	○	○	○	○	○
I feel that the two stage editing process creates hidden dependencies.	○	○	○	○	○

Question 3

Generative effects

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The generative effects in IGME helped me to come up with new ideas.	○	○	○	○	○
I find that the generative effects make annoying mistakes.	○	○	○	○	○
I would like more control over the generative effects	○	○	○	○	○
I would like to be able to define my own generative processes.	○	○	○	○	○
Using generative music/algorithmic techniques helped me to come up with ideas that I would not otherwise have created on my own	○	○	○	○	○

Question 4

Explicit parts

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
I feel that it is easy to distinguish between parts that are computer generated and ones that are purely my own content	○	○	○	○	○
Is it easy to find the type of part I am looking for	○	○	○	○	○
Breaking the music into parts makes it easy to try out new ideas	○	○	○	○	○

Question 5

Dependency arrows **(optional)**

Please answer the questions bellow if you had chance to use reference parts, if you did not please leave this section blank

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The dependency arrows make the relationships between each part explicit.	○	○	○	○	○

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
It was clear that editing a part would have knock on consequences for other parts	○	○	○	○	○

Block 6

IGME system questions

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
When writing music within IGME, there were difficult things to work out in my head	○	○	○	○	○
I feel that I have suitable control over the generative processes in IGME	○	○	○	○	○
I feel that the computer has taken some control of the composition process in IGME	○	○	○	○	○
I feel that my knowledge of generative music has improved since using IGME.	○	○	○	○	○

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The interface provided by IGME allows me to easily explore new ideas.	○	○	○	○	○
IGME's workflow is similar to other score editors and sequencers	○	○	○	○	○
The tutorial system inside IGME helped me to learn the software quickly	○	○	○	○	○

Which feature of IGME would you most like to see in your usual music sequencing application.

Block 7

General

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
I would use generative/algorithmic techniques to help me to come up with new ideas.	○	○	○	○	○
I would use generative/algorithmic techniques to help me to explore different permutations of my own material.	○	○	○	○	○
I would use accompaniments created by the computer alongside my own composed music.	○	○	○	○	○
In general I would use music the computer has generated or suggested in my own compositions.	○	○	○	○	○
Using generative/algorithmic techniques helps to increase my productivity	○	○	○	○	○

Block 8

Creativity

	All of it to the computer	most of it to the computer	neutral	Most of it to myself	All of it to myself
Having used IGME, how much of the musical creativity do you attribute to the computer	○	○	○	○	○

Is the authorship of your creative output a concern when using generative techniques?

○ Yes

○ No

Feel free to explain further

What is the maximum percent of **automated** creativity you would tolerate?

Me only 50:50 Fully the computer

0 10 20 30 40 50 60 70 80 90 100

Amount

Block 10

I found learning IGME using the provided materials (tutorials, presentation and video):

○ Very Easy

○ Somewhat Easy

○ Neutral

○ Somewhat Challenging

○ Very Challenging

I feel that with time I could master IGME

○ Strongly agree

○ Somewhat agree

○ Neither agree nor disagree

○ Somewhat disagree

○ Strongly disagree

Block 9

What are the positive aspects of using generative music?

What are the negative aspects of using generative music?

Is there further comments you would like to make?

Powered by Qualtrics

3. Survey C: Prior Experiences of Music

This survey was given to participants before starting the longitudinal study in chapter 10.

Default Question Block

Composer Workflow Study

This survey looks to gain your insight into software based methods that can assist composers. The research looks at how generative music applications can be embedded inside existing systems such as Sequencers, Score Editors and Digital Audio Workstations, and how these can be integrated with existing musical practices.

The questions included in this survey are subjective, there are no right or wrong answers.

Data gathered as part of this questionnaire will be stored in the UK and will be anonymised. It is envisioned that the data will be used for academic publications and theses. You are free to withdraw from the survey at any time until 1st June 2020, after this the data collected will have been collated and published.

If you have any questions prior to taking this survey you can email the following people:

Samuel Hunt - Principle Researcher: Samuel.hunt@uwe.ac.uk.

Dr Chris Nash - Project Supervisor: Chris.Nash@uwe.ac.uk

Information about the project and consent can be downloaded from the following link:
Composer survey information and consent form

Please note consent for the online survey is given in the section bellow.

Please confirm that you give consent to provide the data required by this survey by clicking and highlighting the following three statements.

- ☐ My participation in this survey is entirely voluntary, I am free to withdraw at any time up until the 1st of June 2020 without reason. If I choose to withdraw any information provided will be securely discarded.
- ☐ Given the nature of the questions relating to sometimes personal experiences. I have the right to decline answering any question without reason.
- ☐ I have downloaded a copy of the Information and consent for participants document, and have read and understood it.

Music

Which of these best describes your background?

What current musical experiences do you have? (tick any that apply)

- ☐ I play piano.
- ☐ I play guitar
- ☐ I play another acoustic instrument
- ☐ I play several musical instruments.
- ☐ I listen to a lot of music.
- ☐ I can read music.
- ☐ I have had music lessons.
- ☐ I have studied music theory (scales, etc.)
- ☐ I have performed live.
- ☐ I have composed music/songs/tunes.
- ☐ I practice a lot.
- ☐ I am a professional performer.
- ☐ I am a professional composer/songwriter.
- ☐ I have performed with friends

Please indicate your current level of knowledge with the following musical activities:

	A great deal	A lot	A moderate amount	A little	None at all
Reading written music.	○	○	○	○	○
Sight-reading music (performing a piece on sight).	○	○	○	○	○
Analysing music by ear.	○	○	○	○	○
Performing music in private.	○	○	○	○	○
Performing music in public.	○	○	○	○	○
Improvising music (live).	○	○	○	○	○
Composing music.	○	○	○	○	○
Composing music using a computer.	○	○	○	○	○
Writing melodies.	○	○	○	○	○

	A great deal	A lot	A moderate amount	A little	None at all
Writing harmonies (e.g. chord progressions).	○	○	○	○	○
Writing rhythms (including drum programming).	○	○	○	○	○
Notating music.	○	○	○	○	○
Notating music using a computer.	○	○	○	○	○

Number of years of musical experience:

General Musicanship	
Composing/Songwriting	
Professional Composition (e.g. paid)	

Software

How much experience do you have using the following software.

	A great deal	A lot	A moderate amount	A little	None at all
Experience with linear sequencers or DAWs (e.g. Logic Pro, Pro Tools, Reaper).	○	○	○	○	○
Experience with score editors (e.g. Sibelius, Finale, Guitar Pro).	○	○	○	○	○
Experience with live/pattern-based sequencers (e.g. Ableton Live, trackers, FL Studio).	○	○	○	○	○
Experience with music programming (e.g. Max, SuperCollider).	○	○	○	○	○

	A great deal	A lot	A moderate amount	A little	None at all
General Experience with Computer Music Software.	○	○	○	○	○

What is your preferred composition/DAW package?

Are there composition techniques that you are interested in, which are not well supported in your current DAW/composition package? (Feel free to list as many as you want.)

Where does most of your musical creativity take place?

	Entirely at the computer	Mostly at the computer	Mostly away from the computer	Entirely away from the computer	Other
I generate most of my ideas...	○	○	○	○	○

If you selected other please briefly explain:

Music Composition

What experience have you had of the following techniques for music composition?

	Unaware	Aware	Have used
Neural Networks	○	○	○
Arpeggiators	○	○	○
Genetic algorithms	○	○	○

	Unaware	Aware	Have used
Harmonisers / one touch chords	○	○	○
Markov Models	○	○	○
Dice Games	○	○	○
Minimalism / Process-based	○	○	○
Serialism	○	○	○

For each of the following statements, indicate how well the description agrees with your own approach to composing music?

	Strongly disagree	Somewhat disagree	Neither agree nor disagree	Somewhat agree	Strongly agree
I use my instrument to experiment with music before notating/entering it into a computer.	○	○	○	○	○
Most of the music I compose is recorded live (audio or MIDI)	○	○	○	○	○
I use the computer to come up with ideas.	○	○	○	○	○
I experiment with different arrangements of notes until I hear something I like.	○	○	○	○	○
I know exactly what I want to write before sitting down at the computer.	○	○	○	○	○
My composition is guided by my knowledge of music theory.	○	○	○	○	○
I feel there is a correct solution that I must find.	○	○	○	○	○

	Strongly disagree	Somewhat disagree	Neither agree nor disagree	Somewhat agree	Strongly agree
I work a track at a time; producing all the music for one part before moving to the next (e.g. writing a whole melody before adding accompaniment).	○	○	○	○	○
I work a segment at a time; building all tracks simultaneously, working incrementally from the start of the piece to the end.	○	○	○	○	○
I rely on how the music sounds to guide my composing/editing.	○	○	○	○	○
I often do not know how or where to start composing a piece.	○	○	○	○	○
I could write a piece of music without listening to it.	○	○	○	○	○

Would you consider using music the computer has generated in your current music practice?

○ Yes

○ ▭ It depends (please specify):

○ ▭ No (please briefly explain):

Technology

what would you say your current ratio of composing music is done on a computer vs non-digital methods (e.g. paper and in your head)?

0 10 20 30 40 50 60 70 80 90 100

	0 10 20 30 40 50 60 70 80 90 100	
Computer		[illegible]
Paper		[illegible]
In Your Head		[illegible]
Mobile device i.e. dictaphone		[illegible]
Total:		**100**

When composing, are there particular tasks that you prefer to complete on the computer; and/or on paper; and/or using a different method? Please briefly describe the tasks and methods used.

Do you feel music composition software supports or impedes creativity?

○ Mostly Supports
○ Somewhat Supports
○ Neither Supports or Impedes
○ Somewhat Impedes
○ Mostly Impedes

If the comptuer was able to suggest musical ideas, would you find this feature:

□ Annoying
□ Intrusive
□ Crucial
□ Useful for 'composer's block'
□ Interesting
□ Novel
□ Inspiring
□ None of the above

Conclusion

Contact Details

Name:

Email:

Are there further comments you would like to add, regarding the research or any of the survey questions in general?

Powered by Qualtrics

C: Existing Music Software Plug-in Summary

1. Computer-Generated Music Techniques in Popular Music Software

This work is an extension of the research presented in Chapter 4.

Name	Summarised	Reference	Version
Ableton Live	Yes	(Ableton, 2020)	9
ACID Pro	No	(MAGIX, 2020)	10
Ardour	No	(Davis, 2020)	5.12
Audiotool	Yes	(AudioTool, 2020)	2020
Bitwig Studio	Yes	(Bitwig, 2020)	3.1.2
Cubase	Yes	(Steinberg, 2020a)	10.5
Digital Performer	Yes	(MOTU, 2020)	DP10
Dorico	Yes	(Steinberg, 2020b)	3.1
FL Studio	Yes	(Image-Line, 2020)	20
GarageBand	Yes	(Apple, 2020a)	10.3.4
Guitar Pro	Yes	(Arobas-Music, 2020)	6/7
Logic Pro X	Yes	(Apple, 2020b)	10
Numerolgy	Yes	(Five12, 2020)	3
Pro Tools	No	(Avid, 2020a)	2020
REAPER	Yes	(Cockos, 2020)	6.07
Reason	Yes	(Reason-Studios, 2020)	10.4
Renoise	Yes	(Renoise, 2020)	3.2
Sibelius	Yes	(Avid, 2020b)	2020
Studio One	Yes	(Electronics, 2020)	4.6
Tracktion	Yes	(Tracktion-Software-Corporation, 2020)	10

Table 1.: List of popular music composition software and digital audio workstations that were analysed in Chapter 4. (Reproduced from Chapter 4 Table 1).

Chords

Many pieces of software contain a chord effect processor, which is a MIDI process that produces more than one output note for a given input note. Designs vary, with some providing the same mapping for any incoming note (Live and Bitwig) while others, such as Logic Pro and Cubase (Figure 3), allow a user to assign a different chord for each note on the keyboard. Reason's scales

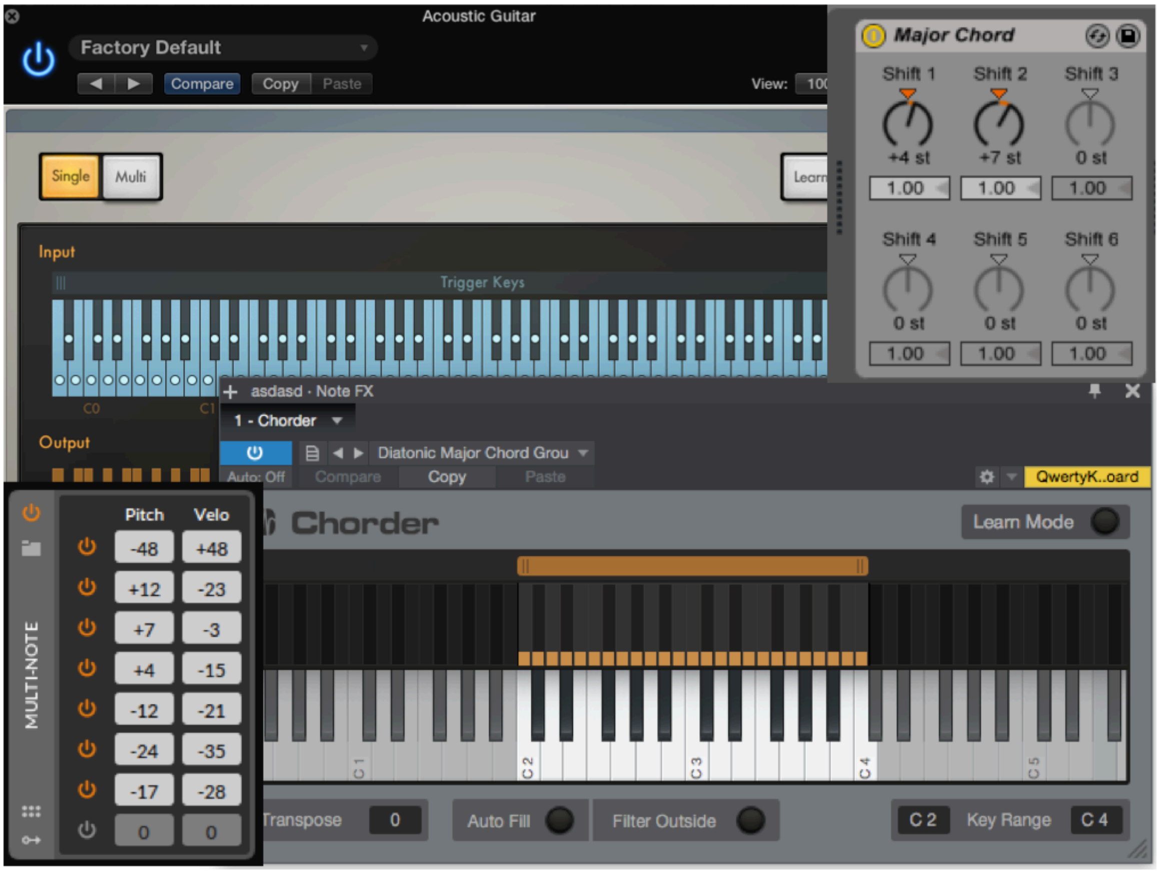


Figure 1.: A collection of various chord effects (Bitwig, Live, Logic and Presonus.

and chords plug-in (Figure 2) maps a single keypress to a chord but is selected from a scale so remains in key.

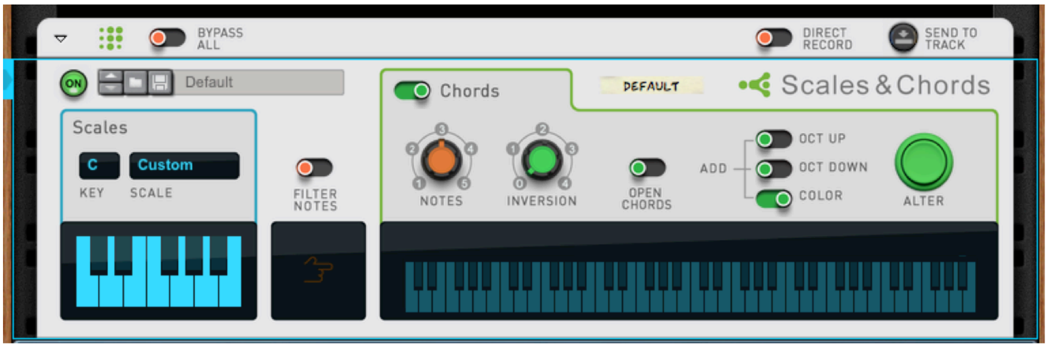


Figure 2.: Reason's chord and scale plugin.

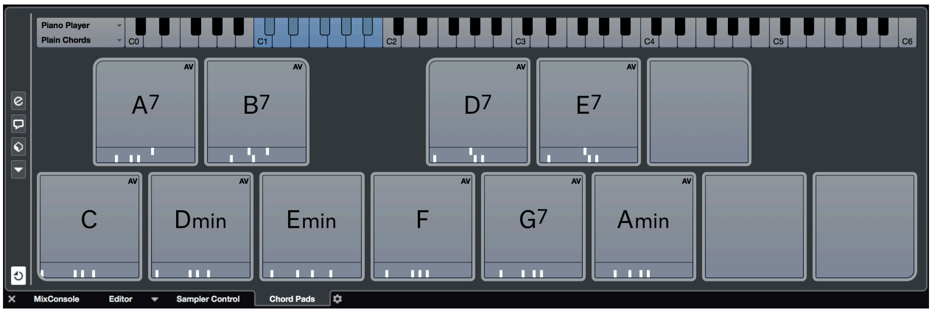


Figure 3.: Cubase chord pad.

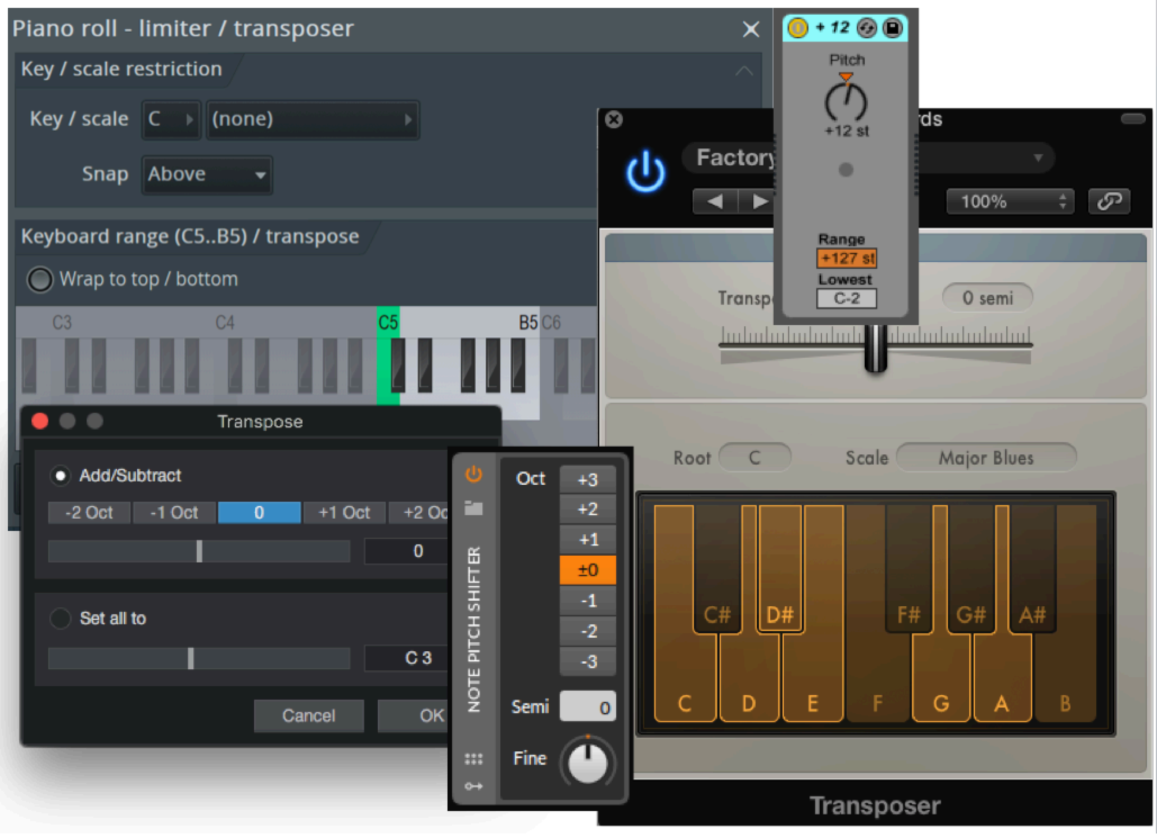


Figure 4.: Collection of various transposers (Bitwig, Live, Logic, Presonus and Studio one).

Transposer

A transposer (Figure 4 is a simple deterministic plug-in that transposes the incoming notes by a set amount. More complex variants include inbuilt limiters and scale quantisers. As with most plug-ins, the controls can be automated in real-time.

Arpeggiator

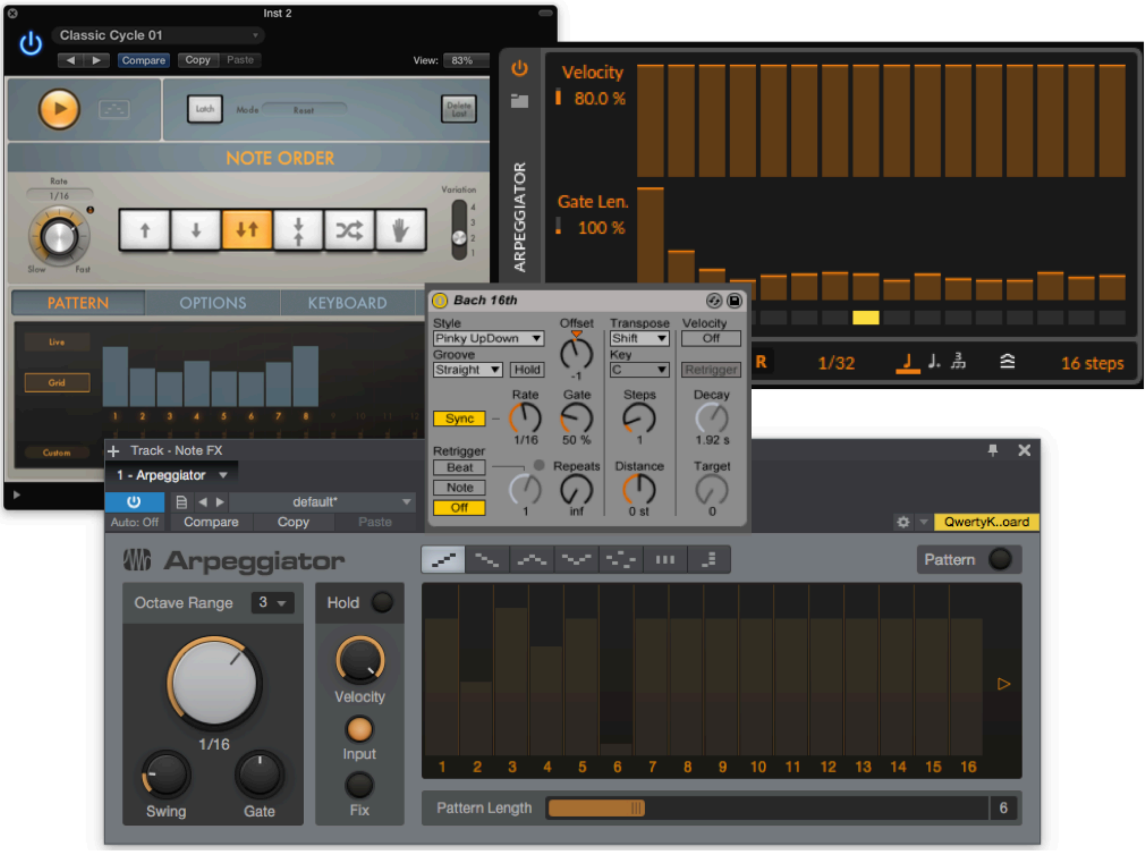
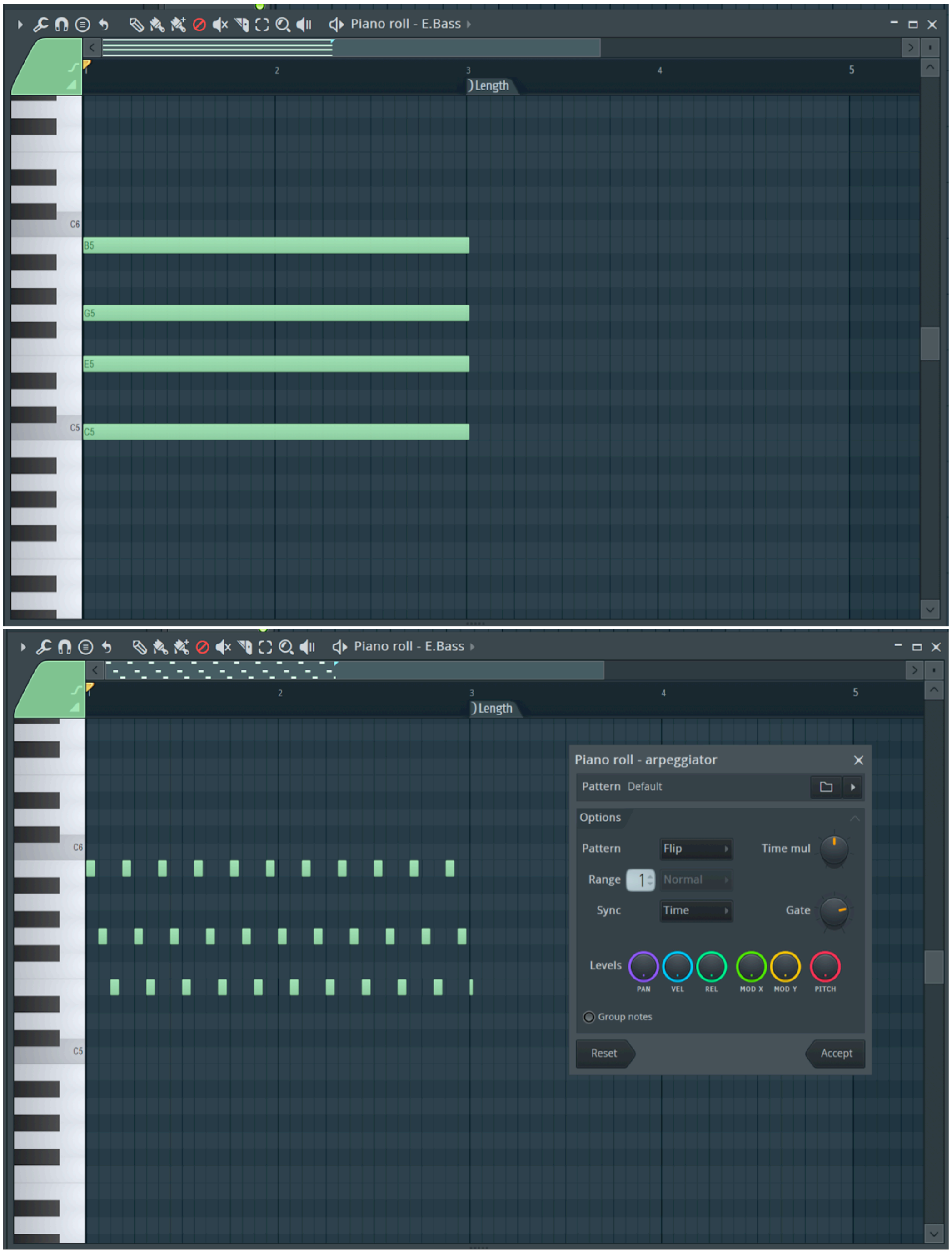


Figure 5.: Collection of various arpeggiators (Bitwig, Live, Logic and Presonus).

An arpeggiator is a process in which notes are rapidly output in a pattern. The ordering of input notes does not necessarily determine the output order. For example, the arpeggiator may sequence notes in ascending order. A pattern can be repeated over a given number of octaves. The effect can be both deterministic and stochastic depending on the configuration. Whereas most arpeggiator effects work in real-time (that is the output is heard audibly rather the being placed back on the timeline), FL Studio works offline by arpeggiating the notes in the editor. Once the effect is computed, the output is put back on the timeline. The effect is destructive once the user hits 'confirm' and the initial notes and settings are lost. After opening the arpeggiator settings, further effects can be applied, essentially working recursively. Figure 6 shows the input notes, and output after the arpeggiator has been computed.



Piano roll - arpeggiator

Pattern Default

Options

Pattern Flip

Time mul

Range 1 Normal

Sync Time

Gate

Levels PAN VEL REL MOD X MOD Y PITCH

○ Group notes

Reset Accept

Figure 6.: FL studio before (top), and after (bottom).

Note Randomiser

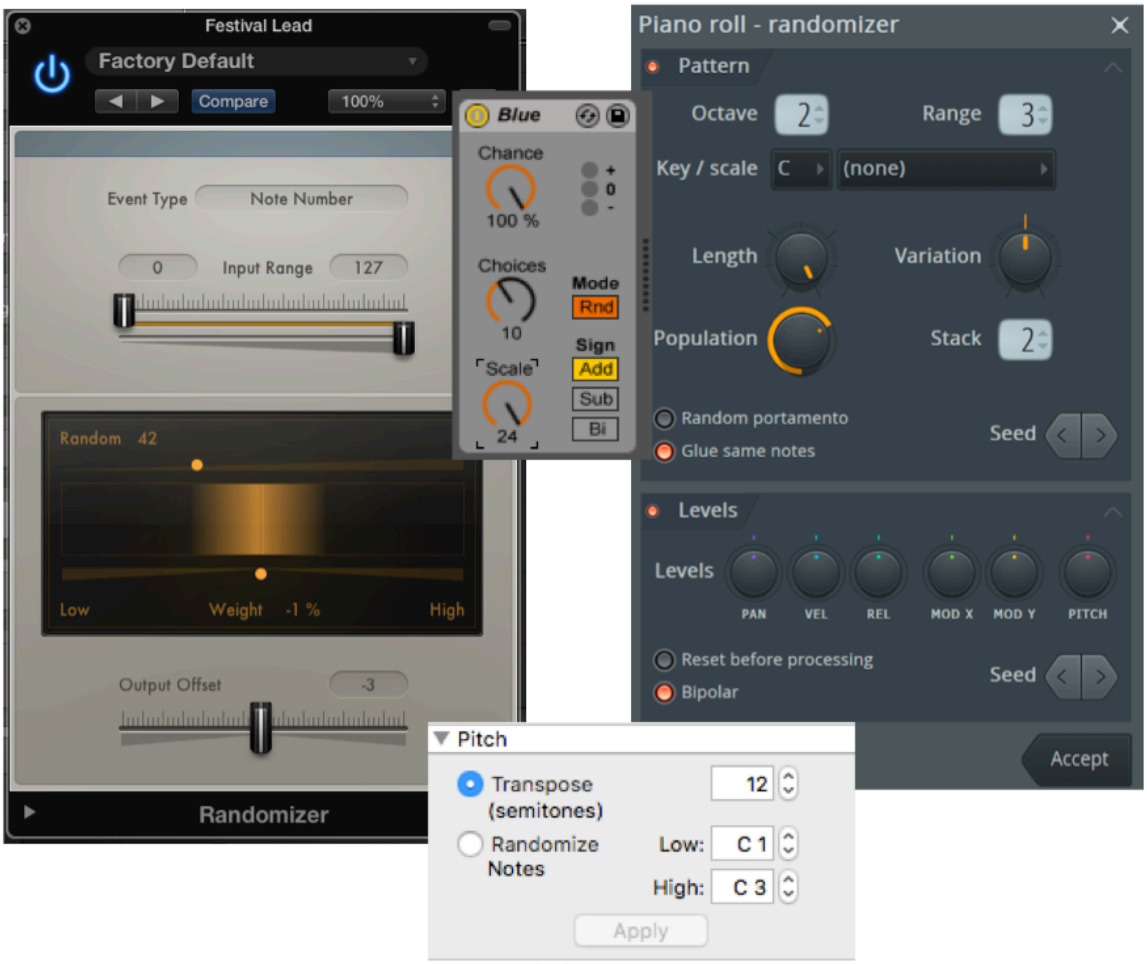


Figure 7.: Collection of note randomiser effects.

A note randomiser is an effect that randomises elements of a sequence of notes. Some variants work in real-time, modifying a note as it is played live, whereas offline tools and editors can randomise the contents of a clip. Some of these will generate new patterns, while others will shuffle an existing sequence. Without the ability to quantise to a scale, the result can be unusable. The top of Figure 8 shows the before and after of applying Reason's note randomiser to a MIDI clip. The bottom of Figure 8 is similar but shows the *'alter notes'* effect, which uses only notes in the original sequence, keeping a sense of consistency. Studio One contains a novel "thin out notes" process that can simplify a sequence, or delete notes randomly from a sequence, as well as a more complex note randomiser (Figure 9) with a range of configuration options.

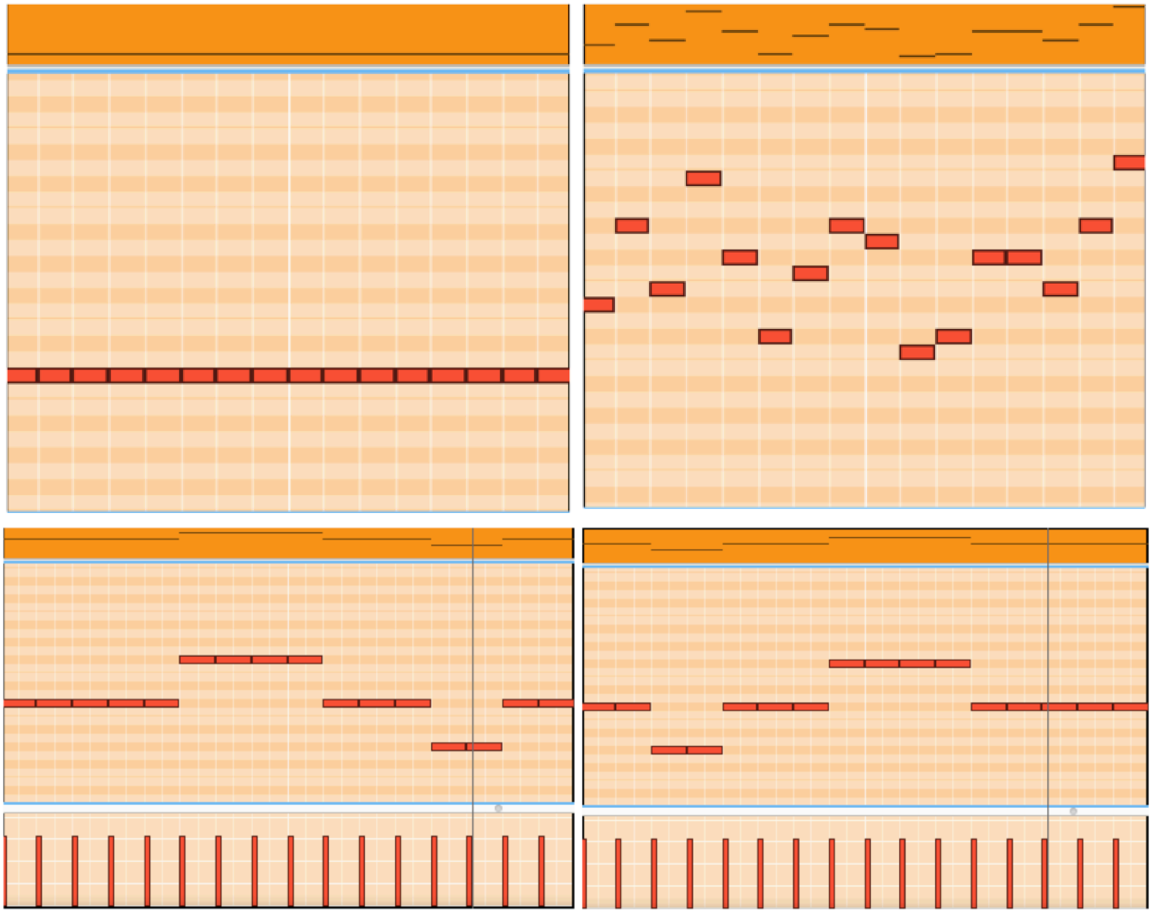
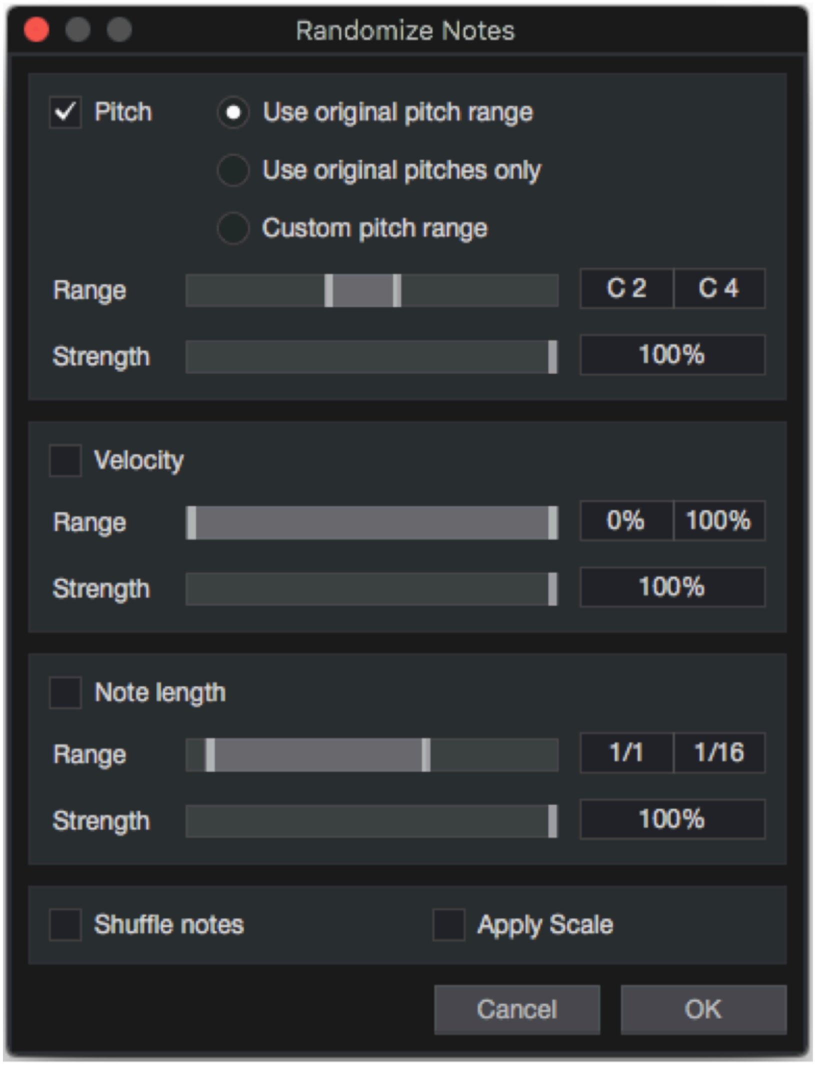


Figure 8.: Applying a note randomiser (top) and note alter effect (bottom) in Reason.



Randomize Notes

☑ Pitch

● Use original pitch range

○ Use original pitches only

○ Custom pitch range

Range C 2 C 4

Strength 100%

□ Velocity

Range 0% 100%

Strength 100%

□ Note length

Range 1/1 1/16

Strength 100%

□ Shuffle notes □ Apply Scale

Figure 9.: Studio One’s note randomiser.

Scale Quantisers

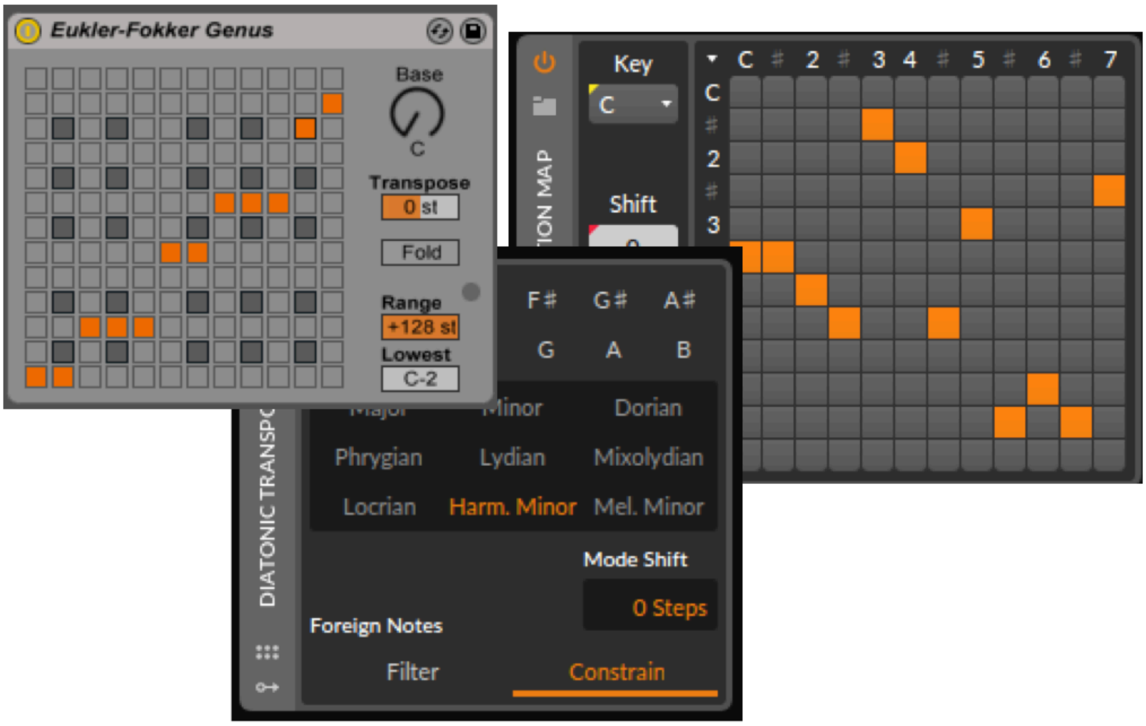


Figure 10.: Various scale quantisers (Bitwig and Live).

A scale effect essentially maps one incoming note number to another, ensuring all notes are in a scale or key. Quantisers are often embedded inside other plug-ins and effects, but standalone ones are available too. The grid-based approaches offered in Bitwig and Live (Figure 10) allow for unorthodox non-linear mappings. When coupled with note randomisers, these two effects allow for more sensible forms of music to be generated.

Note Repeaters

A note repeater (or echo) effect rapidly repeats incoming notes. The velocity of each repeat can be specified, creating different rhythmic effects. Reason's inbuilt note echo (Figure 12) is similar to a step sequencer. Patterns can be applied so that each successive note increases in pitch, or repeats a non-linear pattern.

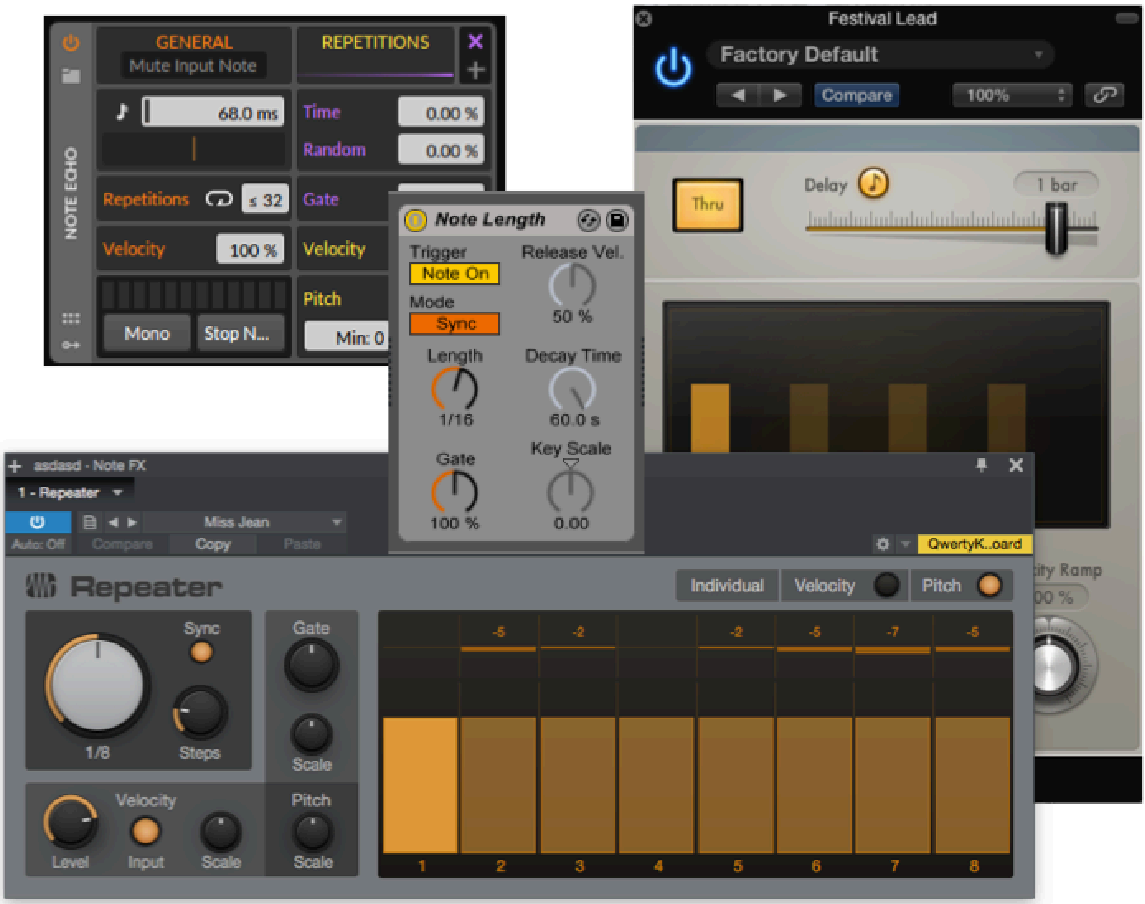


Figure 11.: A selection of note repeater plug-ins (Bitwig and Live).

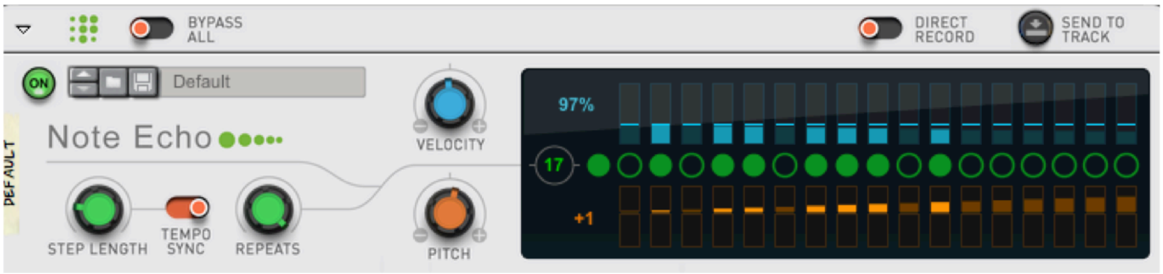


Figure 12.: Reason's powerful note repeater (echo) plug-in.

D: IGME System Documentation

1. IGME's Computer-Generated Processes

This section details each of IGME's computer-generated processes, as summarised in Table 1. Type A is the note properties model. Type B contains the regular plug-ins and type C is the seed generators. Complexity is arbitrarily defined as how complex the process is deemed to use.

Group	Plugin name	Stochastic or deterministic	Complexity (1-5)	Type	Number of sliders	Number of drop-down	Number of checkboxes	Other Controls
B	Transpose	Deterministic	1	Transformation	1	0	0	
A	Note Properties	Stochastic	2	Generative	5 (per note)	0	0	
B	Pitch Quantise	Deterministic	2	Transformation	2	0	0	
B	Transforms	Deterministic	2	Transformation	0	0	4	
B	Note Map	Deterministic	2	Transformation	0	0	0	Many 1-1 mappings
B	Subtractor	Stochastic	2	Transformation	1	0	0	
B	Wind Chime	Stochastic	2	Generative	1	0	0	
B	Repeater	Deterministic	2	Transformation	1	0	1	
B	Constrainer	Deterministic	2	Transformation	2	0	0	
B	Rhythm Quantiser	Deterministic	2	Transformation	0	2	0	
B	Evolution	Stochastic	3	Generative	2	3	0	
B	Arpeggiator	both	3	Transformation	2	2	0	
C	Random Note generator	Stochastic	3	Generative	10	3	1	
C	Perlin Noise Generator	Stochastic	3	Generative	12	3	1	
C	Distribution sample	Stochastic	4	Analytic	31	3	0	Presets and automatic analysis
C	Transition table	Stochastic	4	Analytic	181	1	0	Automatic analysis
C	L-System	Deterministic	5	Generative	1	1	0	Supports the writing of small passages of music, and the rules of a formal grammar

Table 1.: Summary of the the CGM processes/plug-ins included in IGME.

Note Properties Model

The note properties effect (Figures 1 and 2) is used to apply one of 3 effects to each note in the editor and works only for human-computer part types. Applying an effect to a human part implicitly converts it to a human-computer part.

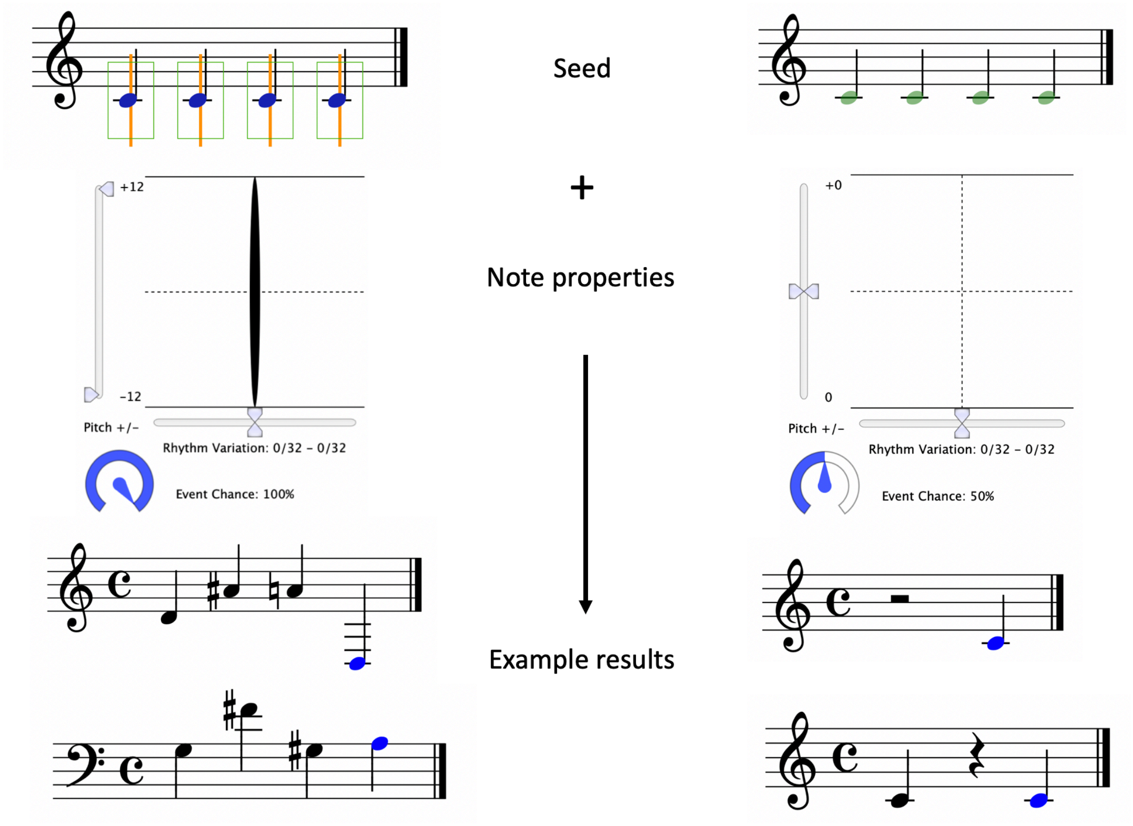
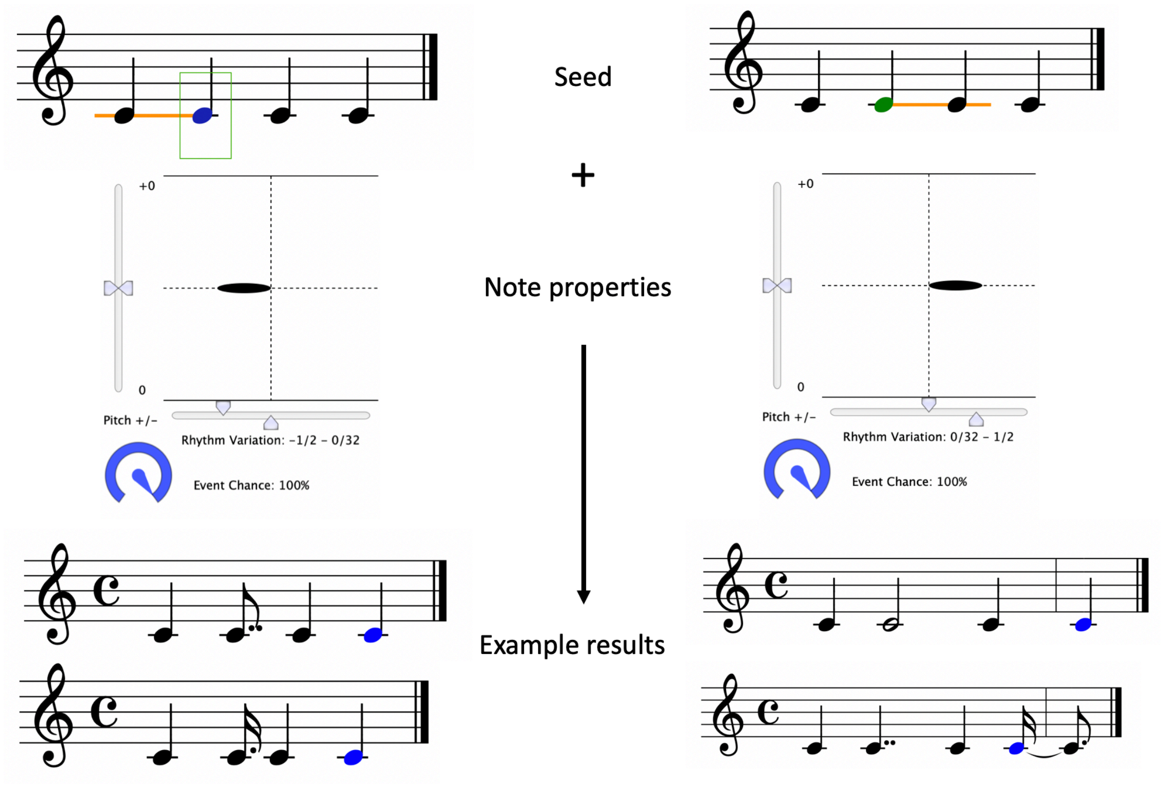


Figure 1.: Note properties showing pitch variation, and chance.

The pitch control (y-axis) controls the pitch randomisation and ranges from -12 to +12. This means that, for each note, the pitch can be transposed up or down. Values are limited to the range 0-127 (same as MIDI).

The chance control (rotary control) applies a probability to a select note, that determines if the note will be in the output sequence (result). By default, this is set to 100%. Notes that are removed from the sequence are replaced by rests, to which chance can also be applied. If the rests are removed, the remaining sequence will be shortened.

The rhythm control (x-axis) changes the individual note's length. This has a range of a single whole note in either direction and is split into 1/32 divisions. Setting the values to between -1/2 and +1/2 would randomly modify the length of the note either by reducing it by up to -1/2 or by increasing it by + 1/2. Should the note length change, so will the notes in the following sequence. Should a note be reduced by an amount greater than its current length, the note would simply be removed.



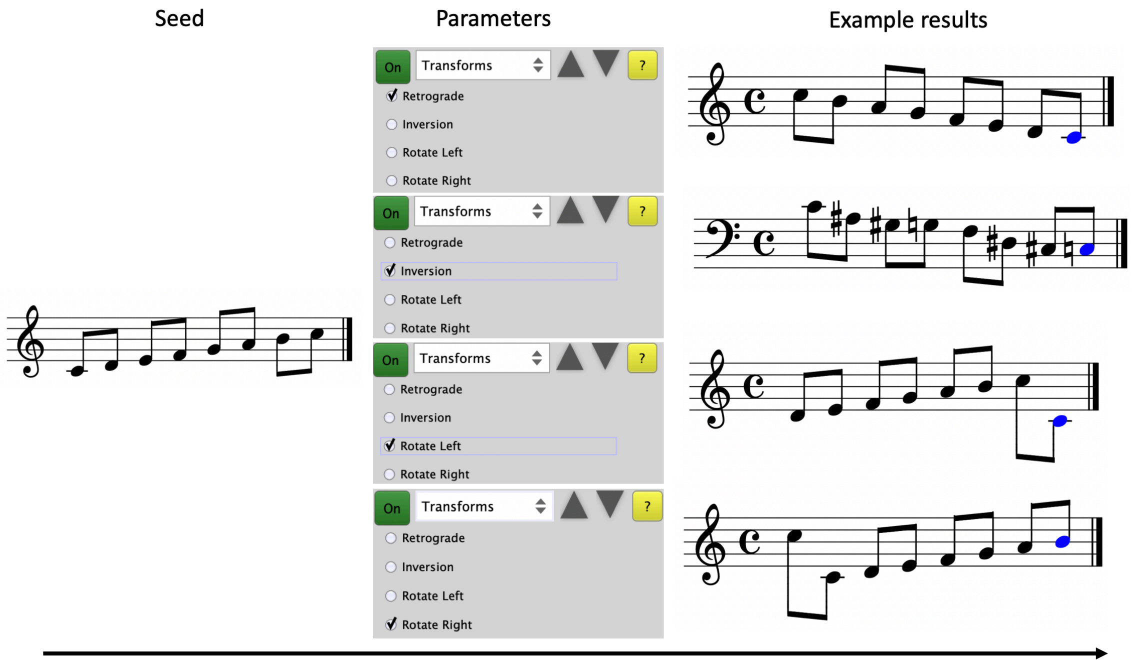
Rhythm Variation: −1/2 – 0/32

Event Chance: 100%

Rhythm Variation: 0/32 – 1/2

Event Chance: 100%

Figure 2.: Note properties showing rhythm variation.



On Transforms ? ☑ Retrograde ○ Inversion ○ Rotate Left ○ Rotate Right

On Transforms ? ○ Retrograde ☑ Inversion ○ Rotate Left ○ Rotate Right

On Transforms ? ○ Retrograde ○ Inversion ☑ Rotate Left ○ Rotate Right

On Transforms ? ○ Retrograde ○ Inversion ○ Rotate Left ☑ Rotate Right

Figure 3.: Transforms plug-in.

Transforms

The transforms plug-in (Figure 3) applies one of 4 processes to the input sequence. Retrograde reverses the sequence (i.e. plays it backwards). Inversion inverts the sequence around its origin. Rotate left takes the first note in the sequence and places it at the end, whereas rotate right does the opposite. The order of operations is prescribed as retrograde, inversion, rotate left, and rotate right. A user who wanted to rotate right and then retrograde would need to add two instances of this plug-in.

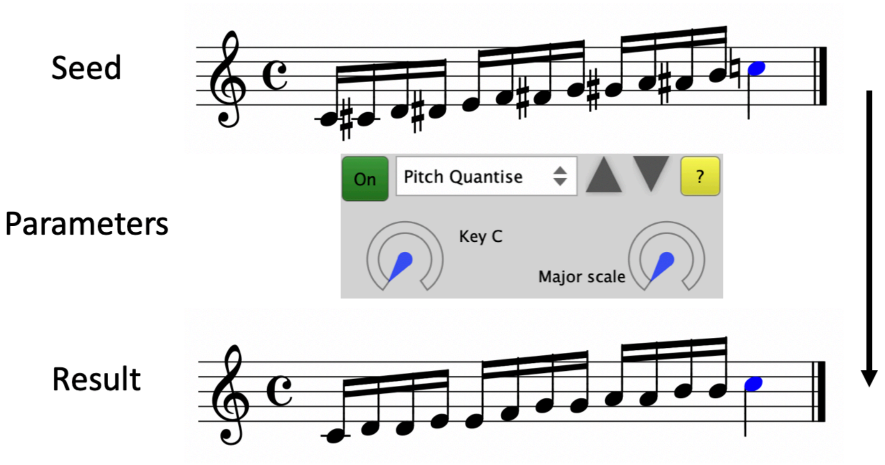


Figure 4.: Pitch quantiser plug-in.

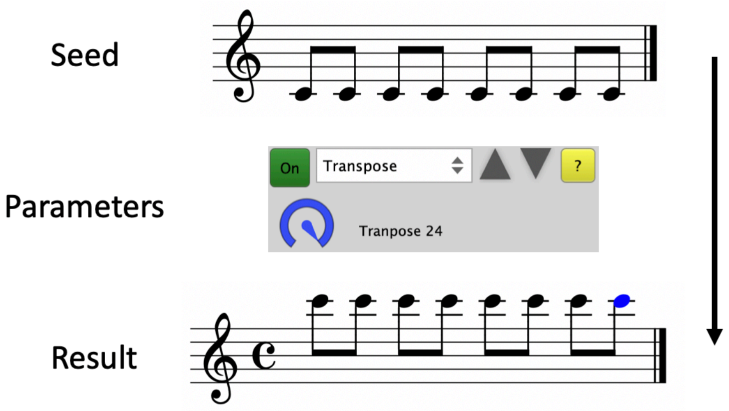


Figure 5.: Transpose plug-in.

Pitch quantiser

The pitch quantiser plug-in (Figure 4) takes a sequence and transposes any note that is not in the selected key/scale. This process is also included in the *'random note'* and *'Perlin noise'* seed generators discussed towards the end of this section.

Transpose

Transpose (Figure 5) takes the input sequence and then transposes all notes by either a positive or negative value in the range -24 to + 24 semitones. Note numbers are unable to exceed 127 or go below 0.

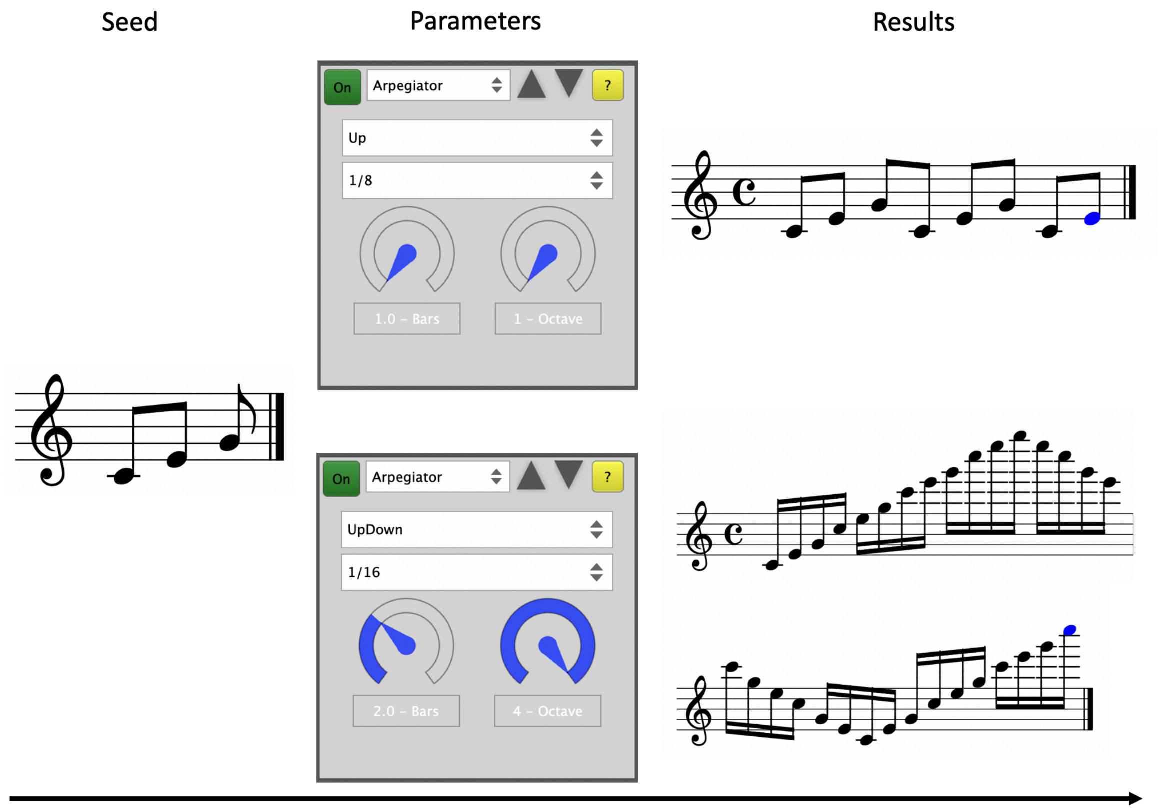


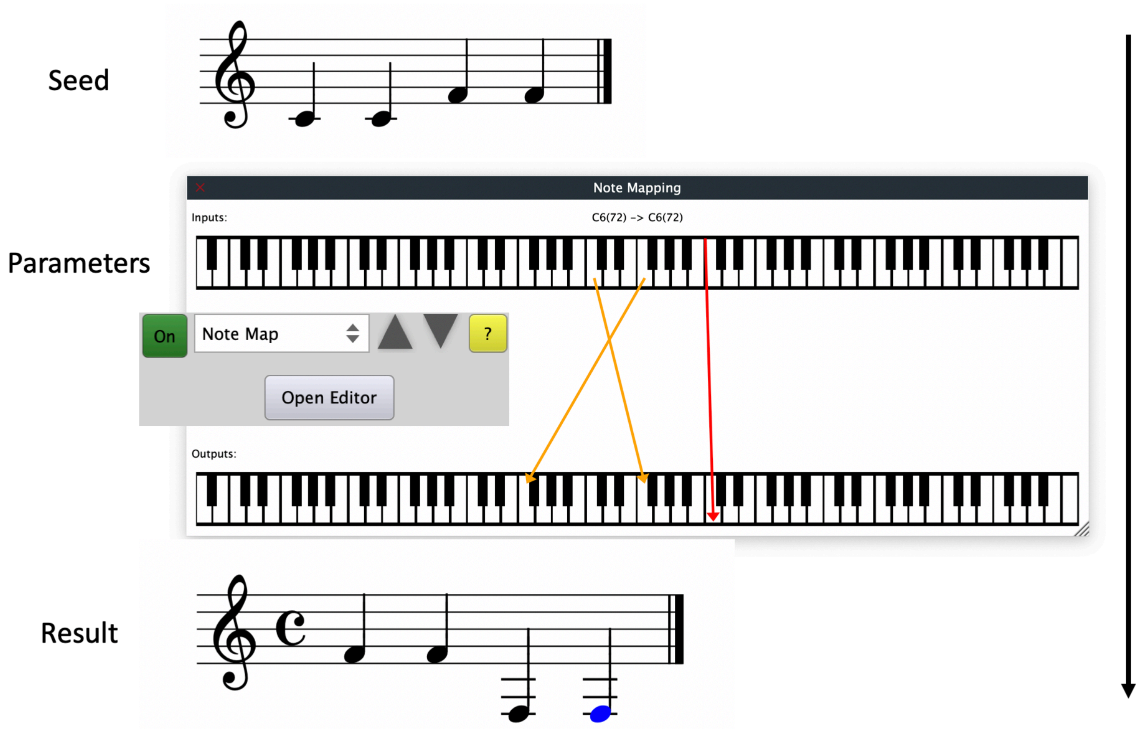
Figure 6.: Arpeggiator plug-in.

Arpeggiator

The arpeggiator (Figure 6) first analyses the input sequence and collects a set of discrete notes. For instance, if there are two C3s, only one will be taken. This set of arbitrary notes is sequenced according to the parameters in the arpeggiator plug-in. The first controls the ordering of output notes. For example, 'up' results in sequencing notes in ascending order. The octave control determines over how many octaves this is repeated. The second drop-down selects how long each note will be. Finally, the bar control determines the overall length of the output sequence.

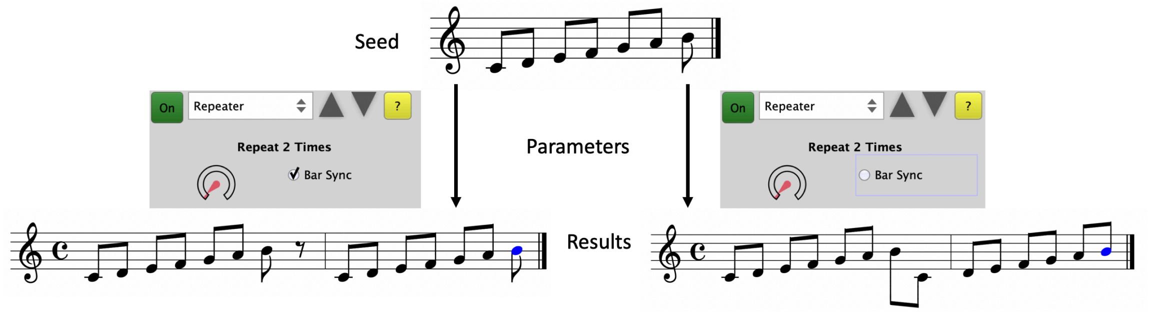
Note Map

The note map plug-in (Figure 7) works similarly to a musical *'find and replace'*. The editor window allows the user to drag arrows between an input and output keyboard. If a mapping between C4 and E5 is created, then all instances of C4 in the seed are replaced with E5 in the result.



C6(72) -> C6(72)

Figure 7.: Note map plug-in.



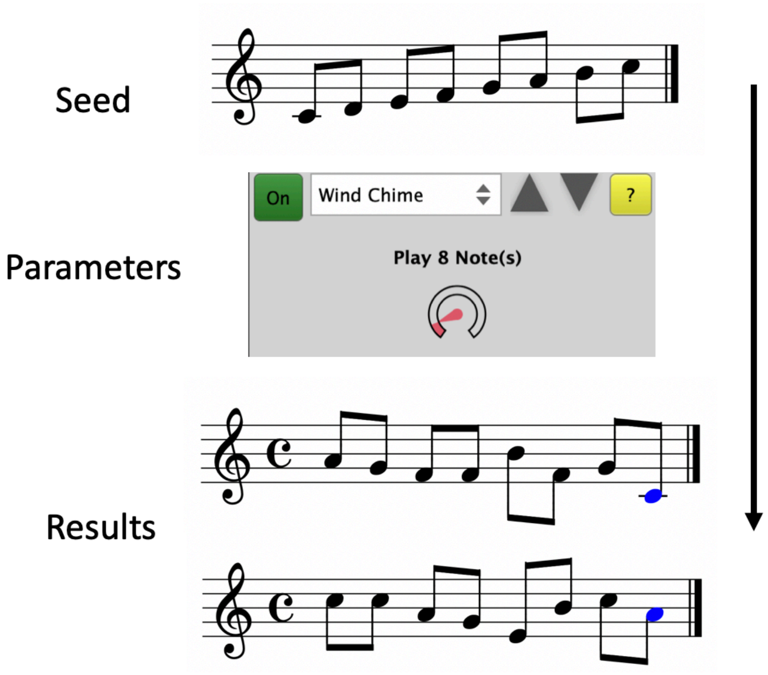
Repeat 2 Times
☑ Bar Sync

Repeat 2 Times
○ Bar Sync

Figure 8.: Repeater plug-in.

Repeater

The repeater plug-in (Figure 8) takes the input sequence and repeats it. The bar sync ensures that the repeats start on a bar division, with rests added where needed.



Wind Chime

Play 8 Note(s)

Figure 9.: Windchime map plug-in.

Windchime

The wind chime plug-in (Figure 9) reorders the input sequence. This works by randomly sampling from the available notes in the input. The output may contain more or fewer notes than the seed depending on the value set in the editor.

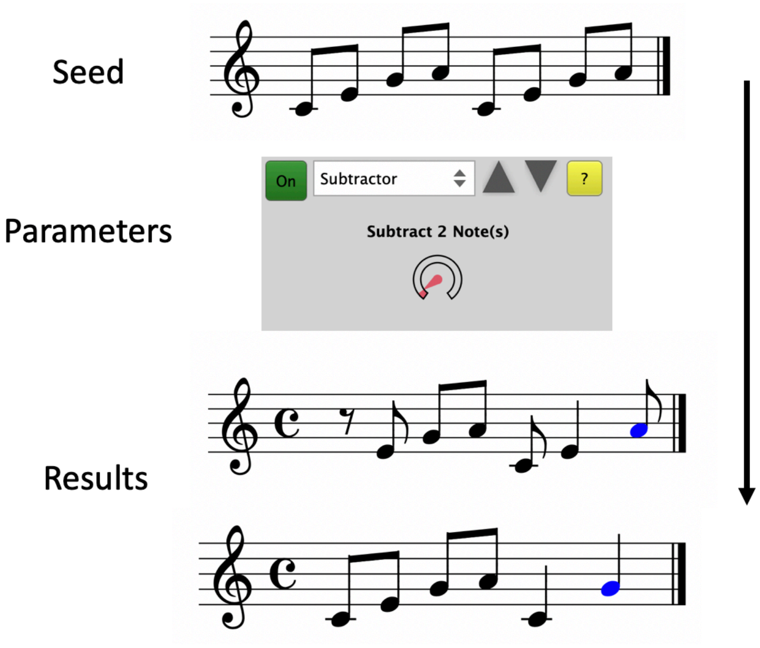


Figure 10.: Subtractor plug-in.

Subtractor

The subtract plug-in (Figure 10) removes a given number of notes from the input sequence. For example, when the parameter is set to a value of '2', then two notes are selected at random and removed (replaced with rests). If rests are removed then the sequence is shifted left. Should more notes be removed than the sequence has, a blank sequence is returned.

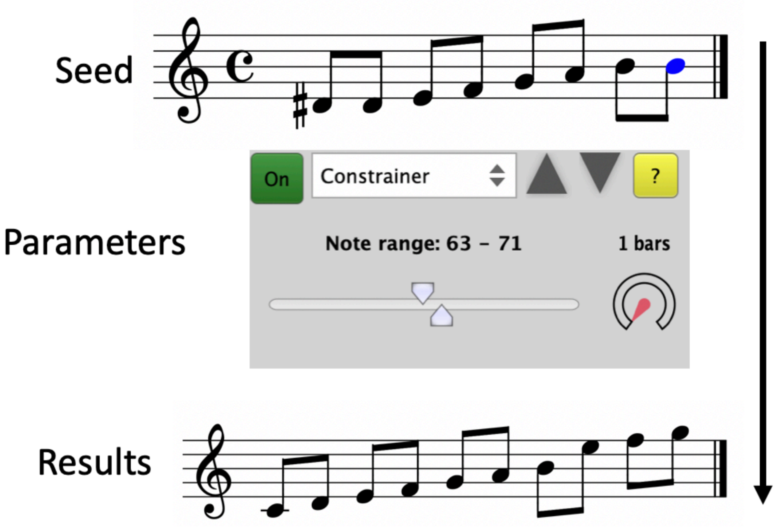


Figure 11.: Constrainer plug-in.

Constrainer

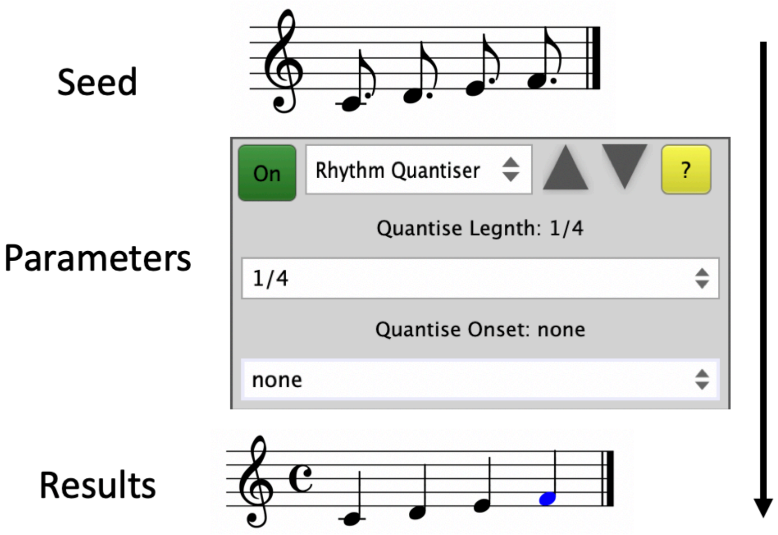
The constrainer plug-in (Figure 11) limits the range of notes and overall sequence length. The note range control prevents notes in the input sequence from exceeding or going below a certain value. Whereas the bar control clips the output at a given bar length.

Rhythm Quantiser

The rhythm quantiser plug-in (Figure 12) quantises both a note's length and onset time. Either control can be set independently and all have values from 1/4, 1/8, 1/16, 1/32, and 1/64 in set divisions.

Evolution Plug-in

Despite being developed as a prototype, the evolution plug-in (Figure 13) was never enabled in the user-facing version of IGME software. It is considered an area for future development.



Rhythm Quantiser

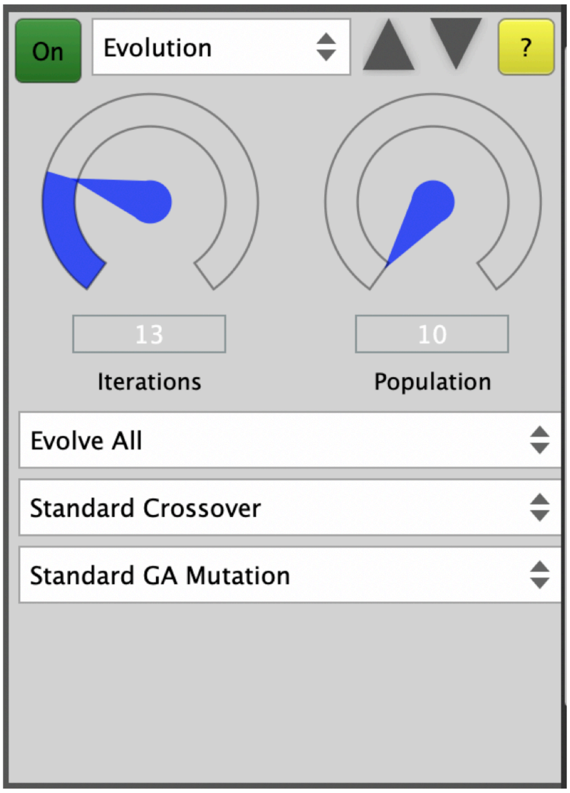
Quantise Legnth: 1/4

1/4

Quantise Onset: none

none

Figure 12.: Rhythm quantiser plug-in.



Evolution

13 Iterations

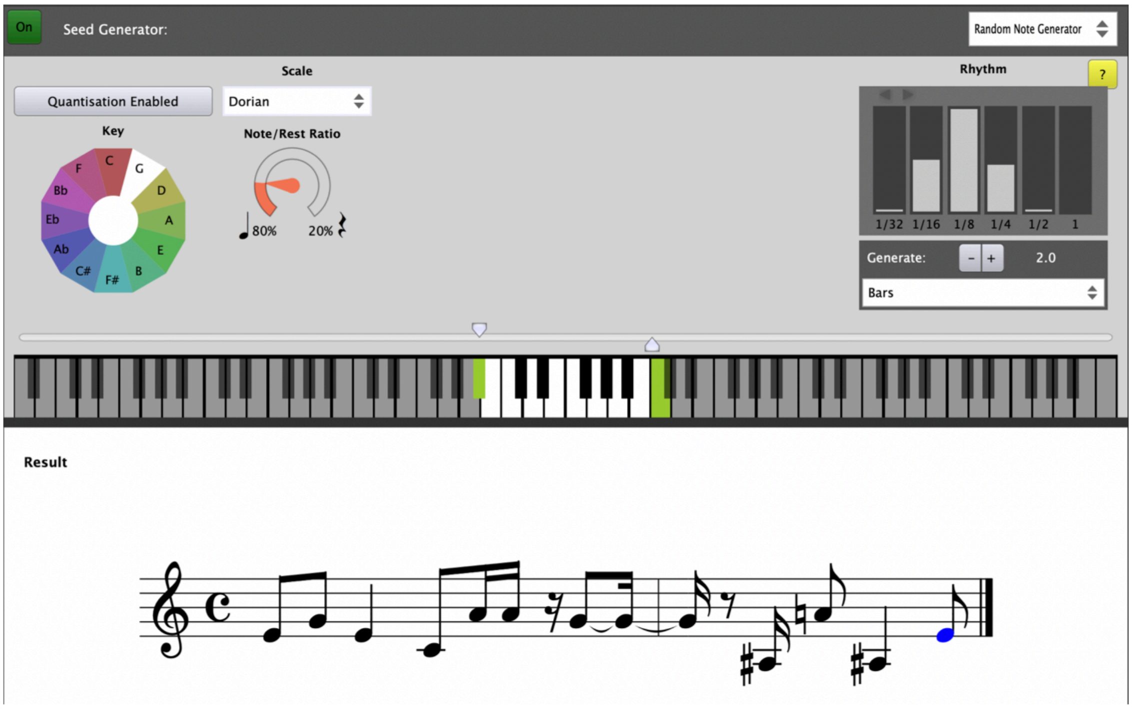
10 Population

Evolve All

Standard Crossover

Standard GA Mutation

Figure 13.: Evolution quantiser plug-in.



Seed Generator: Random Note Generator

Scale: Dorian

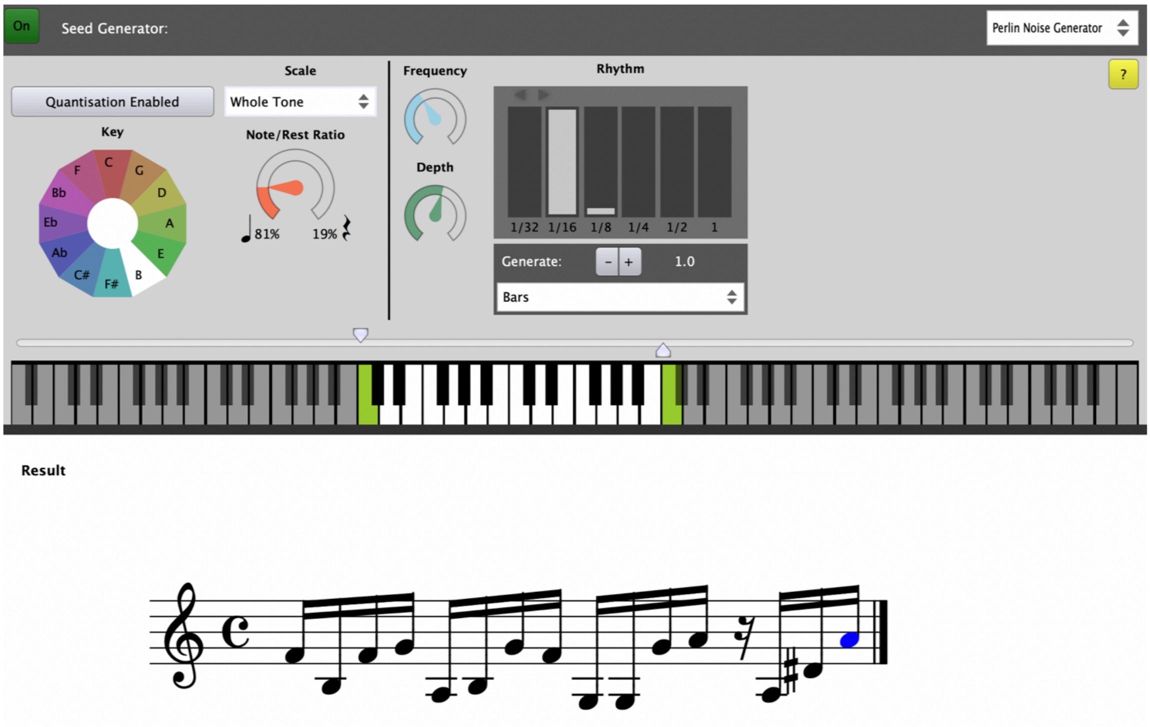
Generate: 2.0

Bars

Figure 14.: Random note seed generator plug-in.

Random Note Generator

The random note generator (Figure 14) has a range of controls for generating monophonic sequences. The fundamentals of the pitch quantiser plug-in are found to the left, ensuring that generated material stays in a given key/scale, but this can be switched off. The note rest ratio determines the probability of each generated event being a rest or note. The keyboard control governs the range in which notes can be generated. The rhythm profile determines the distribution of different note lengths. Finally, the lower right panel determines either how many discrete events or bars' worth of material is generated, with this control present in all of the seed generators.



Seed Generator: Perlin Noise Generator

Scale: Whole Tone

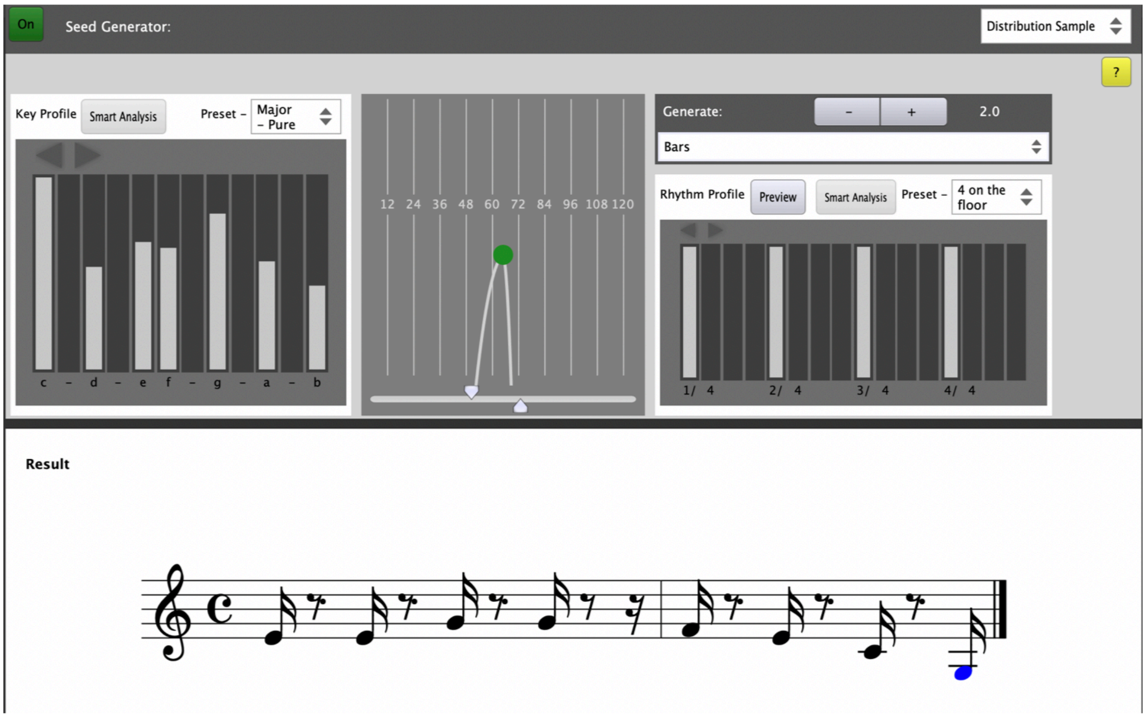
Note/Rest Ratio: 81% 19%

Generate: 1.0 Bars

Figure 15.: Perlin noise seed generator plug-in.

Perlin Noise Generator

The Perlin noise generator (Figure 15) is similar to the random note generator. However, instead of using a regular stochastic note generator, a Perlin noise process is used. This plug-in adds two controls, frequency and depth, to determine the shape of the noise.



Seed Generator: Distribution Sample

Key Profile — Preset – Major – Pure

Generate: 2.0 Bars

Rhythm Profile — Preset – 4 on the floor

Figure 16.: Distribution sample seed generator plug-in.

Distribution sample

The distribution sample (Figure 16) seed generator generates a monophonic sequence by evaluating probabilistic distributions. The key profile determines the likelihood of various pitches. The octave control (middle) sets the range in which notes are generated, while the central node skews the distribution (central octave) in the underlying model. The rhythm profile represents a bar divided into 16 even steps, and each step determines the likelihood of the note being placed at that bar position. This process supports populating the pitch and rhythm profile through the analysis of music already in the sessions. This can be done by analysing the entire song, or by analysing a specific track in a given range of bars.

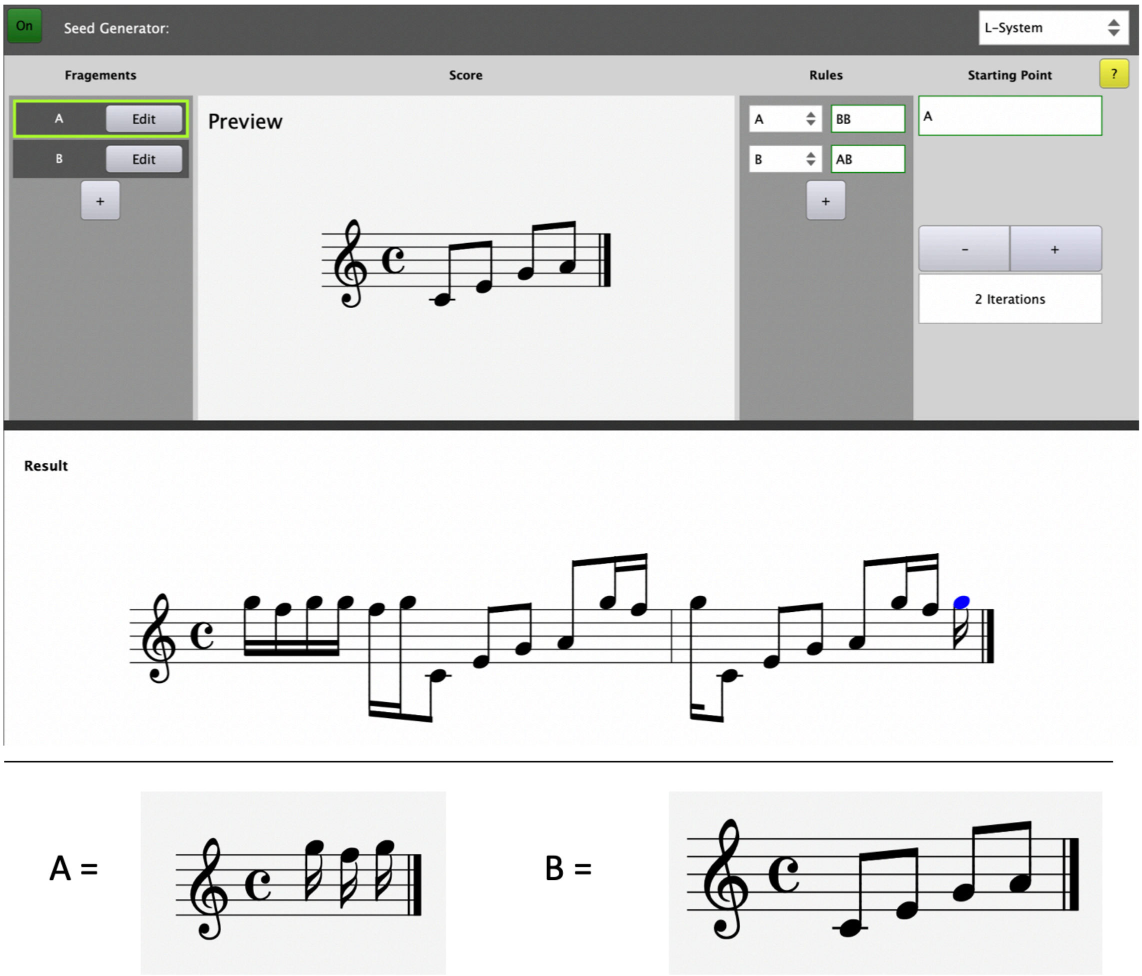


Figure 17.: L-System seed generator plug-in.

L-System

The L-system (Figure 17) seed generator is modelled on the L-system grammar (discussed in Chapter 4). This process is a formal grammar that supports algorithmically sequencing small sequences of music. The fragments panel allows an arbitrary number of sequences to be added, edited and mapped to a unique upper case letter. The rules panel determines what each symbol is replaced by during each iteration of the grammar's execution. In this case, A is replaced by BB, and then B is replaced by AB. The starting point determines the starting grammar. The final control determines how many iterations this process will run for. Once the grammar has been evaluated, each symbol in the final sequence will be mapped back to its musical counterpart. This is illustrated in Figure 18 for the first 3 iterations.

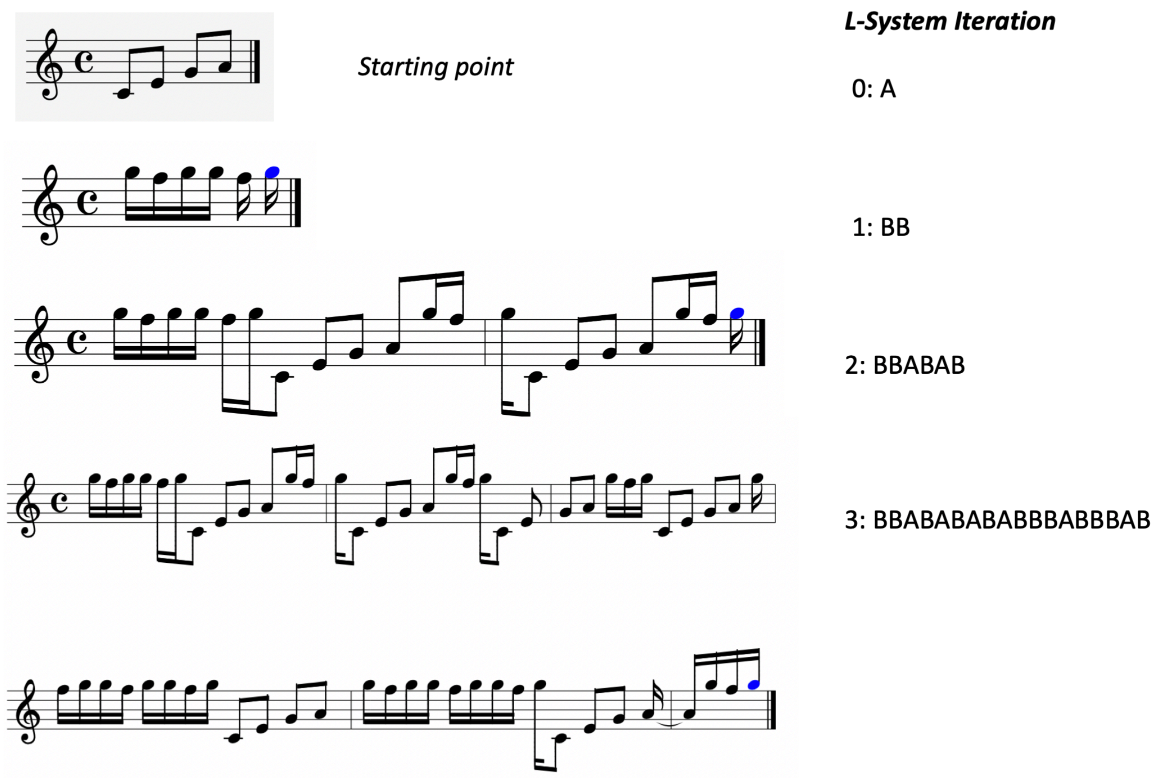
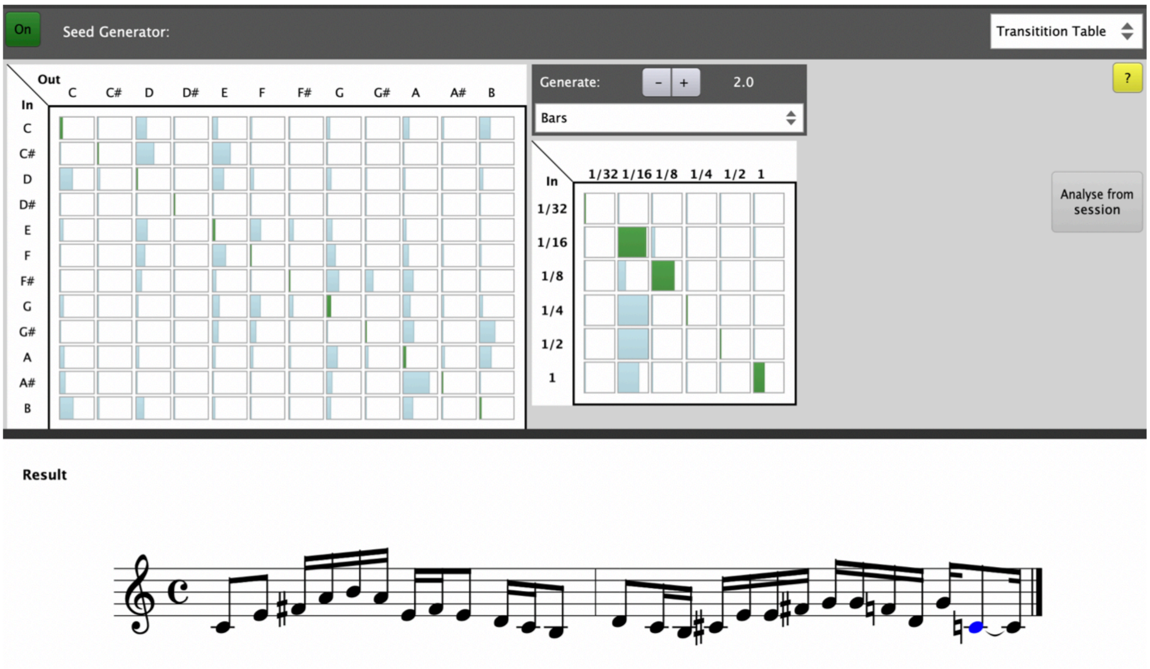


Figure 18.: L-System output.



Seed Generator: Transitition Table

Generate: 2.0

Bars

Figure 19.: Transition table seed generator plug-in.

Transition table

The transition table seed generator models a 1st order Markov model (discussed in chapter 4). The GUI presents two matrices, one for pitch and one for rhythm. For the pitch matrix, the y-axis represents the input pitch and the x-axis the output pitch, with the cell representing the probability of a note transition. Similarly, for the rhythm matrix, the last note length and the next note length is represented. Each matrix is sampled to produce a pair containing a pitch and rhythm value. Each cell can be set manually. The analysis options in the distribution sample plug-in are also available for pre-populating both matrices.

E: Additional Figures

1. Additional Figures

This section contains additional figures that would otherwise convolute the flow of text in the main body.

1.1. From Chapter 8

Type	Arguments	Description
String	Arbitrary string	Generic string message
Key press	Key values	Logs keyboard press
Iteration	Part, track, unique ID	New iteration has been computed
Slider	Slider Id, value	Slider has been moved
Note edited	Part Id, old event, new event	Note has been edited
Button press	Button Id, value	Button has been pushed
View switched focus	View type	Called when a view is switched, i.e. arrange or edit
Add track	Track Id	Track added
Remove track	Track Id	Track removed
Add part	Position, track Id	Part added at ‘position’ on timeline
Remove part	Part Id, position, track Id	Part removed at ‘position’ on timeline
Change plug-in	Type	Plugin type has changed/added
Part selected	Part, track	Called when a part is selected for editing
Playback engine	State	Playback engine state has changed
New note add	Unique Id, part, track	New note added
Note removed	Unique Id, part, track	Note deleted
Editing	Part Id, state	Called when a part is opened/closed for editing
Non fatal error	Description	Generic error message
Part swapped	Unique Id, part (pairs)	Two parts on the timeline have been swapped
Part moved	Part Id, old position, new position	Part has been moved around on timeline
Session file IO	State, name	user has either saved or loaded
Sound changed	Track, sound	Instrument sound has changed
Note parameter	String, unique Id	Note parameters have been edited, string is encoded with all data
Part converted	Part, track	Part is converted from computer to human
Notation type	Type[3]	Logs the current notation type selected for arrange, edit, and result views

Table 1.: Summary of the various types of interaction data collected from within IGME.

1.2. From Chapter 9

“What are the positive aspects of using generative music?”
Group 1
● Not tied down to what you know; funks it up.
● It helps to provides new creative ways to create music
● Quicker than using a d20, and you don’t have to type different sequences in over and over to try things out
● Creates new ideas for the user to develop with.
● New and interesting ideas that would have never been considered before
● you can see new idea
● Quick rough ideas, new ideas possibly found
● Create new ideas
● allow creativity
● New ideas will arise
● It creates new ideas that I’d never had though of
● Workflow
● You can quickly come up with small ideas in a computer that you yourself can expand further
● Creating new ideas that were unseen by the user! Creating very fast good sounding music.
● It is easy to use and float around, easy to input data and understand what is happening
● Exploring new ideas.
● creativity. exploring new possibilities
● It can create ideas that you never otherwise would come up with
● easy to come up with new ideas, which you could then adapt
● helps you to be productive, ideas
● It can help peoples stuck in a creative rut.
● Can make your projects more interesting
● creating new ideas/performing ”impossible” music
● The ability to generate new ideas that may inspire or be used in other music could help musicians to build on material they have created and get them past a road block in the creative process, for example not knowing how to progress an idea further.
● fun, new variations and ideas
Group 2
● It is faster and more efficient, and can easily be rearranged according to the users’ needs. And also allows people to compose their own music without having to learn musical theory.
● Being given an inspirational jolt when you’re stuck in a creative rut.
● That I can understand my own creative process better. Develop new ideas using strictly the values I’ve predetermined
● It increases the creative thinking during music composition and performance.
● Advance research.
● surprises! writes itself! conceptual!
● It’s fun and can eliminate of writer’s block. I delight in the application and relaxation of constraints.

Table 2.: Unedited responses for the above question.

"What are the negative aspects of using generative music?"
Group 1
● You feel insignificant and useless
● It forces different workflow which could be undesirable
● Less guided and precise, you don't have quite as much control initially, but at the end of the day you aren't forced to use exactly what's generated, so it's alright
● Limits the user to create their own style.
● Potentially removing the human artist completely
● it is not your composition
● Doesn't feel as creative, feels like cheating if you find a real good melody
● Easy to use too much generation
● 1.more synth bases instruments 2. allow changeable time parameters
● Criticism from outside for not "owning" your work
● Not always having control of what is made
● Automation - Art's tombstone
● there can be a lack in originality / identity in generative music over music that a person has made themselves
● Can take very long to get something you like by chance
● no comment
● Sometimes, it can be hard to find the exact thing the artist wants.
● loss of control. uncertain of outcome, sometimes terrible result
● Its unpredictable and hence can sound eratic
● almost too easy to make something, takes away part of the challenge?
● none
● You can fall in a wormhole trying to find a sound from your head.
● Can be argued that it isn't really your work
● People might overrely on it?
● Taking control away from the user may encourage laziness where instead of deliberately choosing what to add or change about a piece of music they let an algorithm decide for them.
● unpredictability
Group 2
● The availability and ease of use might have a negative affect on job opportunities for human composers.
● It can make some people feel uncomfortable, "AI" is seen as a threat by some (usually those that don't know much about it!)
● can get carried away with these rules. Some of my rules I don't even recognise nor could I rationalise
● Generative music systems should always be though and developed as tools that enhance and support human creativity rather than replacing it.
● None
● sometimes it sounds horrid
● none!

Table 3.: Unedited responses for the above question.

1.3. From Chapter 10

Part Development profiles

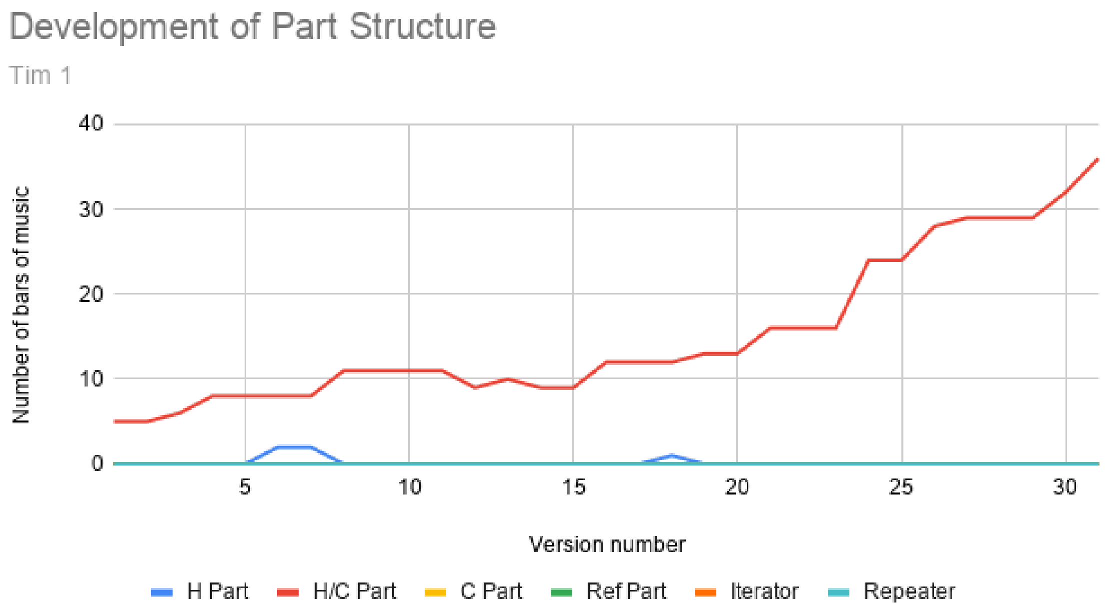


Figure 1.: Part development for Tim 1.

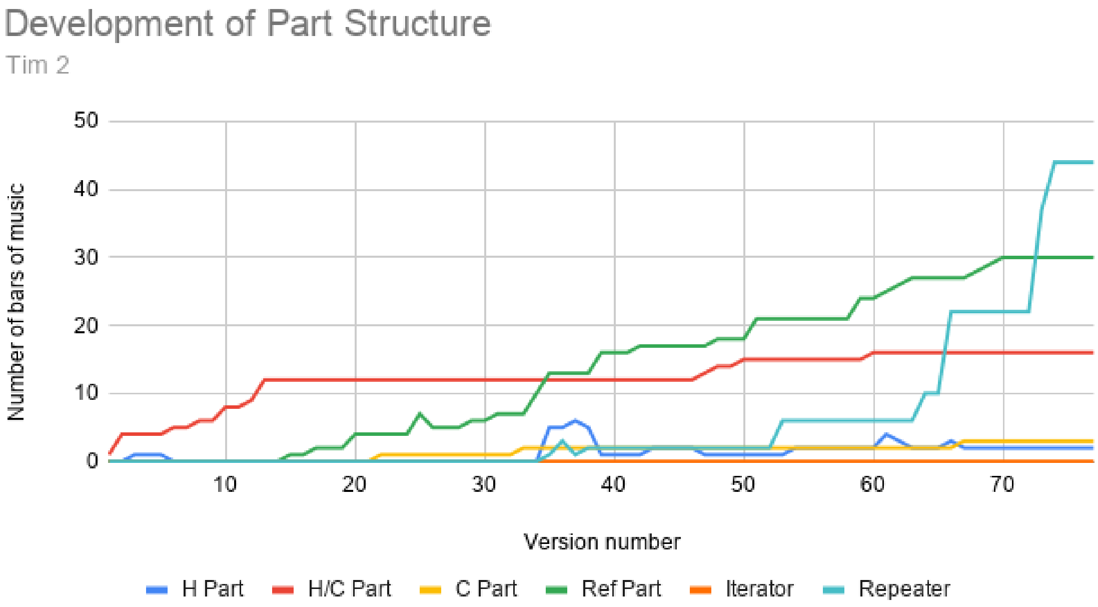


Figure 2.: Part development for Tim 2.

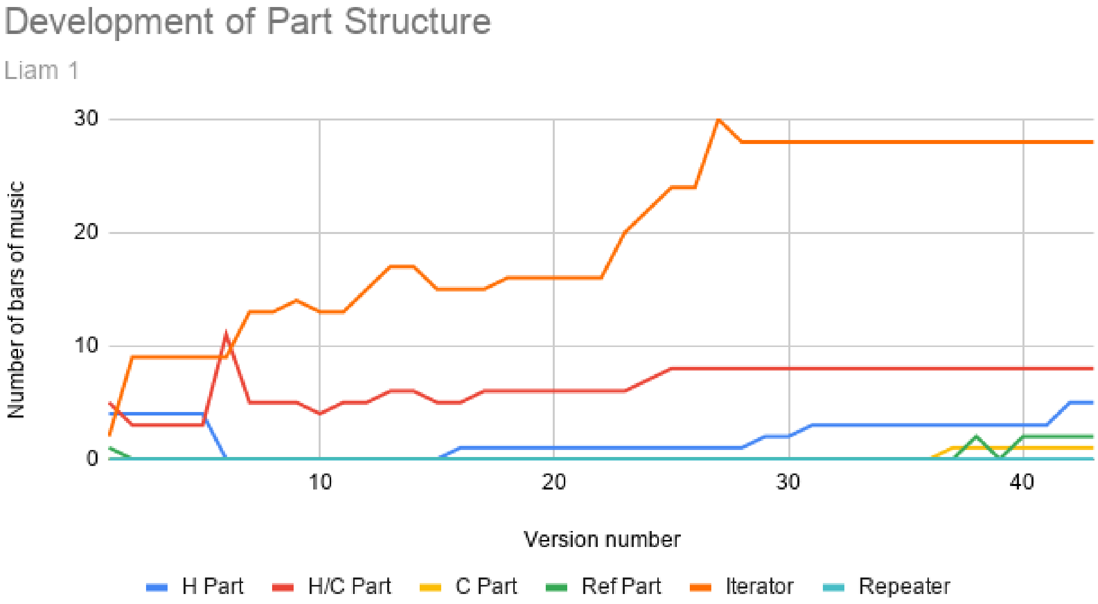


Figure 3.: Part development for Liam 1.

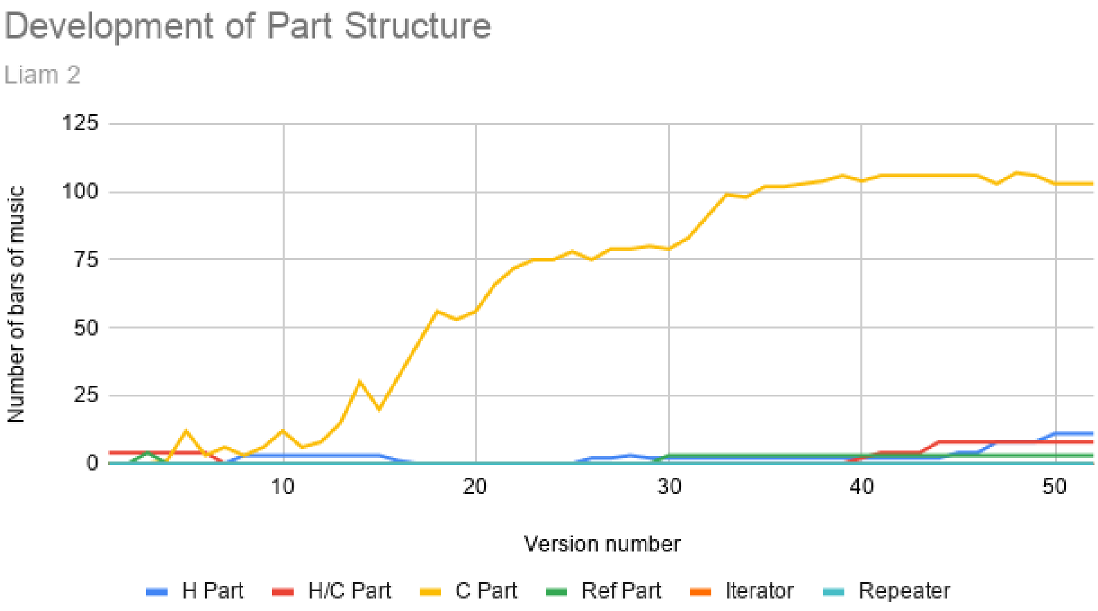


Figure 4.: Part development for Liam 2.

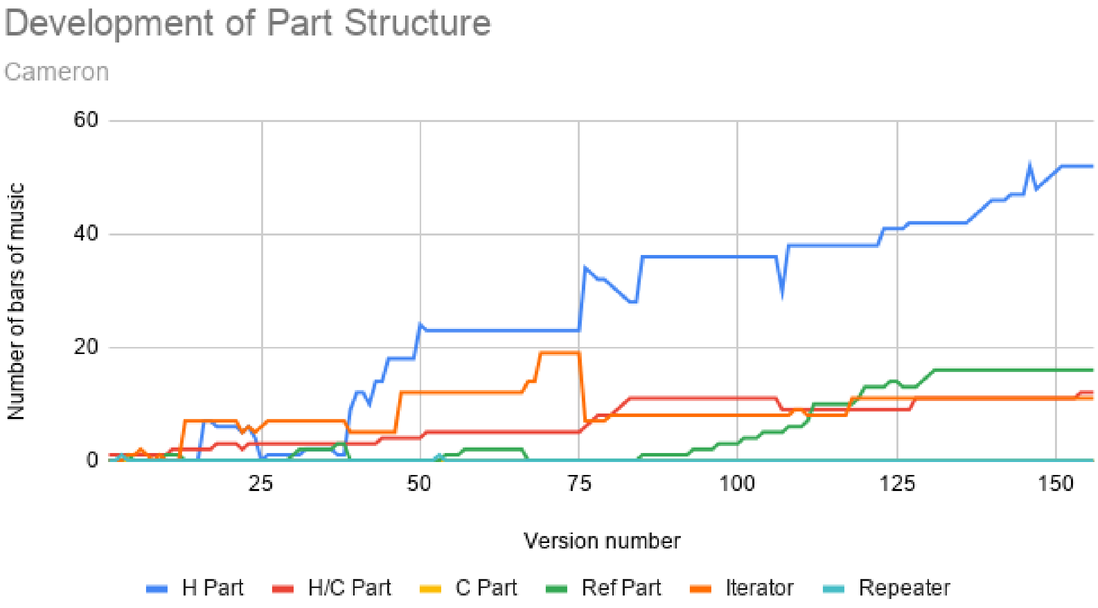


Figure 5.: Part development for Cameron (across both sessions since they worked on just one piece).

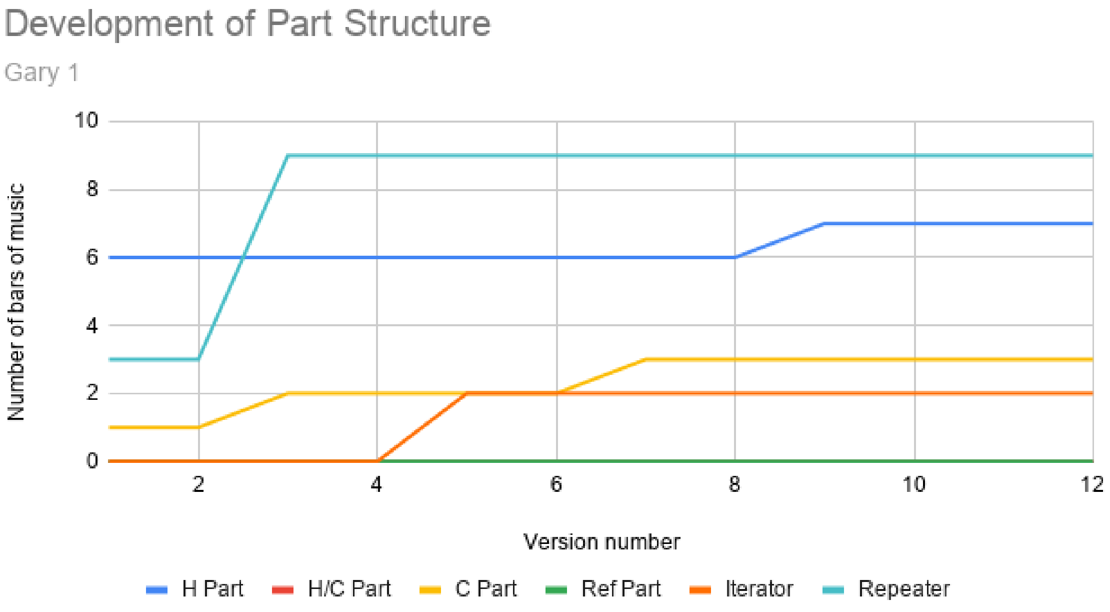


Figure 6.: Part development for Gary 1.

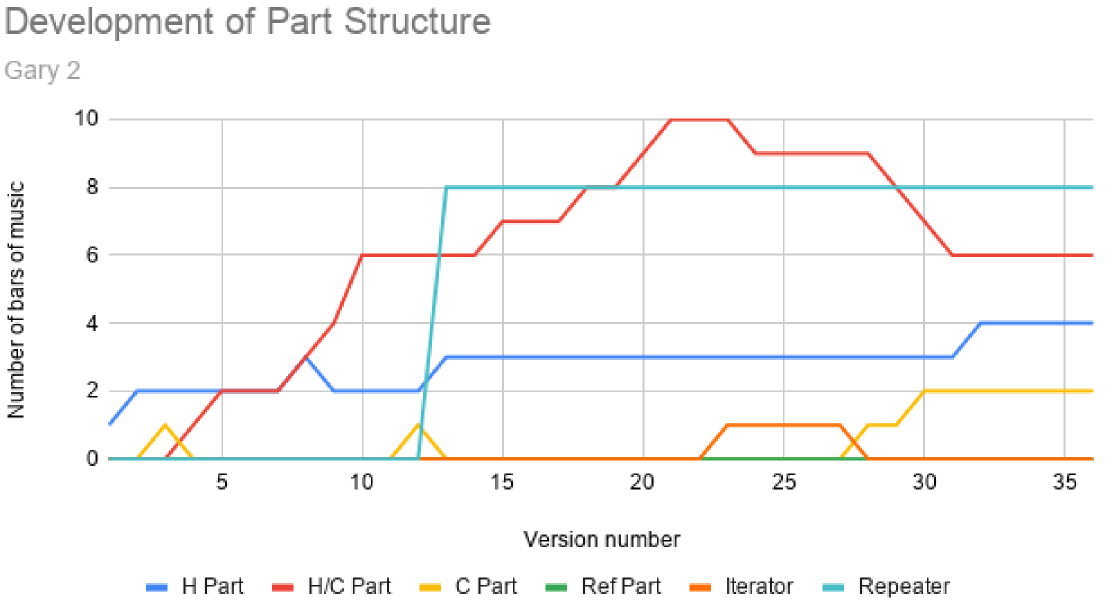


Figure 7.: Part development for Gary 2.

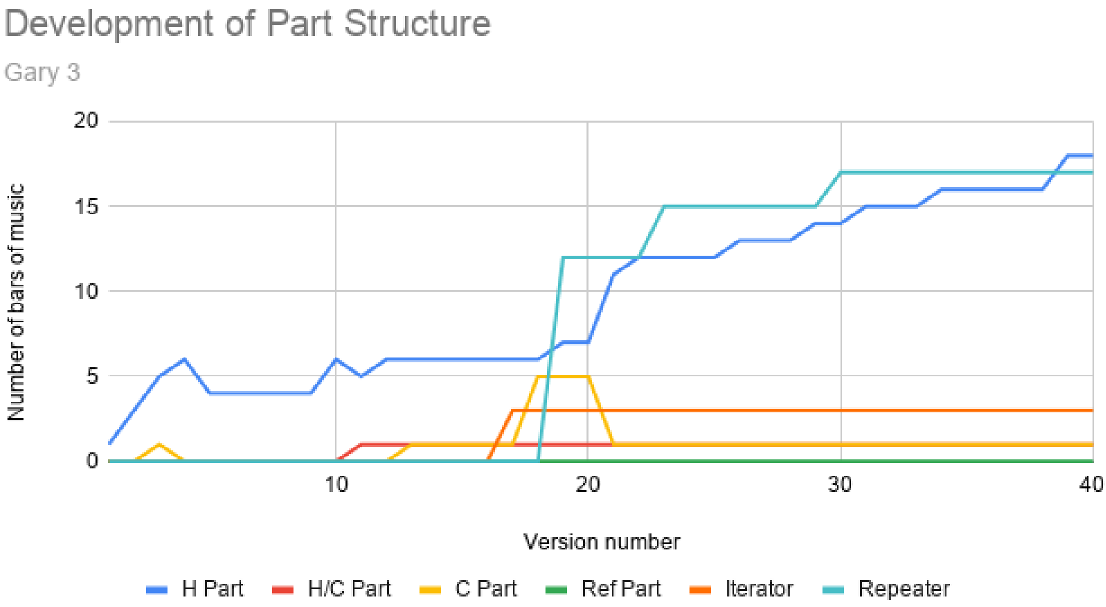


Figure 8.: Part development for Gary 3.

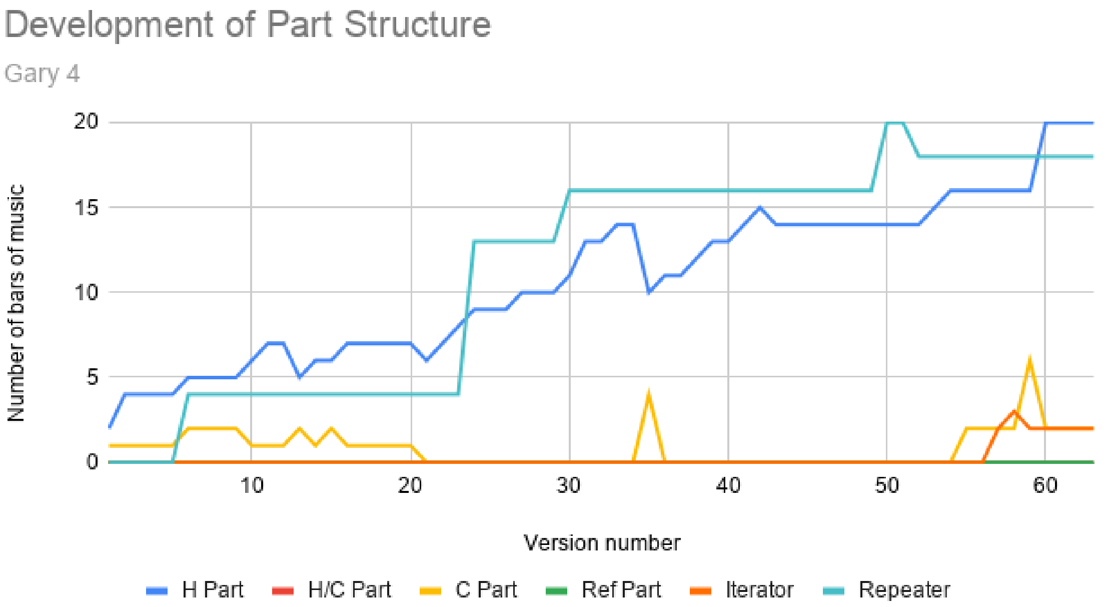
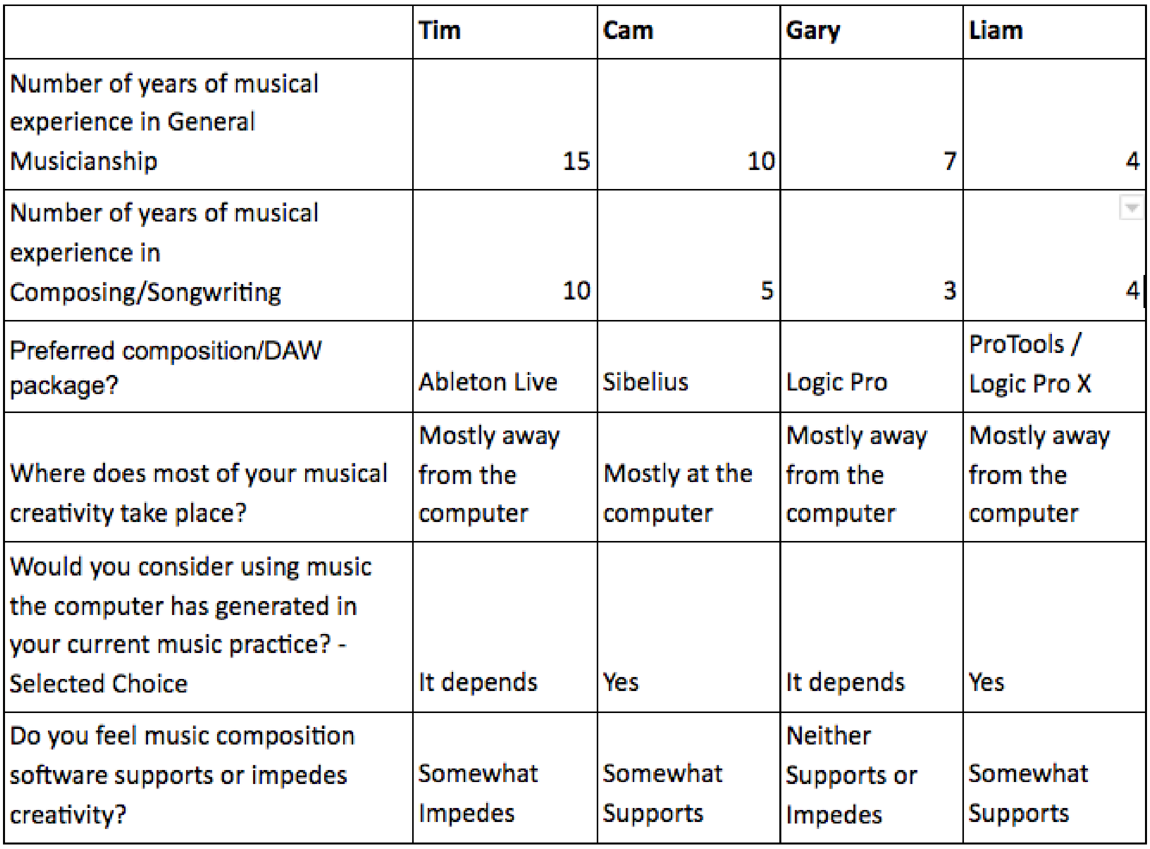


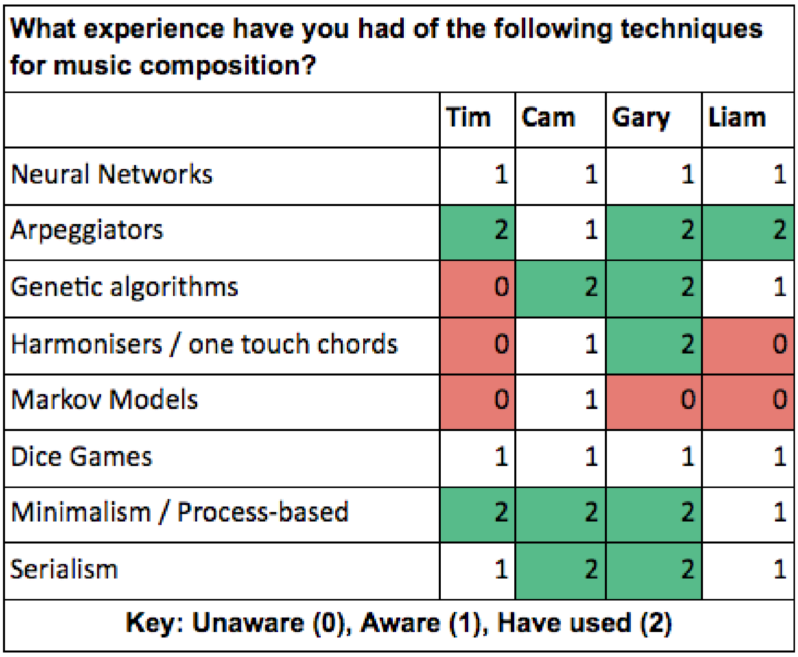
Figure 9.: Part development for Gary 4.

Survey C - Participant background and experiences of music



	Tim	Cam	Gary	Liam
Number of years of musical experience in General Musicianship	15	10	7	4
Number of years of musical experience in Composing/Songwriting	10	5	3	4
Preferred composition/DAW package?	Ableton Live	Sibelius	Logic Pro	ProTools / Logic Pro X
Where does most of your musical creativity take place?	Mostly away from the computer	Mostly at the computer	Mostly away from the computer	Mostly away from the computer
Would you consider using music the computer has generated in your current music practice? - Selected Choice	It depends	Yes	It depends	Yes
Do you feel music composition software supports or impedes creativity?	Somewhat Impedes	Somewhat Supports	Neither Supports or Impedes	Somewhat Supports

Figure 10.: Summary of each user.

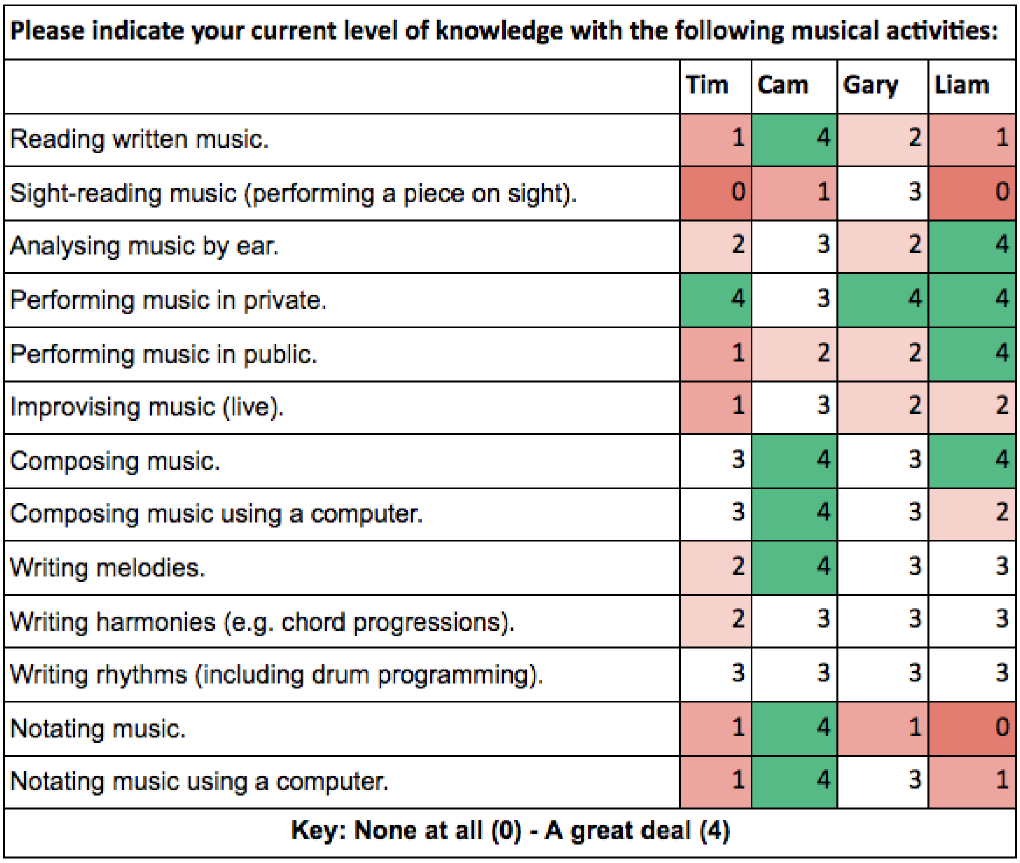


What experience have you had of the following techniques for music composition?

	Tim	Cam	Gary	Liam
Neural Networks	1	1	1	1
Arpeggiators	2	1	2	2
Genetic algorithms	0	2	2	1
Harmonisers / one touch chords	0	1	2	0
Markov Models	0	1	0	0
Dice Games	1	1	1	1
Minimalism / Process-based	2	2	2	1
Serialism	1	2	2	1

Key: Unaware (0), Aware (1), Have used (2)

Figure 11.: User's exposure to generative techniques.

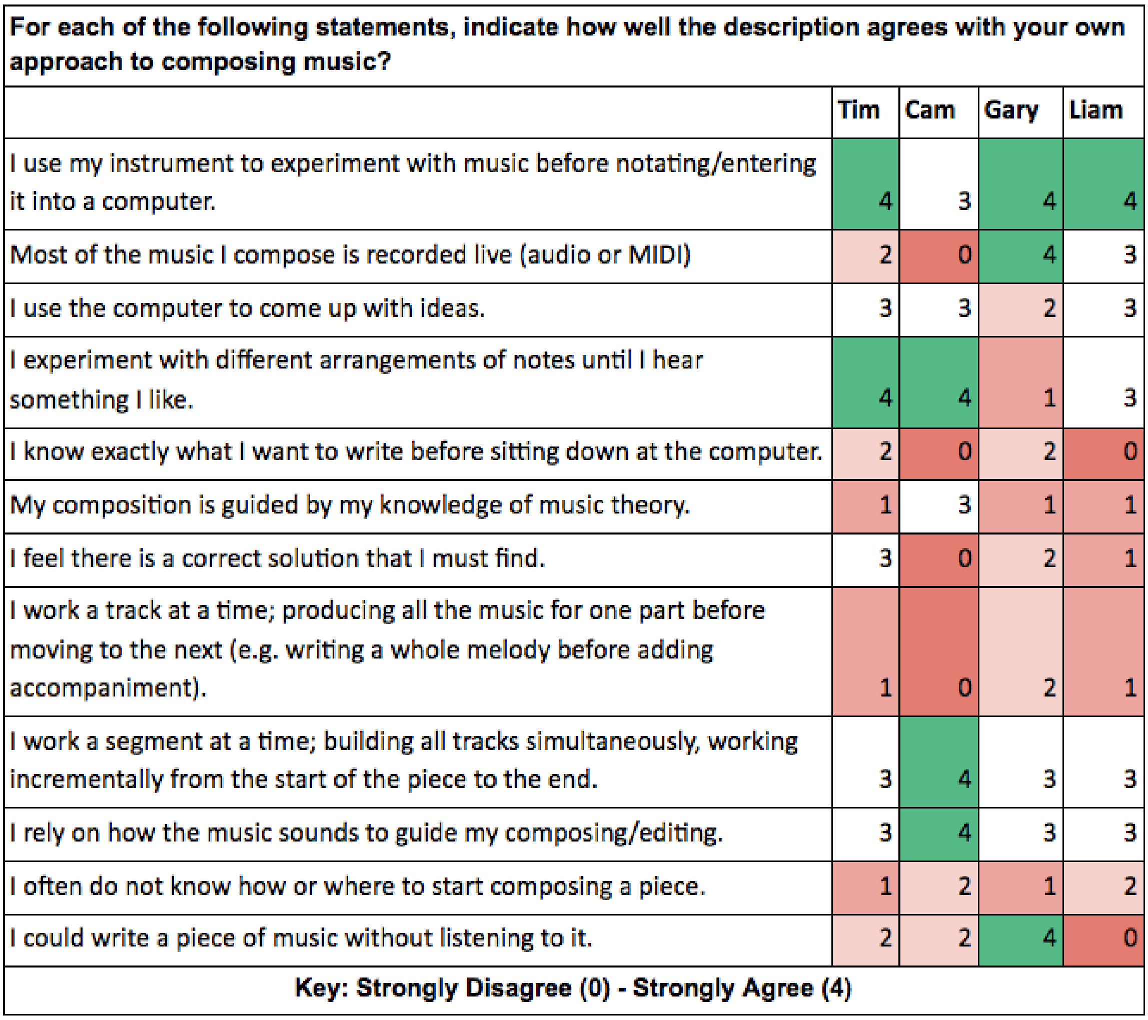


Please indicate your current level of knowledge with the following musical activities:

	Tim	Cam	Gary	Liam
Reading written music.	1	4	2	1
Sight-reading music (performing a piece on sight).	0	1	3	0
Analysing music by ear.	2	3	2	4
Performing music in private.	4	3	4	4
Performing music in public.	1	2	2	4
Improvising music (live).	1	3	2	2
Composing music.	3	4	3	4
Composing music using a computer.	3	4	3	2
Writing melodies.	2	4	3	3
Writing harmonies (e.g. chord progressions).	2	3	3	3
Writing rhythms (including drum programming).	3	3	3	3
Notating music.	1	4	1	0
Notating music using a computer.	1	4	3	1

Key: None at all (0) - A great deal (4)

Figure 12.: User's knowledge of various musical activities.

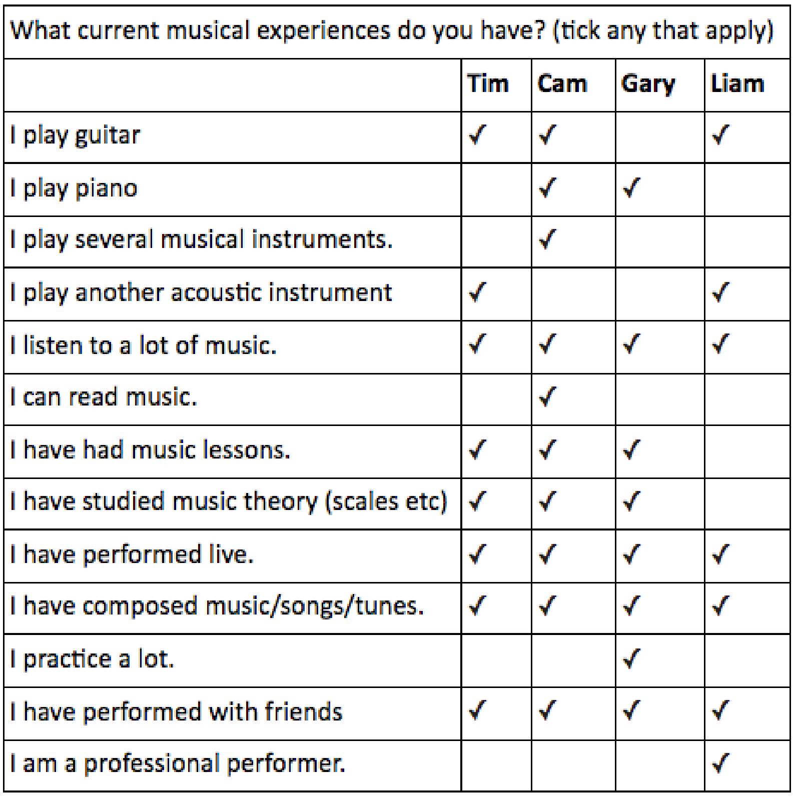


For each of the following statements, indicate how well the description agrees with your own approach to composing music?

	Tim	**Cam**	**Gary**	**Liam**
I use my instrument to experiment with music before notating/entering it into a computer.	4	3	4	4
Most of the music I compose is recorded live (audio or MIDI)	2	0	4	3
I use the computer to come up with ideas.	3	3	2	3
I experiment with different arrangements of notes until I hear something I like.	4	4	1	3
I know exactly what I want to write before sitting down at the computer.	2	0	2	0
My composition is guided by my knowledge of music theory.	1	3	1	1
I feel there is a correct solution that I must find.	3	0	2	1
I work a track at a time; producing all the music for one part before moving to the next (e.g. writing a whole melody before adding accompaniment).	1	0	2	1
I work a segment at a time; building all tracks simultaneously, working incrementally from the start of the piece to the end.	3	4	3	3
I rely on how the music sounds to guide my composing/editing.	3	4	3	3
I often do not know how or where to start composing a piece.	1	2	1	2
I could write a piece of music without listening to it.	2	2	4	0

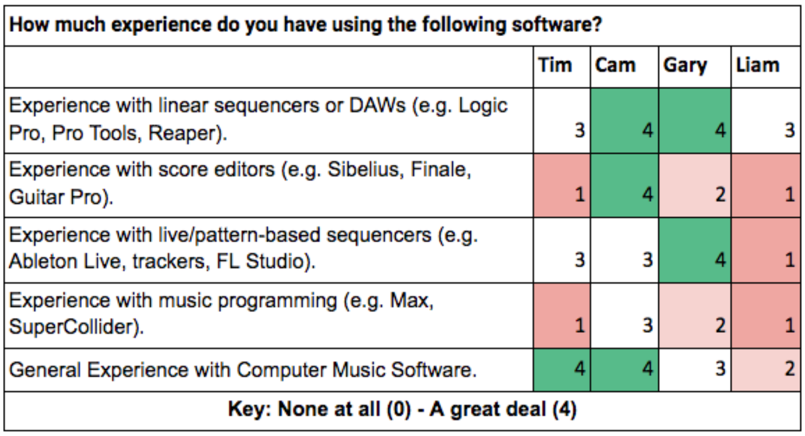
Key: Strongly Disagree (0) - Strongly Agree (4)

Figure 13.: User's approach to composition.



What current musical experiences do you have? (tick any that apply)				
	Tim	Cam	Gary	Liam
I play guitar	✓	✓		✓
I play piano		✓	✓	
I play several musical instruments.		✓		
I play another acoustic instrument	✓			✓
I listen to a lot of music.	✓	✓	✓	✓
I can read music.		✓		
I have had music lessons.	✓	✓	✓	
I have studied music theory (scales etc)	✓	✓	✓	
I have performed live.	✓	✓	✓	✓
I have composed music/songs/tunes.	✓	✓	✓	✓
I practice a lot.			✓	
I have performed with friends	✓	✓	✓	✓
I am a professional performer.				✓

Figure 14.: User's general experience of music.



How much experience do you have using the following software?				
	Tim	Cam	Gary	Liam
Experience with linear sequencers or DAWs (e.g. Logic Pro, Pro Tools, Reaper).	3	4	4	3
Experience with score editors (e.g. Sibelius, Finale, Guitar Pro).	1	4	2	1
Experience with live/pattern-based sequencers (e.g. Ableton Live, trackers, FL Studio).	3	3	4	1
Experience with music programming (e.g. Max, SuperCollider).	1	3	2	1
General Experience with Computer Music Software.	4	4	3	2
Key: None at all (0) - A great deal (4)				

Figure 15.: User's experience of music software.

Survey B responses

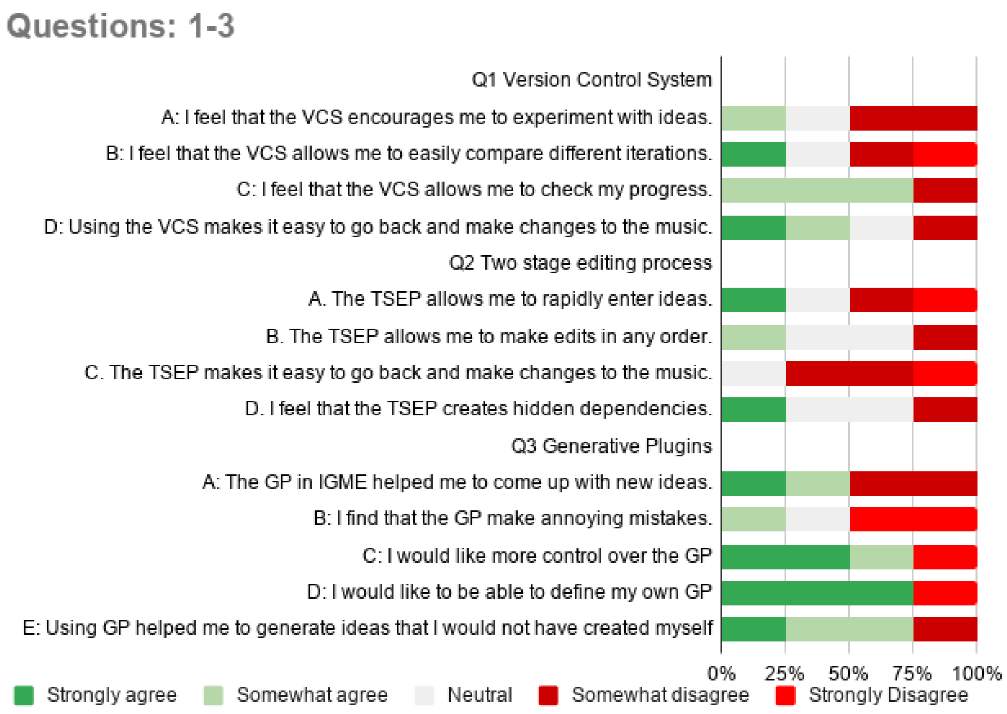


Figure 16.: Results for questions 1-3 from survey B, at the end of the first session of the longitudinal study in chapter 10.

Having used IGME, how much of the musical creativity do you attribute to the computer?

Session 1	*Session 2*
Most of it to myself	Most of it to myself
Most of it to myself	Most of it to myself
Most of it to the computer	Most of it to the computer
Neutral	Most of it to myself

Table 4.: Summary of responses for above question.

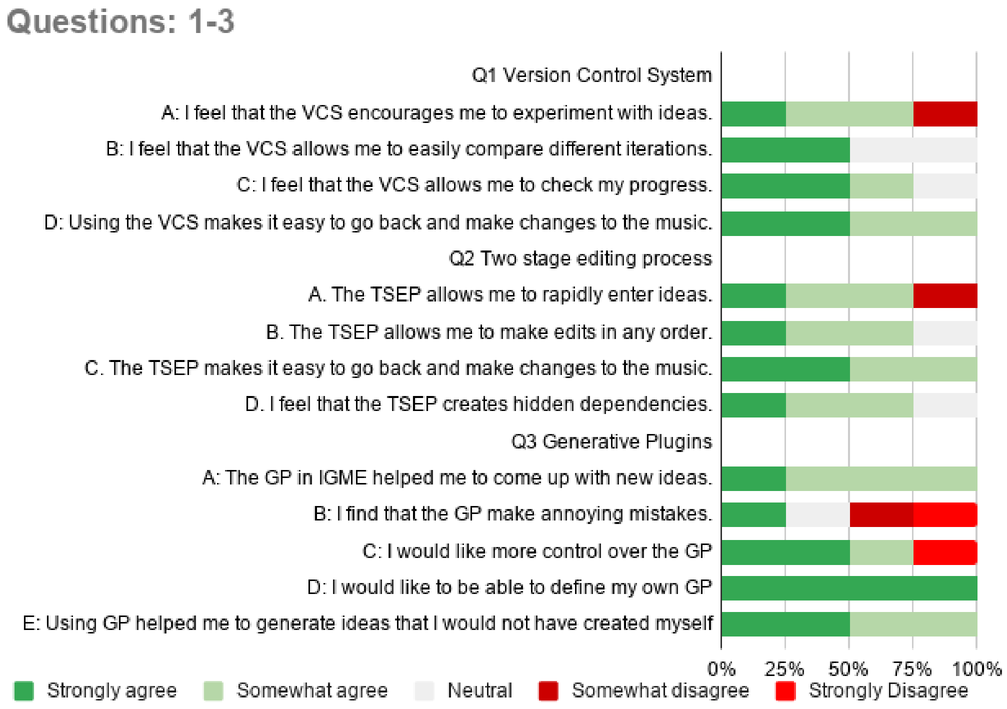


Figure 17.: Results for questions 1-3 from survey B, at the end of the second session of the longitudinal study in chapter 10.

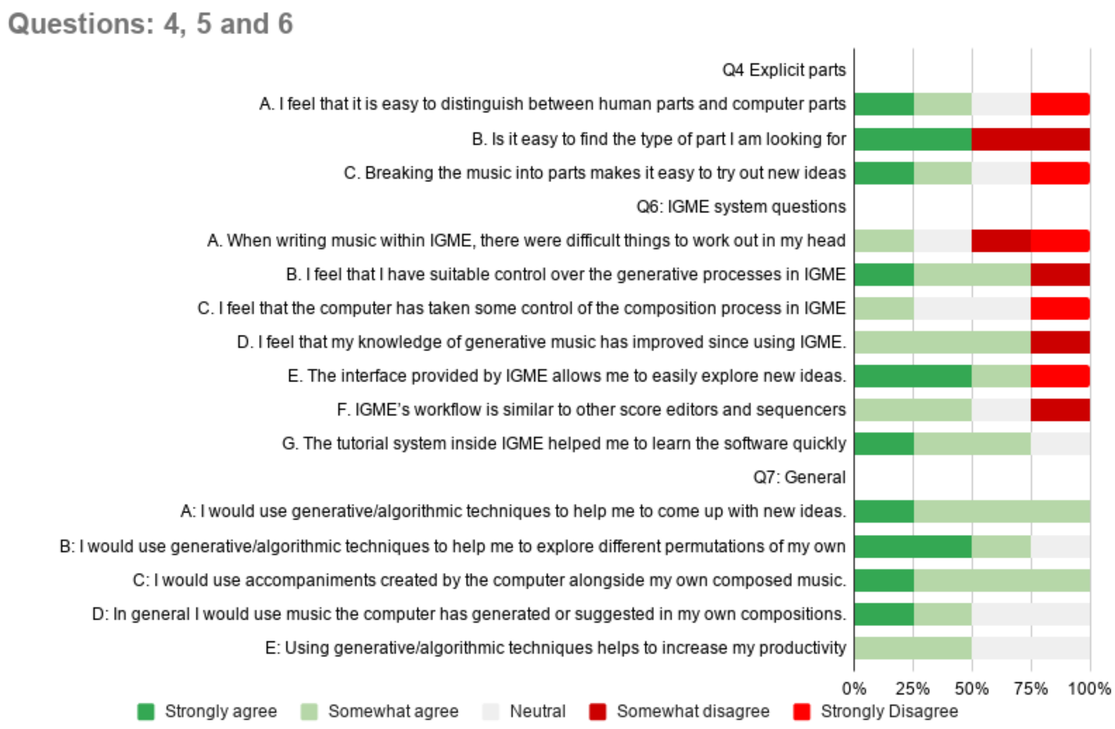


Figure 18.: Results for questions 4, 6, and 7 from survey B, at the end of the first session of the longitudinal study in chapter 10.

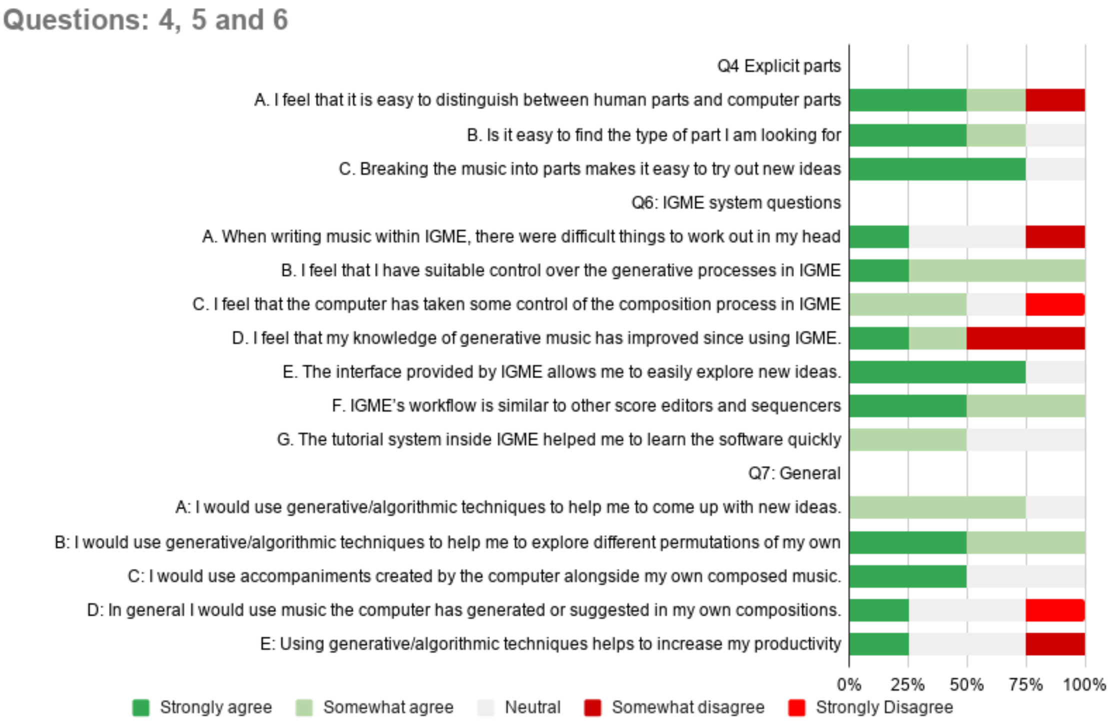


Figure 19.: Results for questions 4, 6, and 7 from survey B, at the end of the second session of the longitudinal study in chapter 10.

Is the authorship of your creative output a concern when using generative techniques?

Session 1	Session 2
No - My general composition style is quite process-based anyway. Even without the computer assistance, I would end up following some form of pattern.	No - I feel like as I choose the processes, I have influenced and created the result.
No - Because it is up to the user to dictate how the computer will generate the music not purely the computer on it's own. e.g. generative music vs AI music.	No
Yes - If the artist is the one who created the generative algorithms, then authorship isn't a concern. If an artist is using pre made generative algorithms then it can be a concern.	Yes
No	No - Generative effects aren't forced, is it still your choice. You have the control, especially when given more tool to manipulate the output.

Table 5.: Summary of responses for above question (un-edited).

F: IGME Tutorial Sheet

1. IGME Tutorial Sheet From Initial Pilot Study 1

Introduction to IGME: the Interactive Generative Music Editor

Samuel Hunt

March 2018

1 Preword

Firstly thank you for taking the time to be a participant in this workshop. Your participation will have significant bearing on the outcome of this research, and the development of future music technology software.

Contents

2 Introduction

Many pieces of software exist for the creation of algorithmic and generative music using a computer. However the interfaces provided by these software systems requires the user to edit the data directly, in either a patch based (i.e. MaxMsp) or code based environment. IGME (Interactive Generative Score Editor) is designed to address this problem, by integrating a wide array of generative and algorithmic techniques inside commonly used score editing interfaces. This workshop is designed to get your feedback on this early prototype. The primary aim of this research looks at how generative music techniques can be embedded inside existing systems such as Sequencers, Score Editors and Digital Audio Workstations.

2.1 Getting Started

To start your session, please open the following application by double clicking on the IGME Icon. Please run the application from the supplied memory stick.

3 Exploring the arrange View

To those familiar with existing Digital Audio Workstations (DAW's) this view should feel familiar. The arrange window is divided into 3 sections.

The **arrange** part is where the individual musical blocks are arranged. The example shows a two track arrangement, with three 1-bar blocks per track. The sliders can be used to navigate the arrange window as it gets larger. Clicking on an individual musical block shows a **preview** of its musical contents on a score (Figure 1 - Preview). The final part of the session view **Other**, shows the Changes, Scratch-pad, Library, and other components, these are controlled in a tab view. The exact purpose of these are not explored in this session.

3.1 Playback and Global View

Upon start up, each of the segments on the time-line are randomly generated. To view an overview of the score the **render** button must first be pushed, it should turn green to indicate it has been completed. The role of this is

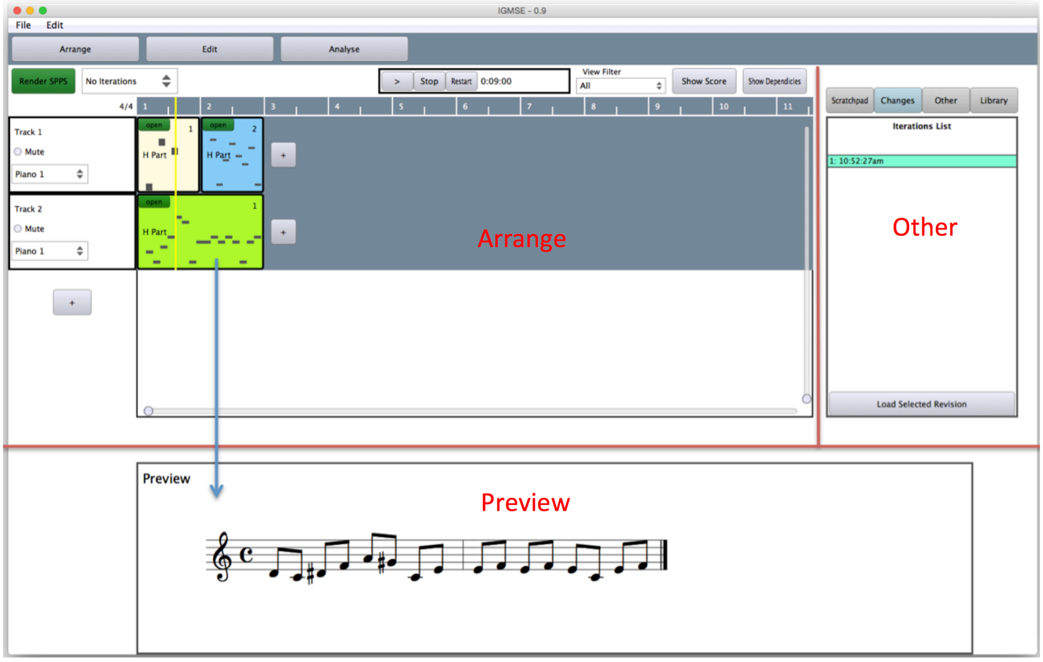


Figure 1: Main View of IGME. You should see this after opening the application.

to combine each of the blocks together to form a single multi-track piece of music. The **option box** next to the **render** button will be explored later. You can now click the **show score** button to view the score. You can use the transport bar to audition piece using the inbuilt synthesis engine. Please note that the synthesis engine is still a work in progress.

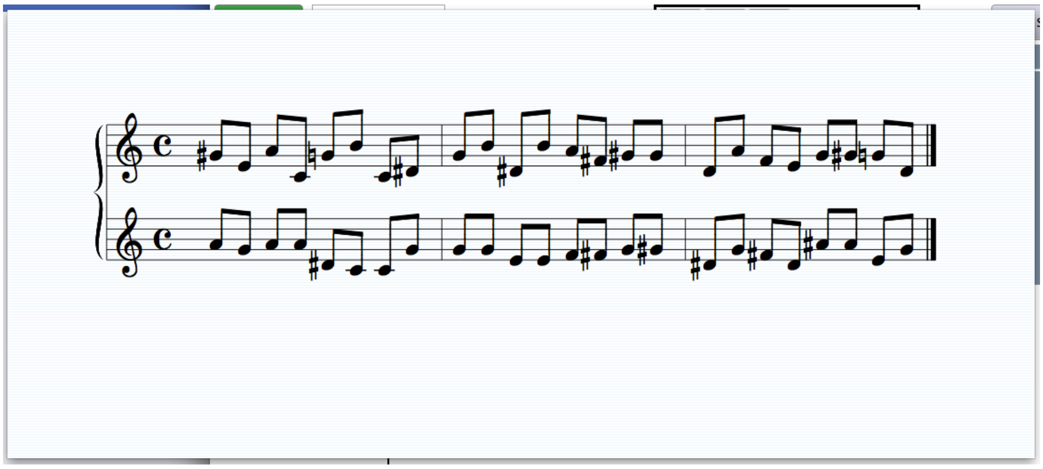


Figure 2: Pop-up window showing the complete score.

4 Exploring the Edit View

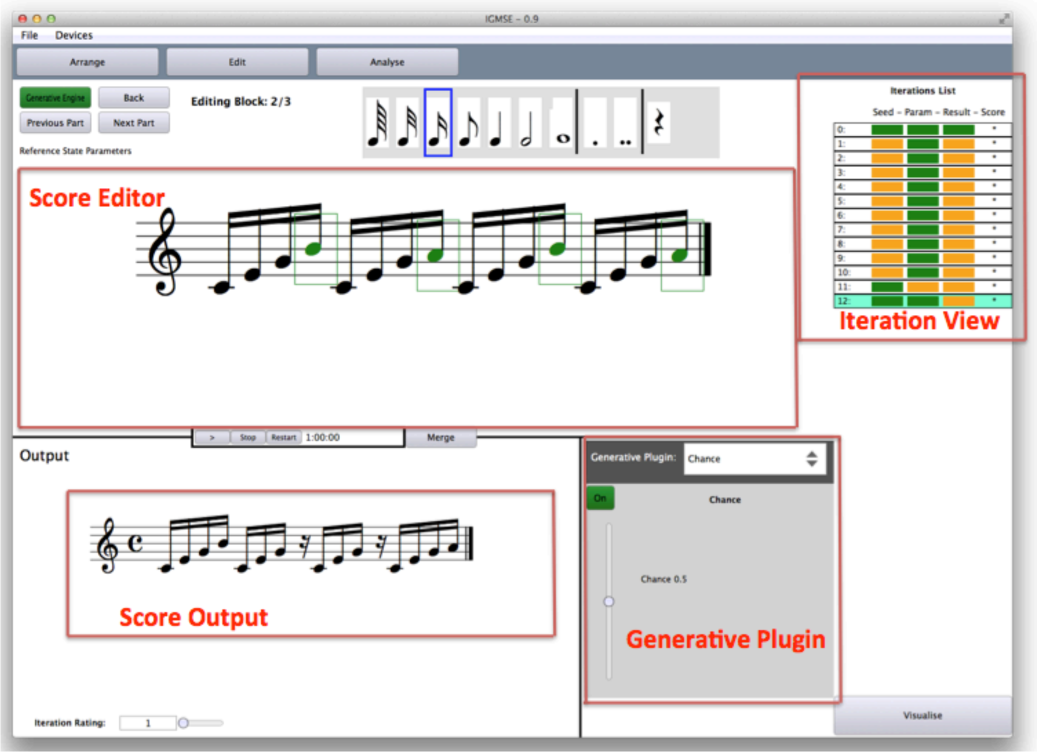
By double clicking on a block the edit view (Figure 3) should open. This view is responsible for editing the individual notes of each block, as well as applying various generative techniques. Similar to before, the **Edit** view is broken down into 4 main sections.

4.1 Score Editor

The score editor section allows the content to be edited using familiar score notation. This view is context sensitive and can be switched for different forms of notation, e.g. western score or piano roll.

4.2 Score Output

The output window at the bottom shows the net result of the input score and any generative parameters that have been applied. If no such parameters have been applied the output will be exactly the same as the input. Note that we don not hear the input score, only the output score. Without applying any generative parameters, IGME works much the same way as other common music sequencers.



Editing Block: 2/3

Generative Plugin: Chance

Chance 0.5

Iteration Rating: 1

Figure 3: Overview of the main part editor.

4.3 Generative Plugin

The Generative Plug-in window at the bottom right is used for applying generative techniques to the content in the score editor. Different techniques can be applied by switching between them in the options box.

4.4 Iteration View

To help keep track of the various edits made to the input score and generative output score the iteration view keeps track of all of these edits, so they can be recalled at any time, ensuring no content ever gets lost.

5 The First Activity

Now IGME is up and running we can now begin the first activity. Although IGME is designed for interactive composition with algorithmic and generative music techniques it is also possible to use IGME simply as a music sequencer, as we shall now demonstrate. Firstly open up Activity 1, by selecting **File - Load Session**. Note you need to load the folder, rather then the files inside it.

You should now see a single track with only one block. Open up the edit view for block one on track one by **double clicking**.

The task of this activity is to notate the following sequence (shown in figure 4) inside the block we have just opened for editing. See the keyboard commands section (bellow).

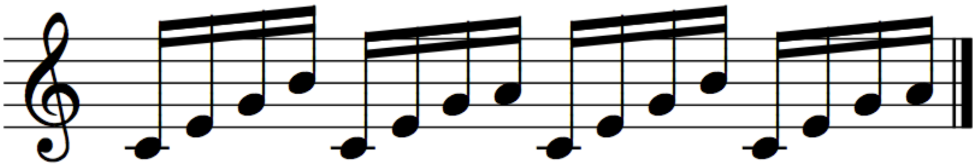


Figure 4: Score to notate.

5.1 Keyboard Commands

The following keyboard commands are used to edit notes more quickly in IGME. Please note that these controls work only on notes that are selected (i.e. blue).

A-G : Used to change the pitch of a note

Up, Down : Used to increment/decrement the pitch of a note

Cmd + Up or Down : Used to increment/decrement the octave of a note

1-7 : Used to change note duration.

Left, Right : Used to navigate between notes.

5.2 Manual vs Auto note entry

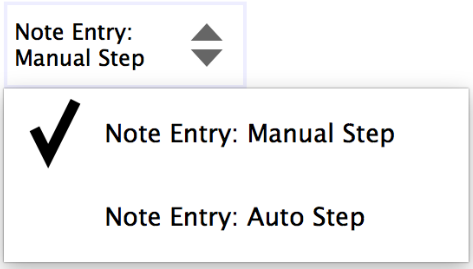


Figure 5: Note Entry

When editing notes two different entry methods can be used, **auto note step** and **manual note step**. In manual mode the selected notes will not progress to the next note when the selected notes' pitch is changed. In auto mode, changing the pitch of the selected note will cause the next note in the sequence to be subsequently selected. Both modes have their advantage.

Use Manual step: when you want to edit multiple selected notes at the same time, or want complete control over the editing process.

Use Auto step: to quickly enter a new sequence of notes, or to edit an existing sequence.

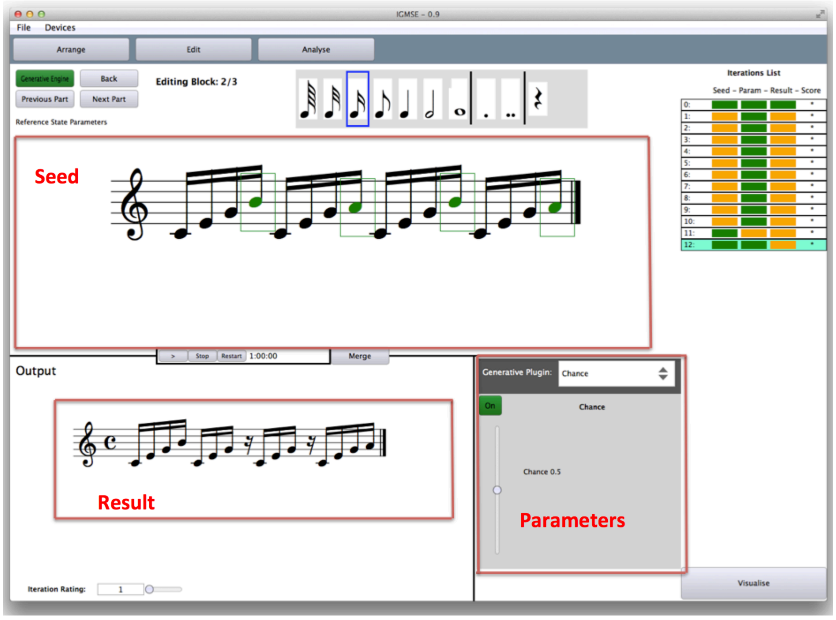
6 The Seed, Parameters and Result

The theory behind IGME breaks the composition of blocks down into three parts. These are:

Seed : The name given to the musical material that is edited (Figure 6).

Parameters : The parameters allow you to choose the type generative model and configure it.

Result : The result is the final music material produced from passing the seed material through the generative process. This is music that gets rendered into the final score.



Editing Block: 2/3

1:00:00

Generative Plugin: Chance

Chance 0.5

Iteration Rating: 1

Iterations List

	Seed	Param	Result	Score
0:	🟩	🟩	🟩	•
1:	🟧	🟩	🟧	•
2:	🟧	🟩	🟧	•
3:	🟧	🟩	🟧	•
4:	🟧	🟩	🟧	•
5:	🟧	🟩	🟧	•
6:	🟧	🟩	🟧	•
7:	🟧	🟩	🟧	•
8:	🟧	🟩	🟧	•
9:	🟧	🟩	🟧	•
10:	🟧	🟩	🟧	•
11:	🟩	🟧	🟧	•
12:	🟩	🟩	🟧	•

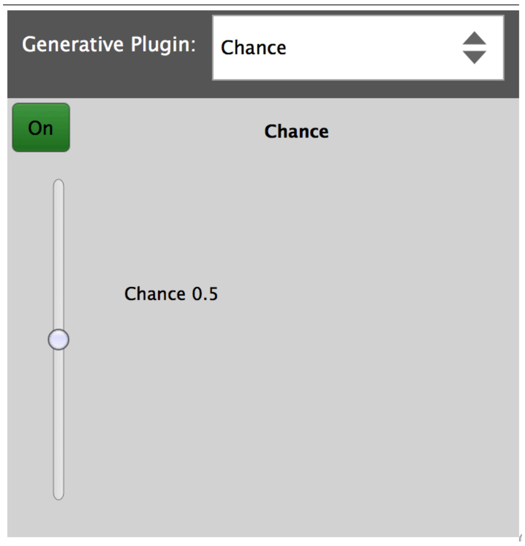
Figure 6: Seed, Parameters and Result

7 The Second Activity

Although IGME can make use of complex generative music algorithms, it can also do simple note transformations. The activity here is to take the block we prepared in activity 1 and apply a simple stochastic algorithm that randomly removes notes present in the input sequence to produce output variations (iterations). You can also load a preprepared version from file (activity 2).

7.1 Adding chance

The **chance** plug-in sets a selected note to have a certain probability of appearing in the final score. For example a note with only 50% probability has


Generative Plugin: Chance

Chance 0.5

Figure 7: Chance plugin

only a 50% chance of appearing in the result output.

In the generative plug-in window, select **Chance** from the options drop down window. The interface has only two controls, a button for turning the effect on and off, and a slider for setting the probability. To apply this to a note in the input score simply select the note(s) you want and then set their probability by moving the slider in the plug-in window. A probability of 100% means that the note is guaranteed to be in the output sequence.

For this activity set each 4th note in the sequence to have a 50% probability of occurring. See figure 8 for more information.

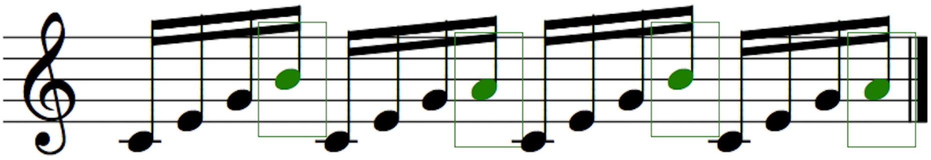
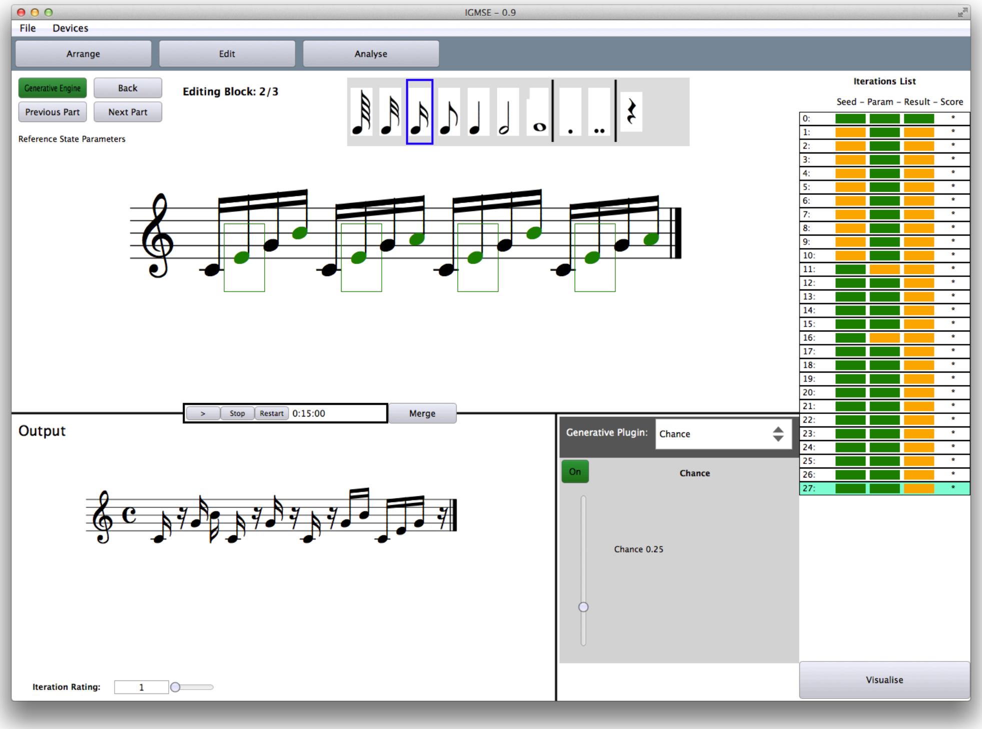


Figure 8: Correct notes positions for 50% chance parameter.



Editing Block: 2/3

0:15:00

Generative Plugin: Chance

Chance 0.25

Iteration Rating: 1

Iterations List

Seed – Param – Result – Score

0:	🟩	🟩	🟩	*
1:	🟧	🟩	🟧	*
2:	🟧	🟩	🟧	*
3:	🟧	🟩	🟧	*
4:	🟧	🟩	🟧	*
5:	🟧	🟩	🟧	*
6:	🟧	🟩	🟧	*
7:	🟧	🟩	🟧	*
8:	🟧	🟩	🟧	*
9:	🟧	🟩	🟧	*
10:	🟧	🟩	🟧	*
11:	🟩	🟧	🟧	*
12:	🟩	🟩	🟧	*
13:	🟩	🟩	🟧	*
14:	🟩	🟩	🟧	*
15:	🟩	🟩	🟧	*
16:	🟩	🟧	🟧	*
17:	🟩	🟩	🟧	*
18:	🟩	🟩	🟧	*
19:	🟩	🟩	🟧	*
20:	🟩	🟩	🟧	*
21:	🟩	🟩	🟧	*
22:	🟩	🟩	🟧	*
23:	🟩	🟩	🟧	*
24:	🟩	🟩	🟧	*
25:	🟩	🟩	🟧	*
26:	🟩	🟩	🟧	*
27:	🟩	🟩	🟧	*

Figure 9: Note and parameter positions for activity 7.1.

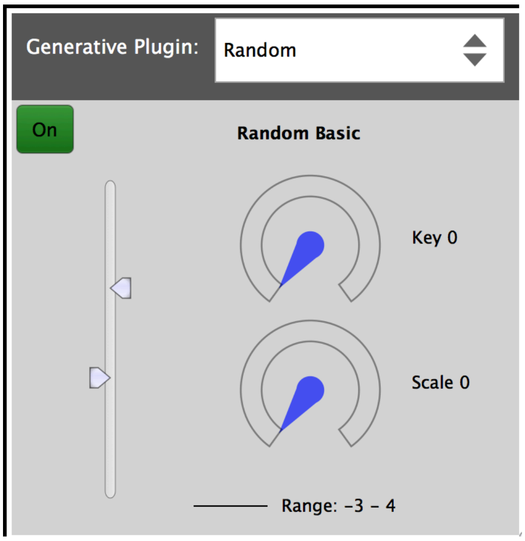
Now set each 2nd note in the sequence to have a 25% probability (notes 2,6,10 and 14). See figure 9 for more information.

You should now be able to press **space bar** to toggle a new generation and subsequently hear the output. Watch as the right hand side change tracker adds a new entry for each iteration. Note that the generative toggle button must be enabled for the changes to take place.

7.2 Adding Randomness

We will now use a different generative technique to randomise and modify the pitch of each note. Using the same sequence as before, hit the **Clear Generative Effects**. In the generative plug-in window select **Random** from the drop down window.

Next select a selection of notes (as many as you want) then move the **range** slider in the plugin window. Set the range to go between -2 and +2 semitones. When you audition the piece each iteration will vary the range of notes between -2 and +2 producing some interesting variations. Feel free to experiment with different ranges and selections of notes.



Generative Plugin: Random

Range: -3 – 4

Figure 10: Random effect, plugin window

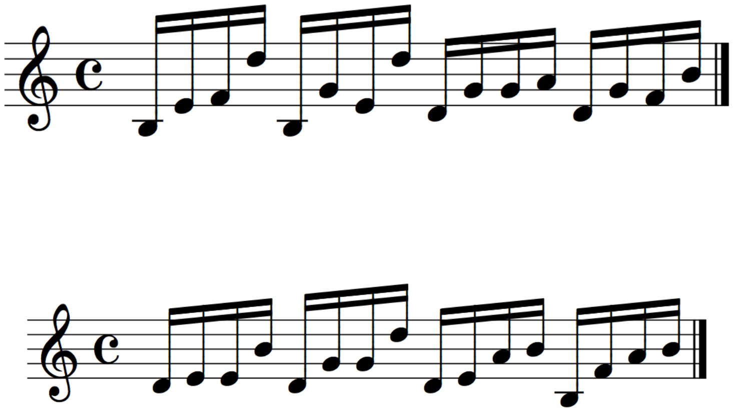


Figure 11: Examples of random permutations created with IGME.

8 Generative Techniques

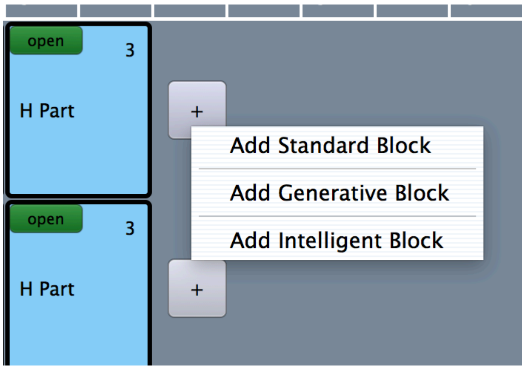
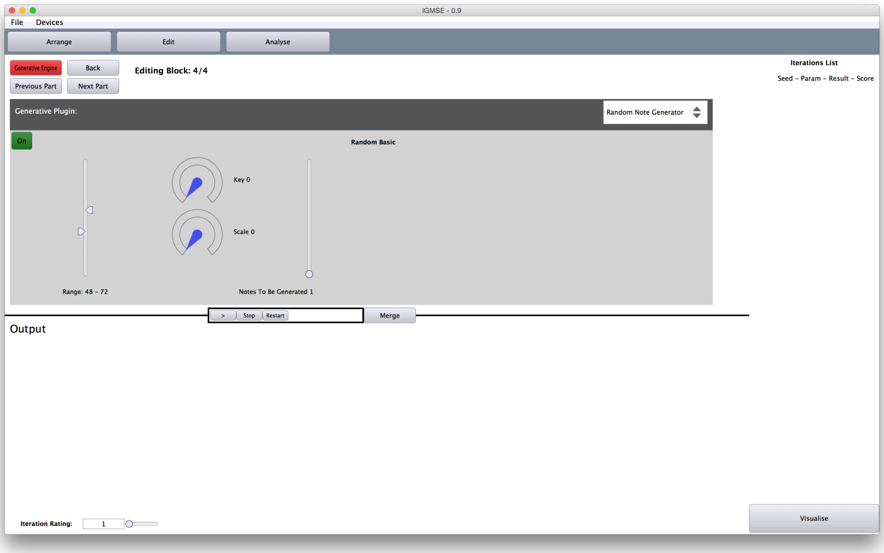


Figure 12: Adding a new generative block.

So far we have been editing a specific type of block referred to as **human/computer** block. That is a block that has both human and computer input to produce music. There is however another type of block that we can use, referred to as a **Generative** block. The music produced by this block is specified by controlling certain parameters, there is no input (seed) musical material.

To add a generative block, first return to edit view using the **back** button. Next push the **add** button at the end of track 1, and select **generative block** from the drop down menu.

Once again double click on the newly created block to open up the edit view (see figure 13). Select **random note generator** from the plug-in drop down menu. This view should look familar to the previous exercise. The **range** controls the range in which notes will be generated, **key** controls the key, **scale** controls the scale, finally **notes to be generated** controls how many notes get generated.



Editing Block: 4/4

Generative Plugin: Random Note Generator

Key 0

Scale 0

Range: 48 – 72

Notes To Be Generated 1

Iteration Rating: 1

Figure 13: Generative edit view.

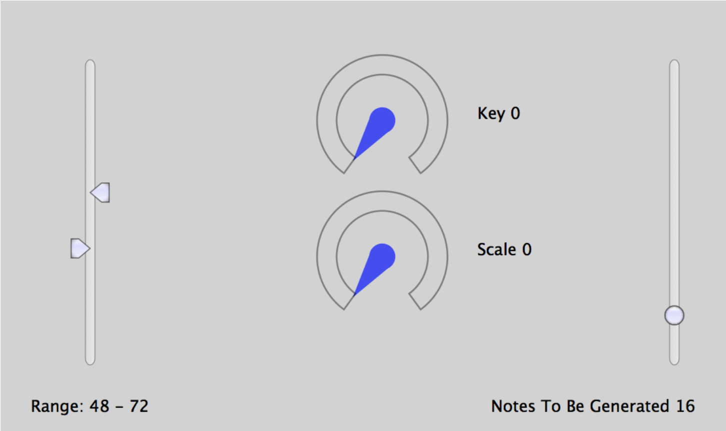


Figure 14: Suggested controls for random generative process.

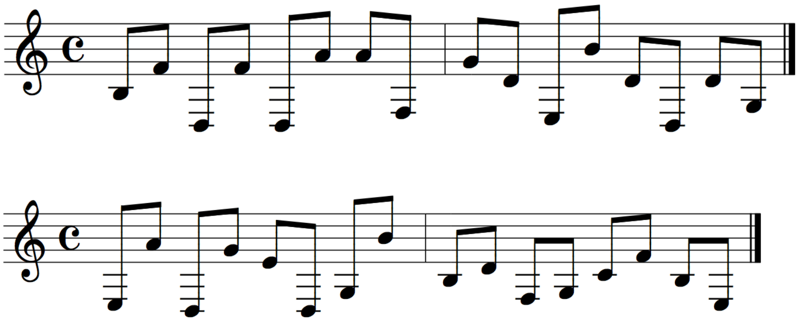


Figure 15: Examples of music produced with the random generative process.

It is also possible to convert musical content generated using a purely generative block into a human/computer block so that both further editing and generative processes can be applied. To achive this simply right click on the part in the arrange view and click "covert to human/computer part"

9 Survey

Thank you for taking part in the workshop. Before progressing further would you please take the time to fill out the supplied paper questionnaire.

10 Conclusion

We hope you have enjoyed using IGME during this workshop. Both constructive criticism and positive comments are welcome, and you are encourage to discuss your experience with the workshop tutor. In addition we ask you kindly fill out the supplied questionnaire.
Due to scope and size of IGME there are many more complex features that distinguish the program, that could not be explored inside the session. However if you are interested in learning more about IGME and wish to participate in more in-depth studies, then please feel free to get in touch.

G: Additional Publications

1. Tenor 2017: How Can Music Visualisation Techniques Reveal Different Perspectives on Musical Structure

HOW CAN MUSIC VISUALISATION TECHNIQUES REVEAL DIFFERENT PERSPECTIVES ON MUSICAL STRUCTURE?

Samuel J. Hunt
University of The West of England
Department of Computer Science
& Creative Technologies.
Samuel.hunt@uwe.ac.uk

Tom Mitchell
University of The West of England
Department of Computer Science
& Creative Technologies.
Tom.mitchell@uwe.ac.uk

Chris Nash
University of The West of England
Department of Computer Science
& Creative Technologies.
Chris.nash@uwe.ac.uk

ABSTRACT

Standard western notation supports the understanding and performance of music, but has limited provisions for revealing overall musical characteristics and structure. This paper presents several visualisers for highlighting and providing insights into musical structures, including rhythm, pitch, and interval transitions, also noting how these elements modulate over time. The visualisations are presented in the context of Shneidermans Visual Information-Seeking Mantra, and terminology from the Cognitive Dimensions of Music Notations usability framework. Such techniques are designed to make understanding musical structure quicker, easier, less error prone, and take better advantage of the intrinsic pattern recognition abilities of humans.

1. INTRODUCTION

Standard western notation serves as a strict, formal set of instructions for the performance of composed music. However, it omits explicit representation of a rich amount of hidden data that exists between individual notes, and the location of the notes within an overarching musical structure. One way to understand this structure is to analyse the music: either manually, requiring an experienced musicologist; or via computer, resulting in several multi-dimensional data fields, which may be difficult to represent and comprehend. Representing this data visually utilises the brains pattern detection abilities, supporting easier and faster comprehension of material to enable insight and speculation that can inform further formal analysis.

Visualisation presents non-visual data in a visual format, usually as 2D/3D images or video. Shneiderman [1] introduces a framework for guiding the design of information visualisation systems, known as the Visual Information-Seeking Mantra (VISM). The framework consists of seven tasks for presenting information in a visual form to a user (Table 1). Craft and Cairns [2] elaborate on this by stating the VISM serves as inspiration and guidelines for practitioners designing visual information systems. /par

Task	Description
Overview	Gain an overview of the data.
Zoom	Zoom in on items of interest.
Filter	Filter out uninteresting items.
Details-on-Demands	Selected an item or group and get details when needed.
Relate	View relationships between items.
History	Keep a history of actions to support undo, replay, and progressive refinement.
Extract	Allow extraction of sub-collections and of the query parameters.

Table 1. The 7 tasks of the VISM.

Shneiderman emphasises that humans have remarkable perceptual abilities, allowing them to easily detect changes of and patterns in size, colour, shape, movement or texture in visual media. Such advanced and robust feature extraction capabilities are considerably more difficult to encode as automated analysis using computer systems.

In a musical context, visualisers also enable rapid, automated methods for visualising not only a single piece of music, but an entire corpus - allowing understanding and comparisons of musical material at a higher and more generalised level to that of manual score analysis.

The level meter which features in the majority of consumer audio products, represents a ubiquitous visualisation method, whereby the current sound level is visualised using vertical bars, and for the majority of situations a more useful presentation than a display of audio sample values (amplitudes). Digital audio is stored as a series of numbers, a sequence of amplitude measurements with respect to time. Sonograms convert this information to visualise the distribution of frequency content. An example of this is illustrated in Figure 1, whereby the musical score has been synthesized using piano samples on a computer and analysed with a sonogram.

This paper focuses on visualizing scores at the note-level (e.g. MIDI), avoiding the many difficulties of audio feature extraction. Sequenced music, encoded as MIDI, by contrast allows for rapid and reproducible analysis [3]. The aim of the paper is to present novel techniques that support the analysis of music.

The remainder of the paper is broken down as follows.

Copyright: ©2017 Samuel J. Hunt et al. This is an open-access article distributed under the terms of the Creative Commons Attribution 3.0 Unported License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

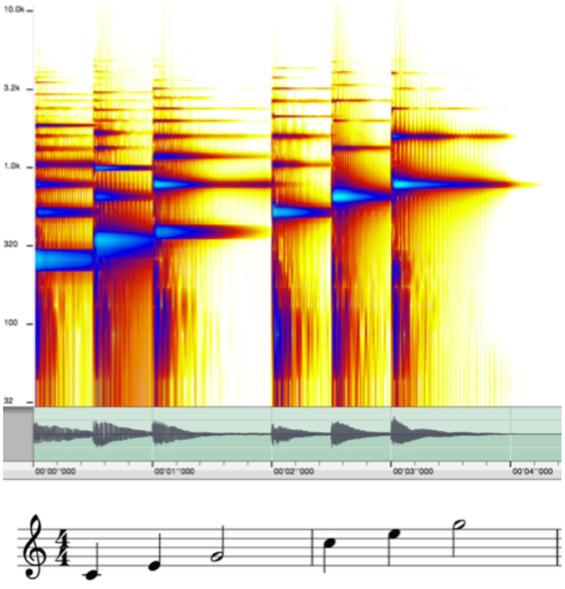


Figure 1. Sonogram plot of the score.

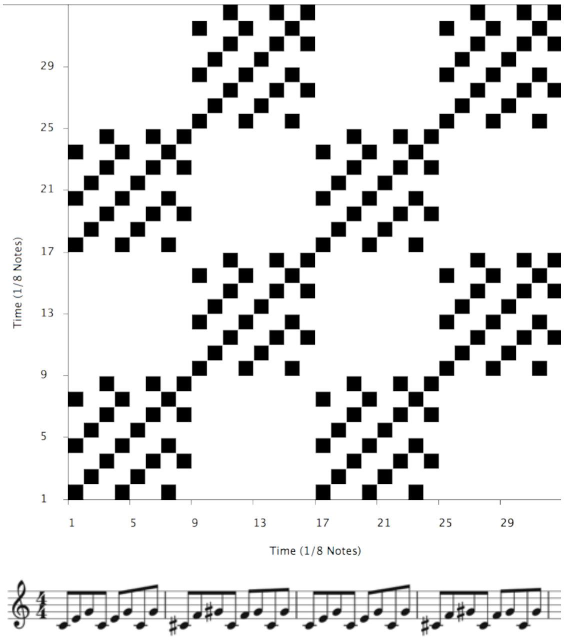


Figure 2. Self-Similarity visualisation of the score.

Section 2 presents relevant prior work and theory, followed by a brief discussion in Section 3 of the software system developed to support this research. Section 4 reviews visualisation techniques for pitch, contours, intervals and key, followed by Section 5 looking at rhythmic elements. Section 6 discusses visualisation techniques that integrate both pitch and rhythmic elements. The final section considers future work for the area and proposes evaluation techniques.

2. RELATED WORK

Prior work in music visualisation can be broadly categorised into two groups: those exploring sampled audio data and those exploring sequenced music data (scores and MIDI). Soriano et al [3] present methods for browsing an audio-based music collection, using graphical metaphors designed to convey the underlying song structure. This analysis is performed via feature extraction from MIDI files, enabling easy identification of simple and meaningful musical structure, such as pitch and rhythm.

Foote [4] and Wolkowicz & Brooks [5] both used self-similarity matrix visualisations to reveal similarity in music. This visualisation approach relies on the measurement of pitch content at quantised time intervals, and plotting this against all other intervals. Figure 2 shows a self-similarity matrix visualisation, whereby the music proceeds through time from the bottom left to the top right, with regions of similar patterns appearing as clusters of squares. Both axes represent the same input vector. The music example uses a repeating motif of one bar, with a modulation at bars 2 and 4.

Bergstrom [6] presents several visualisers that convey information about interval quality, chord quality, and the chord progressions in a piece of music, helping users to comprehend the underlying structure of music. Feedback from engagement with the system revealed users who having quickly understood the basics, wanted to compare music from multiple genres and composers. Holland [7] presents a similar system (Harmony Space) to allow beginners to interact with harmony using a visual grid.

Jeong and Nam [8] discuss a system that visualises audio streams, to show audio features such as, volume, onset density, and dissonance. The authors also state that as music is an auditory art, visual representations can contain information that cannot be transferred or perceived accurately with sound. Herremans and Chew [9] use visualisation to highlight tonal tension in music, creating an explicit representation of something that is not easily quantifiable, presenting graphics alongside the scored elements.

Established analytical frameworks for music, such as the Generative Theory of Tonal Music (GTTM) [10] and Schenkerian analysis [11], also present ways to annotate music and reveal structure. The GTTM proposes a series of preference rules for determining the different musical structures that underlie the perception of western music. Schenkerian analysis is an established musical analysis technique that aims to explicitly reveal hidden dependencies and structures implicit in the music. This analysis primarily aids score reading by marking it with elements of musical structure. Both of these theories have been mechanised in software [12][13].

Nash [14] presents research that adapts the Cognitive Dimensions of Music Notations framework (CDMN) [15], for use in designing and analyzing music notations and user interfaces for digital and traditional music practice and study. This paper utilises the framework as a vocabulary for comparing visualised music content and metadata against western notation and other forms of visualisation. However not all of the 16 core dimensions originally specified are of relevance here. A list of the terms and their definitions relevant in this research are listed in Table 2.

Using the self-similarity visualisation in Figure 2 as an

Dimension	Description
Visibility	How easy is it to view and find elements of the music during editing?
Juxtaposabillity	How easy is it to compare elements within the music?
Hidden Dependencies	How explicit are the relationships between related elements in the notation?
Hard Mental Operations	How difficult is the task to work out in your head?
Conciseness	How concise is the notation?
Provisionality	How easy is it to experiment with ideas?
Consistency	Where aspects of the notation mean similar things, is the similarity clear in the way they appear?
Viscosity	Is it easy to go back and make changes?
Role Expressiveness	Is it easy to see what each part of the notation means?
Error Proneness	How easy is it to make annoying mistakes?
Closeness of mapping	Does the notation match how you describe the music yourself?

Table 2. Terms of the Cognitive Dimensions of Music Notations framework used in this paper [14].

example of the terms used in the table, the visibility of the figure is good, showing a clear overview of the entire piece, likewise the juaxtaposability scores highly as the patterns can be compared much more easily than sequences in the score. There are high hidden dependences as the original information has been transformed, with each square representing a smaller amount of information. The simplistic nature of the visualiser scores high on provisionality, consistency and conciseness. The visualisation does not have any meaning unless related to the score, with the underlying notated elements looked up, so has a poor closeness of mapping. Comparing sequences using just the notation would require both hard mental operations, and would be prone to error (error proness), whereas the automated analysis used to build the self-similarity visualisation is easily reproducible and more accurate.

A core concept of visualisation for notated music is its ability to reduce the hard-mental operations arising from manual score analysis [14]. Computer aided analysis also reduces the error proneness of operations. Visualisation can remove un-needed details (filtering [1]) from the score, for example performance markings, therefore improving the conciseness of the results.

Temperley [16] [17] uses visualisation to inform, explain, and evaluate formal analysis by computer. Often using these techniques when analysing a large corpus of music, to immediately show data that would otherwise be difficult to extract from looking directly at the score, or in fact thousands of individual pieces. Temperley also uses these as a way of comparing and refining models for music analysis.

3. INTERACTIVE VISUALISATION

This paper discusses visualisers developed for an original software package (Figure 3), the design of which has been influenced by the seven principles of the VISM (listed in Table 1). In general, it allows different pieces of music in MIDI format, to be opened and visualised quickly, in order to support high provisionality and enable rapid experimentation with analysis techniques. The software can analyse and compare entire corpora or individual pieces, as well as sub-sections or voices (tracks). A historic list of analyses is kept so these can be recalled and modified, retaining low levels of viscosity and commitment, therefore further facilitating experimentation and evaluation (provisionality).

Software and automated analysis has the advantage of processing large amounts of data quickly (compared to manual techniques), but takes considerable amounts of time and care to design and implement. Visualisation tools, such as that described, allow a user to speculatively interrogate data, before committing to more detailed and formal music analysis methods, be they traditional (e.g. Schenkerian) or computer-based (e.g. machine learning see Section 4.4).

4. VISUALISING PITCH

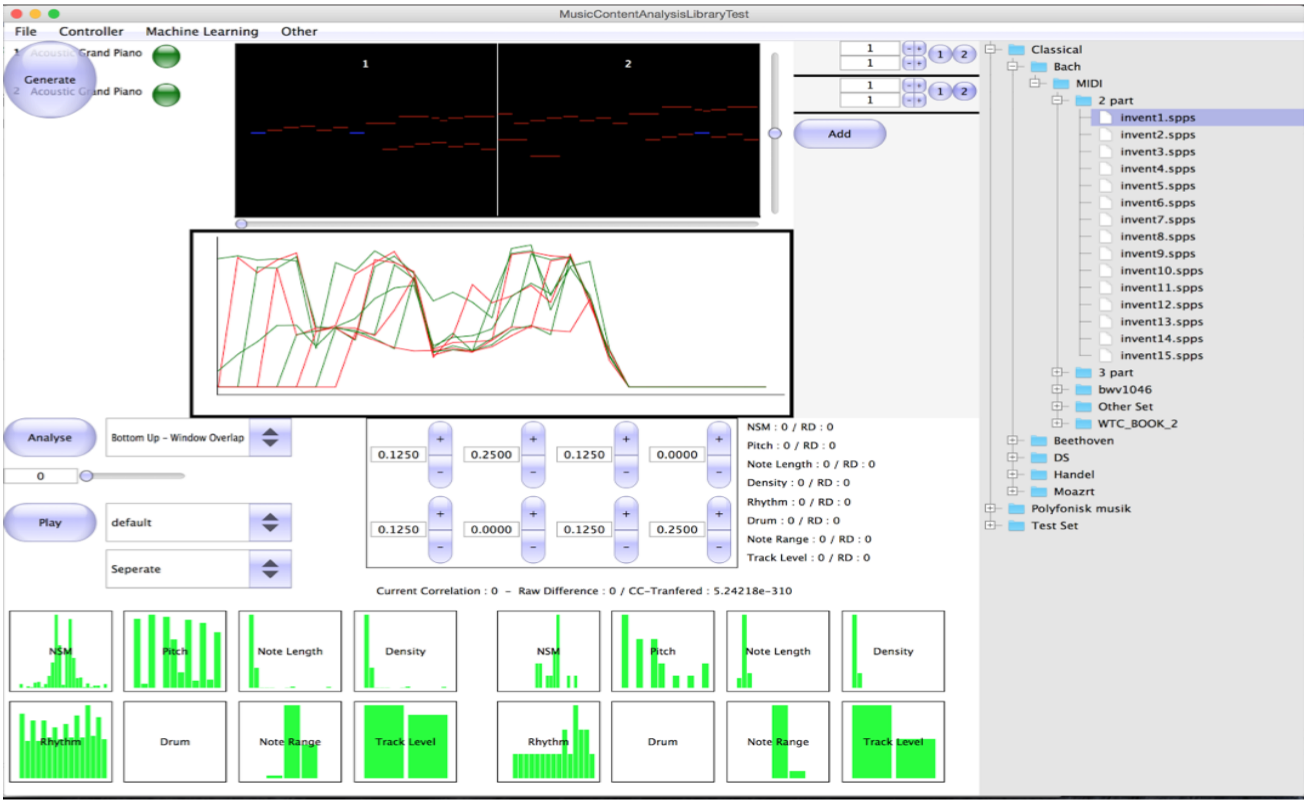
The set of visualisers presented in this section focus on elements of pitch, contour, and melodic interval. Some techniques present the material as overviews of the piece as a whole, others present excerpts in time. For the purposes of discussion and comparison, the majority of visualisations present Bachs Two-part Invention No. 1 (BWV 772) [18], but can be applied to many other examples and genres of music, including non-Western.

4.1 Melodic Contours

A contour representation of music can simply be defined as information about the up and down pattern of pitch changes, regardless of their exact size [19]. Melodic contours are also a key psychological part of music, one that aids the recollection of musical themes [20].

Melodic contours themselves can be illustrated using a score, where it is usually clear in which direction the pitch is going (Figure 4, top). However, once accidentals are introduced (Figure 4, bottom), it becomes less visually distinct. A piano roll (Figure 5) provides a clearer representation of melodic contour. This provides improved closeness of mapping [14], and increases the ease with which sequences can be compared (improving juxtaposability).Piano rolls provide ways for shapes, patterns and contours to be identified. Wood [21] presents related research in which the standard note head is visually modified to show the pitch degree in a more role expressive way, and reports improved speed for sight-reading when compared with standard note heads.

This type of visualisation can also be used to reduce a search space, allowing sequences represented as contours to be visually clustered. The items in Figure 6 show a series of monophonic melodies extracted from Bachs BWV 772. Visually, we can see that the first two patterns are



Bottom Up – Window Overlap

0

default

Seperate

0.1250 0.2500 0.1250 0.0000

0.1250 0.0000 0.1250 0.2500

NSM : 0 / RD : 0
Pitch : 0 / RD : 0
Note Length : 0 / RD : 0
Density : 0 / RD : 0
Rhythm : 0 / RD : 0
Drum : 0 / RD : 0
Note Range : 0 / RD : 0
Track Level : 0 / RD : 0

Current Correlation : 0 – Raw Difference : 0 / CC-Tranfered : 5.24218e-310

Figure 3. Software created to support visualisation tasks.

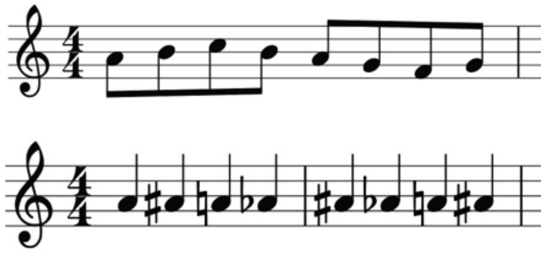


Figure 4. Score with clear melodic contour (top) and obfuscated melodic contour (bottom).

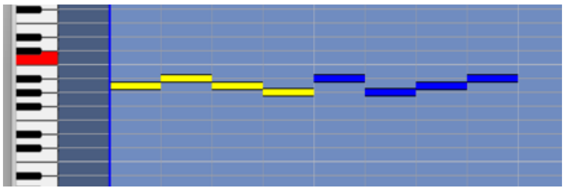


Figure 5. Piano roll representation of Figure 4.

similar, and that pattern 14 is the same pattern inverted. This kind of visualisation allows the viewer to employ the gestalt principles of visual perception, in this case similarity, to group together similar shapes [22]. In this situation the data has filtered out everything but the contour, giving a better overview of the types of contours, which can then be easily related against one another.

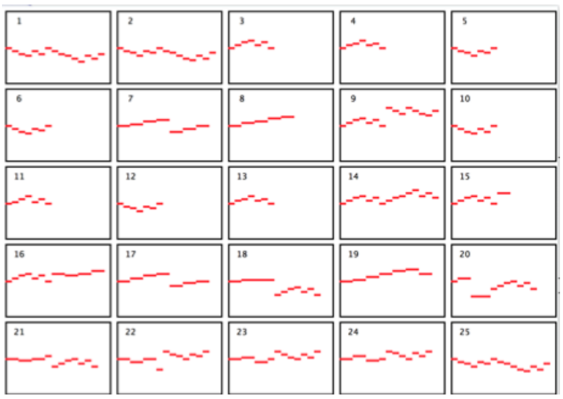


Figure 6. Selection of melodic contours from Bach's BMW 772.

4.2 Intervals

The contour plots provide an overview of the melodic patterns present in the music, but reduce the visibility and role expressivity of the intervals. Temperley [16] uses a histogram of melodic intervals to show the distribution of interval leaps between melodic note sequences within an entire corpus of music material, revealing wider patterns and trends in music. In-so-doing, this hides dependencies in the music, such as the local context and note-to-note relationships (i.e. certain pitches are more unlikely to transition to those depicted in the figure because of their relation to the home key and sensitivity to tonal context). The diagram in Figure 7 shows the interval profile for Bachs BWV 772.

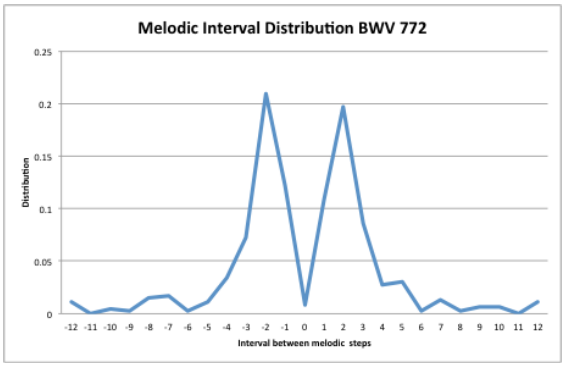


Figure 7. Interval distribution over two octaves in Bach's BWV 772.

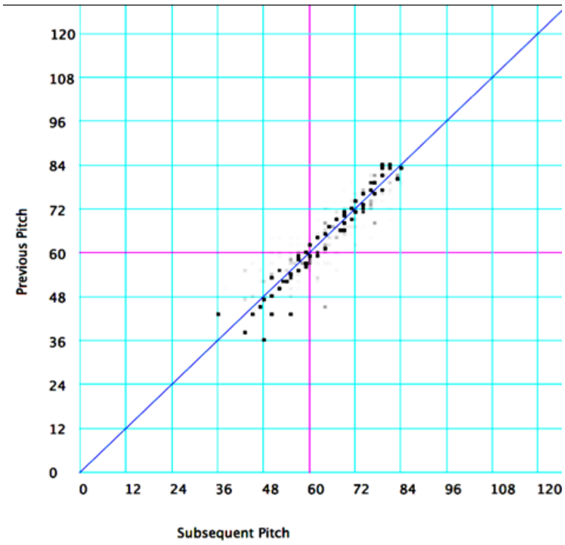


Figure 8. 2D Markov plot of Bach's BWV 772.

A different way to analyse this data, in a way that allows interpretation of pitch, intervals and range, is to use a Markov type model, defining the transition probability between any given notes, in a numeric table format. This, however produces a data table of size 127x127 elements, which is difficult to comprehend in a numeric format, but easily visualised to reveal musical trends and characteristics as illustrated in Figure 10. The design of this once again takes an overview of the data, filtering out the timed elements of the music, to give a detailed overview of the pitch and interval elements. Parts of the plot can be further inspected to reveal exact transition probabilities (details on demand).

From the plot, it can be noted that the intervals in the upper ranges are more likely to jump down in interval, while the opposite effect can be observed in the lower range. Towards the middle the width of the melodic jumps are slightly larger. The blue line along the leading diagonal represents the unison interval (repeated notes), the horizontal deviation from which reflects transitions to subsequent notes. The darker the marker, the more likely the transition. The diagram can also be thought of as a layered series of melodic interval distributions (as in Figure 7), given different starting notes (y-axis).

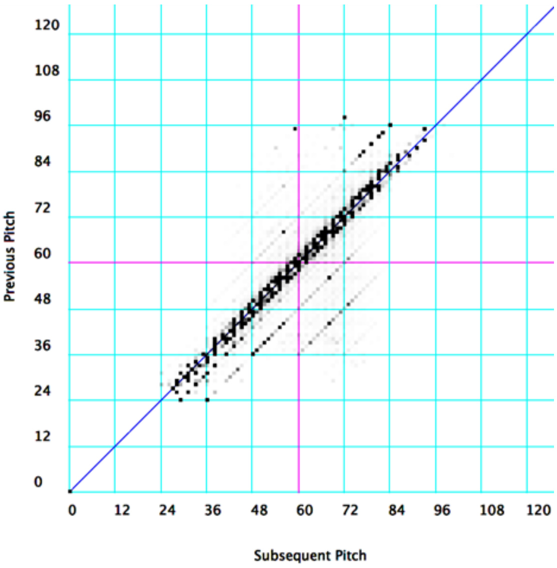


Figure 9. 100 randomly-selected common repertoire Baroque pieces.

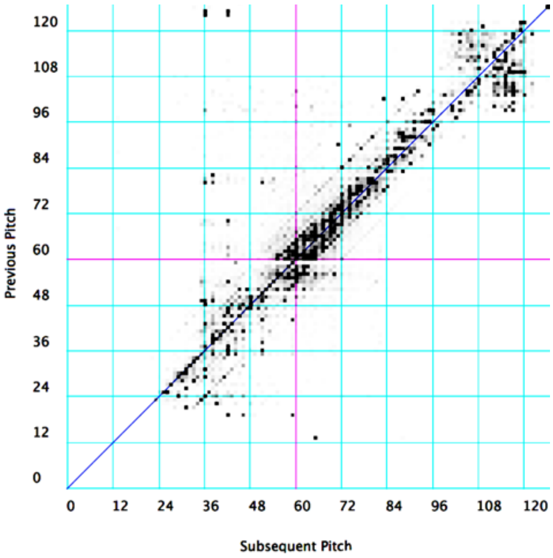


Figure 10. 100 randomly-selected common repertoire Jazz pieces.

Two more plots are shown in this style, but illustrating trends in, and differences between, larger corpora of music: respectively, a collection of 100 pieces of baroque music (Figure 9) and jazz music (Figure 10), selected randomly from a larger corpus. The visualisation process helps to reveal differences between the corpora that would otherwise be harder to discover or articulate. For example, the range of intervals in the jazz corpus is far wider, whereas the baroque is limited to mostly to an octave, and multiples thereof and appears more uniform throughout the range.

4.3 Pitch Distribution

It is instructive to consider pitch usage in general terms. Temperley [16] considers the distribution of pitches within a piece to be an intrinsic element that grounds the overall tonality and key in western music. Key is something that

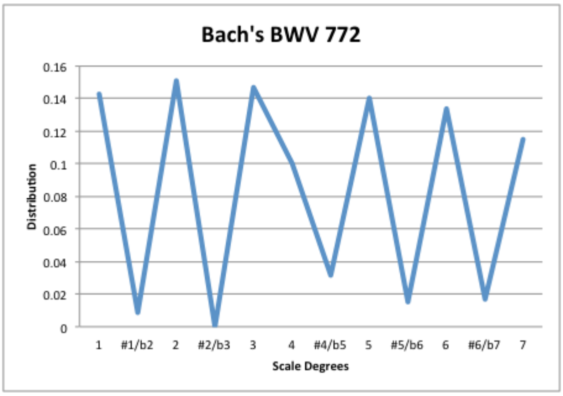


Figure 11. Major Key Profile.

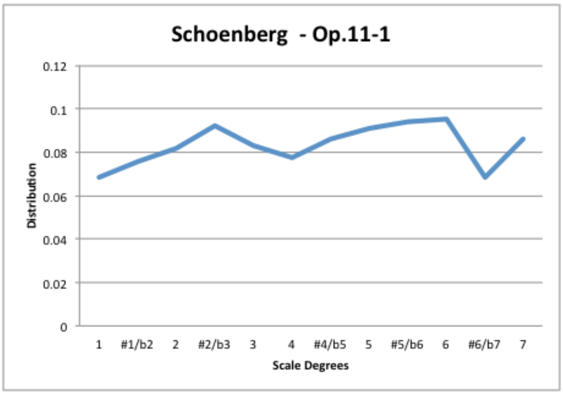


Figure 12. Pitch distribution in Bach's BWV 772.

musicians are trained to detect [16], but for which Temperley has developed automated methods. To illustrate, Figure 11 shows an ideal key profile describing the average distribution of pitches within a piece in C major, which can also be considered a coarse measure of pitch-class appropriateness in relation to key. For comparison Bachs BWV 772 (Figure 12) is also visualised. It is easy to visually infer the similarity of the distribution within the piece (known to be in the key of C) and the generalised representation (Figure 11). Smaller more nuanced details are also visible, such as the fact that the piece, although in C major, has more instances of D than the tonic C. Such details can be enough to fool automated analysis, as detailed in the next section, but things are clearer to the eye.

Other metadata can also loosely be inferred. A less pronounced distribution may indicate a piece that uses several different keys or tonalities beyond the diatonic. Atonal music, such as serialism, may confound such analysis and appear entirely different when visualised, such as Schoenberg Op.11-1 (Figure 13).

4.4 Key

Visualisation can help guide and test formal analysis. For example, a machine learning algorithm was developed that could infer the key based on the pitch profile of a piece. Bachs Well-tempered Clavier (Book 2) [24] was chosen as a test set, as it has two pieces in each of the 24 keys, providing an ordered pattern of tonality.

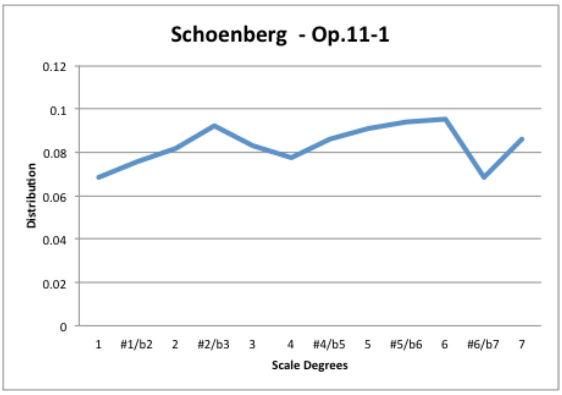


Figure 13. Pitch Distribution in Schoenberg Op.11-1 [23].

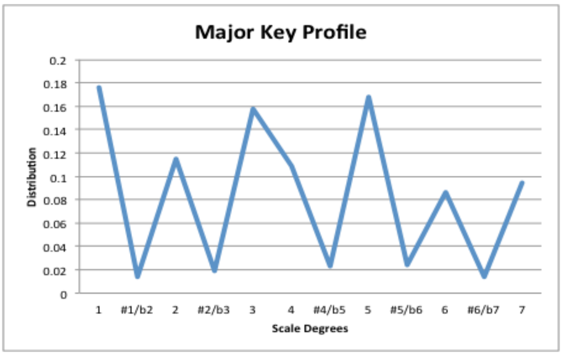


Figure 14. Pitch Distribution for Bach's BWV 870.

Figure 15 presents the detection results of the model, for each piece, ordered by their BWV number. The results of this experiment show that the algorithm is mostly able to predict each of the keys, and the graph can be inspected to find the relative confidence of each prediction as well as identify anomalies and deviations from the expected results. Bachs methodical progression through alternating major and minor keys within the collected work produces a visual pattern in the plot (discernable from the gestalt law of good continuance [22]), the deviations from which identify errors in the key detection model and, in turn, nuances in Bachs approach to key.

The algorithm makes three mistakes, out of a total of 48 predictions, corresponding to the anomalies circled in the figure. In one instance, the algorithm has predicted a key of D minor when the nominal key is C-Major. By visualizing the pitch profile of the piece (Figure 14), using the techniques suggested in Section 4.3 it can be observed that the overall ratio of pitch D, is higher than the tonic and 5th compared with an ideal plot (Figure 11), Indeed, this detection anomaly is attributable to Bachs actual use of D minor (and other keys) in the piece. This indicates a limitation of the analysis technique, in conflating the pitch profile of an entire piece without sensitivity to modulation, but nonetheless raises an interesting musicological question of why this and not other pieces from the set fall foul of this limitation.

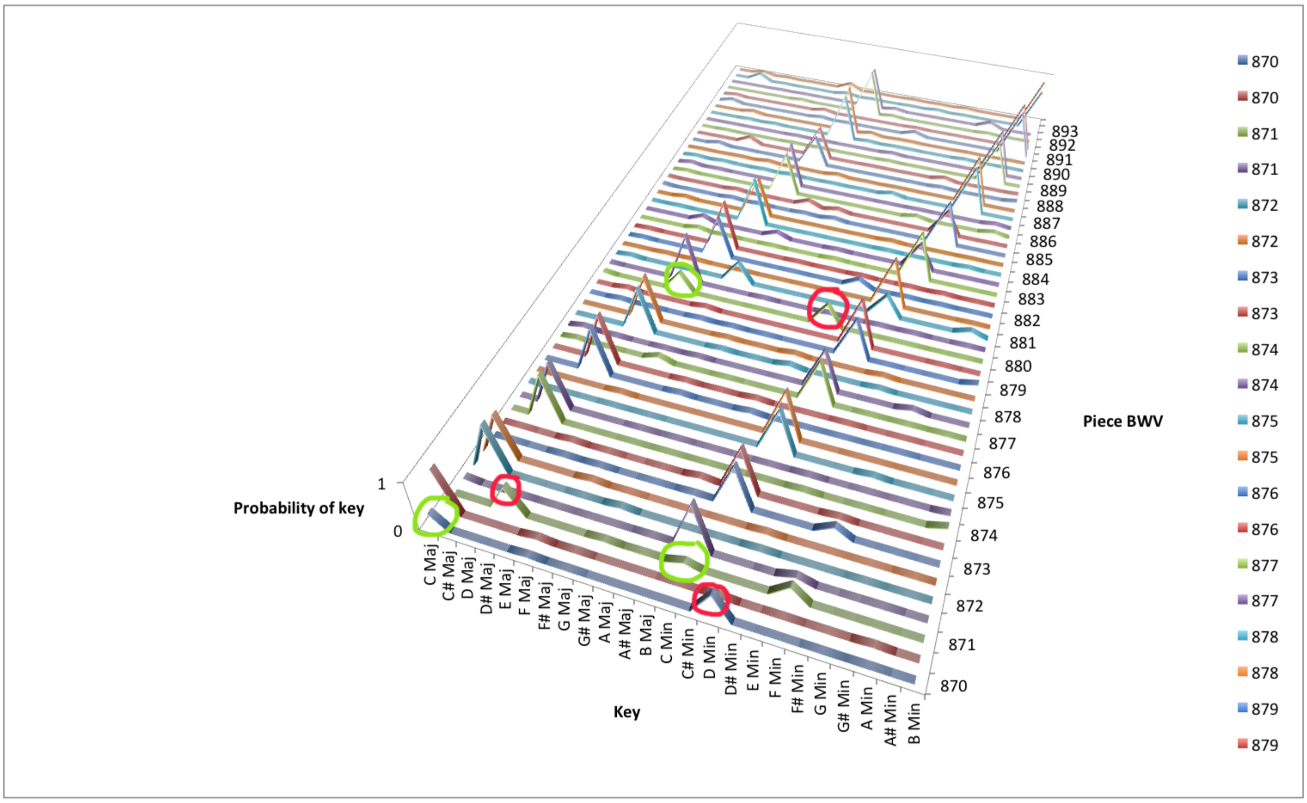


Figure 15. Visualisation of a machine learning algorithms prediction of the 48 pieces of Bachs well-tempered clavier book 2 [24]. The 3 mistakes are BWV numbers 870 part 1, 871 part 1 and 880 part 1. The red highlighting shows the mistakes and the green shows the actual keys.

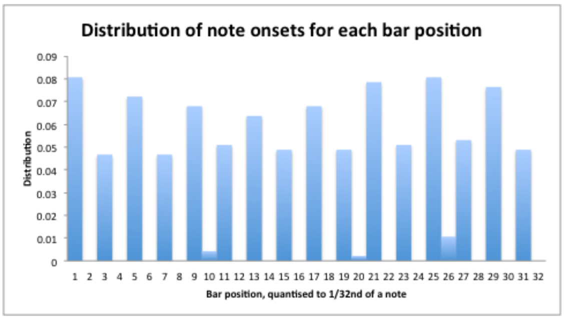


Figure 16. Distribution of rhythm for Bach's BWV 772.

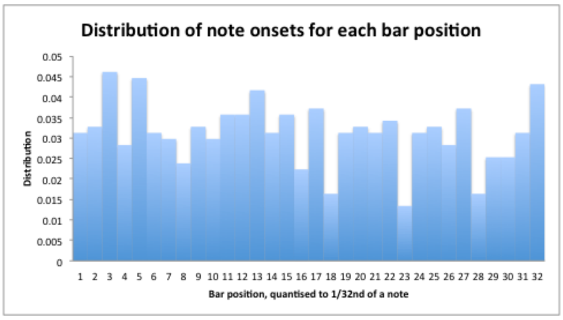


Figure 17. Distribution of rhythm for Beethoven's Op. 53..

5. VISUALISING TIME

Visualisation can also be used to reveal patterns in musical time, as in the case of rhythm, tempo, and density. Time also provides the metrical structure to a sequence of pitches. Taking the Bach piece BWV 772 as before, and visualizing the rhythmic aspects of the piece, several patterns are revealed. The elements under consideration are Note Onset, Note Length, and Density should be merged.

5.1 Note Onset

The basic rhythmic plot, note onset (Figure 16) shows the ratio of note onsets in each position of the bar for the entire piece. The events are first quantised to 1/32nd of a note, to remove noise caused by micro variations in time. The plot shows us, that simpler divisions of the bar are more likely to contain notes than more complex ones, shown by the regular distribution and preponderance of quavers and semi-quavers. The middle of the bar has the least note activity in general, whereas the 1st quaver beat, and 4th quaver beat have the most. Comparing this to Beethovens piano sonata No.21 Op. 53 (Figure 17), a piece from a much later period, shows a complete contrast in the structure, with a much more uniform distribution of note onsets, with the second semi-quaver bar position (3/32) being the most likely place for a note to be played.

5.2 Note Length

Note length visualisation (Figure 18) does not reveal as much information as some other techniques, but confirms

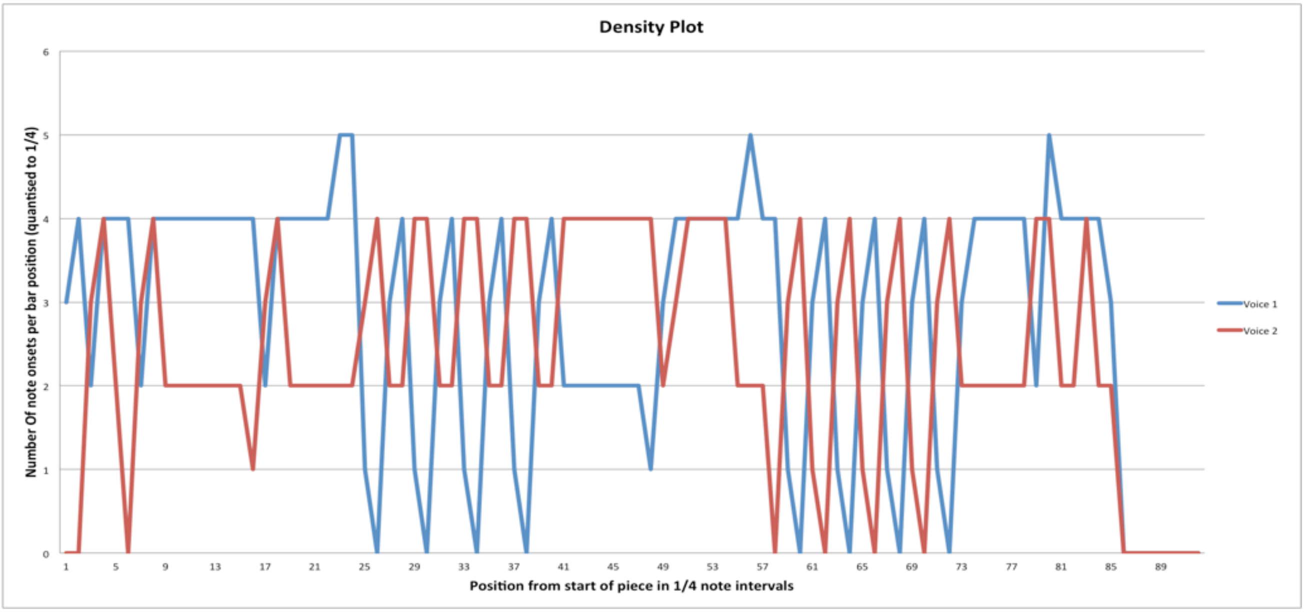


Figure 18. Software created to support visualisation tasks.

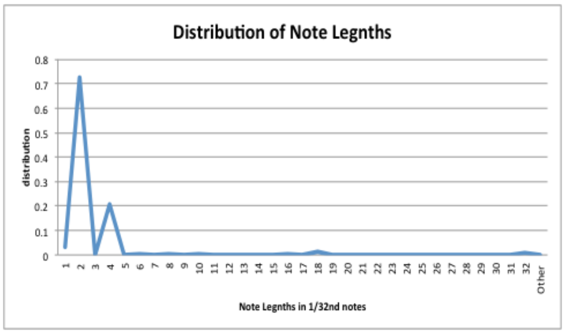


Figure 19. Distribution of note length in Bach's BWV 772.

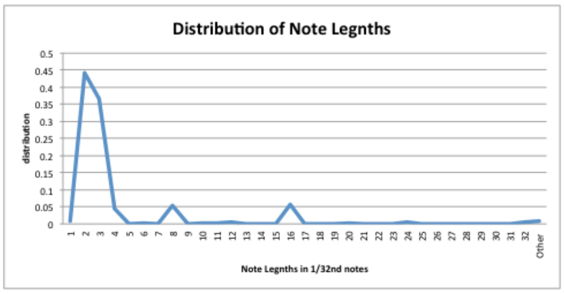


Figure 20. Distribution of note length in Beethoven's Op. 53.

this piece uses mostly note lengths of a semi-quaver in length. Roughly four times as many as using a quaver note. However, comparing this with other examples of music, for example Beethovens piano sonata No.21 Op. 53 (Figure 19), shows for example the use of a dotted semi-quaver (3/32) note length is more common than either a quaver or crotchet, and a value not even used in the Bach piece.

5.3 Rhythmic Density

Rhythmic density can be defined as the number of note onsets that happen during a beat or other window of time. The analysis is computed by calculating the number of onsets in each density window, and plotting the changes over time for each voice (note that only the first 12 measures are shown in Figure 20). Using Bachs BWV 772 again, several repeating patterns are visually observable between the two voices.

Figure 20 shows that only three of 48 windows have both voices indicating a density reading of 4 simultaneously. The sharp peak in Voice 1 at 23-24, is indicated as the most intense, a result of the piece using demi-semi-quavers (see figure 21). From windows 25 to 41, the voices are alternating in a strict pattern. This representation provides a concise overview, but does not differentiate between chords and rapid melodic phrases, reducing the visibility and juxtaposability of data. However, while a finer resolution could reveal more detail, it would also reduce conciseness, with four times as many data points. This represents a common trade-off between the dimensions, as observed in other notations [14].

In general, the techniques discussed in this section show that one method will reveal certain information at the sake of obscuring others, and that sometimes multiple perspectives are needed to fully understand the data.

6. INTERGRATED VISUALISATIONS

Previous sections considered elements of music in isolation, but visualisations can also reveal relationships between different dimensions of music. The ability to integrate musical characteristics and model the complex interwoven principles between them is a prime objective of music analysis and visualisation. The diversity and variety of such interconnections makes this difficult, but it is possible to combine multiple dimensions of characteristics to reveal more complex and interesting patterns.

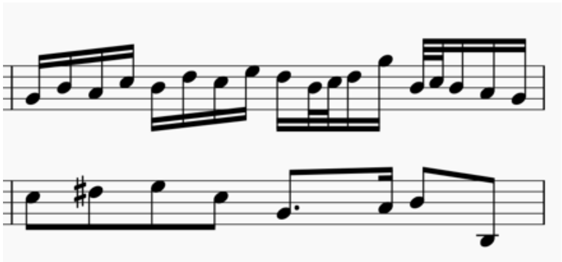


Figure 21. Demi-semi-quavers in bars 6. Relative to points 23-24 on figure 20.

Two related elements of music that can be integrated for visualisation and analysis are rhythm and pitch. A sequence of notes can be considered a pitch change after a given length of time, and it is possible to build up the frequency of these different event combinations and display the result. Given a standard composition the number of options is vast, and represents a complex problem. However, this is relatively easy to visualise (Figure 22) by plotting the change in interval against the difference between note onset, with the colour level (brightness) showing the ratio. In the example (Figure 22), a visualisation of Bachs Brandenburg concerto BWV 1046 [24] is shown, using this method.

Looking at the analysis, it is clear how consistent the timing of the piece is, with most events falling on quaver note divisions. There is some evidence of quaver-triplets as shown between 12 TPQ (Ticks Per Quarter Note or Crotchet) equivalent to a semi-quaver and 24 TPQ (Crochet), with these taking a value of 16 TPQ. Looking at the overall pitch range the widest range of pitch intervals is a note following on a quavers length after the previous note, with events ranging from +24 semitones, to -17 semitones. This is also where the most events are likely to be played, shown by the density of red dots. At the 1 and 2 semi-quaver duration (12 and 24 TPQ) the pitch is more likely to increase, on any value greater than this, the pitch is likely to decrease. At the semiquaver difference, almost all intervals are present, but compare this to longer duration differences, and intervals start to disappear. An interval change of +4 semitones (major 3rd) does not happen following a previous note whose duration was a quaver. This is quite possibly linked to the rules of strict counterpoint, a technique regularly employed by the composer, but further investigation is subsequently required before drawing specific conclusions. Finally, at the 3-semi quaver duration (32 TPQ) interval, a pitch increase is more likely, but at the crotchet level (48 TPQ) a pitch decrease is more likely.

7. CONCLUSIONS

This paper has reviewed a variety of basic music visualisations to demonstrate their utility to reveal implicit details, patterns, and structures in musical phrases, pieces and broader corpora. Although the visualisations have been informally evaluated with reference to the CDMN framework, another way to evaluate the use of visualisation is to establish whether or not it revealed something that was either not known before or complicated to reveal using other methods. As several of these techniques have made such novel observations about musical structure, they can therefore be considered successful.

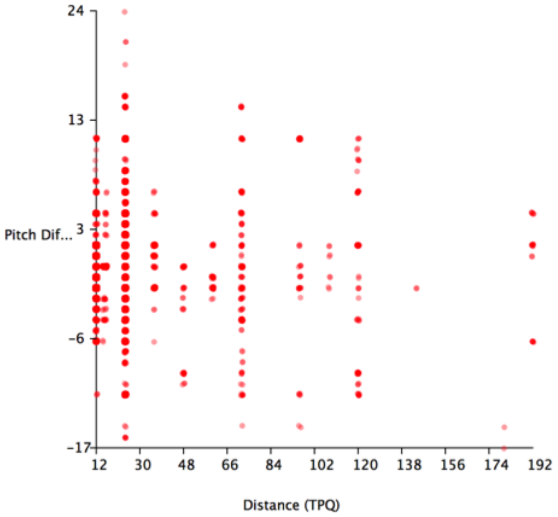


Figure 22. Visualisation of change in interval vs time between note onsets for Bachs BWV 1046. TPQ is defined as the number of ticks per quarter (crotchet) note.

Other further types of studies are also planned in this area, including embedding these visualisation techniques inside music composition software. Such investigations will explore the pedagogical benefits of alternative visual representations of music, looking at how visualisations can inform students understanding of musical process and structure.

Visualisation techniques can also inform the design of generative musical techniques. They allow the identification of characteristics that can become factors of a computer composition models, such as the parameters of a machine learning process. It also allows a degree of quantitative evaluation and comparison between music generated algorithmically and the target musical result. Vickery [25] advocates re-sonifying visualised music representations, formed through analysis of the original music.

While this review of visualisation techniques only scratches the surface of both visual and musical possibilities, it is clear the visual domain can be exploited to provide different perspectives on musical patterns and structures, and make hidden information and insights more accessible to musicians and scholars.

8. REFERENCES

[1] B. Shneiderman, “The eyes have it: A task by data type taxonomy for information visualizations,” in *Visual Languages, 1996. Proceedings., IEEE Symposium on.* IEEE, 1996, pp. 336–343.

[2] B. Craft and P. Cairns, “Beyond guidelines: what can we learn from the visual information seeking mantra?”

in *Ninth International Conference on Information Visualisation (IV'05)*. IEEE, 2005, pp. 110–118.

[3] A. Soriano, F. Paulovich, L. G. Nonato, and M. C. F. Oliveira, "Visualization of music collections based on structural content similarity," in *2014 27th SIBGRAPI Conference on Graphics, Patterns and Images*. IEEE, 2014, pp. 25–32.

[4] J. Foote, "Visualizing music and audio using self-similarity," in *Proceedings of the seventh ACM international conference on Multimedia*. ACM, 1999, pp. 77–80.

[5] J. Wolkowicz, S. Brooks, and V. Kešelj, "Midivis: Visualizing music pieces structure via similarity matrices," in *Proceedings of the 2009 International Computer Music Conference, ICMC'09*, 2009, pp. 53–6.

[6] T. Bergstrom, K. Karahalios, and J. C. Hart, "Isochords: visualizing structure in music," in *Proceedings of Graphics Interface 2007*. ACM, 2007, pp. 297–304.

[7] S. Holland, K. Wilkie, A. Bouwer, M. Dalgleish, and P. Mulholland, "Whole body interaction in abstract domains," in *Whole body interaction*. Springer, 2011, pp. 19–34.

[8] D. Jeong and J. Nam, "Visualizing music in its entirety using acoustic features: Music flowgram," in *in Proceedings of the International Conference on Technologies for Music Notation and Representation - TENOR2016*, Anglia Ruskin University. Anglia Ruskin University, 2016, pp. 25–32.

[9] D. Herremans, E. Chew *et al.*, "Tension ribbons: Quantifying and visualising tonal tension," in *in Proceedings of the International Conference on Technologies for Music Notation and Representation - TENOR2016*, vol. 501, Anglia Ruskin University. Anglia Ruskin University, 2016, pp. 8–18.

[10] R. Jackendoff, *A generative theory of tonal music*. MIT press, 1985.

[11] A. Forte and S. E. Gilbert, *Introduction to Schenkerian analysis*. Norton, 1982.

[12] M. Hamanaka, K. Hirata, and S. Tojo, "Implementing "a generative theory of tonal music"," *Journal of New Music Research*, vol. 35, no. 4, pp. 249–277, 2006.

[13] A. Marsden, "Schenkerian analysis by computer: A proof of concept," *Journal of New Music Research*, vol. 39, no. 3, pp. 269–289, 2010.

[14] C. Nash, "The cognitive dimensions of music notations," in *in Proceedings of the International Conference on Technologies for Music Notation and Representation - TENOR2015*, Institut de Recherche en Musicologie. Institut de Recherche en Musicologie, 2015, pp. 191–203.

[15] T. R. G. Green and M. Petre, "Usability analysis of visual programming environments: a 'cognitive dimensions' framework," *Journal of Visual Languages &amp; Computing*, vol. 7, no. 2, pp. 131–174, 1996.

[16] D. Temperley, *Music and probability*. The MIT Press, 2007.

[17] ——, *The cognition of basic musical structures*. MIT press, 2004.

[18] Y. Tomita. (2016) The inventions and sinfonias. [Online]. Available: http://www.music.qub.ac.uk/tomita/essay/inventions.htm

[19] T. Fujioka, L. J. Trainor, B. Ross, R. Kakigi, and C. Pantev, "Musical training enhances automatic encoding of melodic contour and interval structure," *Journal of cognitive neuroscience*, vol. 16, no. 6, pp. 1010–1021, 2004.

[20] R. Aiello and J. A. Sloboda, *Musical perceptions*. Oxford University Press Oxford, 1994.

[21] M. Wood, "Visual confusion in piano notation," in *in Proceedings of the International Conference on Technologies for Music Notation and Representation - TENOR2016*, Anglia Ruskin University. Anglia Ruskin University, 2016, pp. 230–309.

[22] K. Koffka, *Principles of Gestalt psychology*. Routledge, 2013, vol. 44.

[23] imslp. (2016) Schoenberg op.11. [Online]. Available: https://imslp.org/wiki/3_Pieces,_Op.11_(Schoenberg,_Arnold)

[24] D. J. B. Page. (2016) Dave's j.s. bach page. [Online]. Available: http://www.jsbach.net/midi/

[25] L. R. Vickery, "Hybrid real/mimetic sound works," in *in Proceedings of the International Conference on Technologies for Music Notation and Representation - TENOR2016*, Anglia Ruskin University. Anglia Ruskin University, 2016, pp. 19–24.

2. InMusic 19: Automating Algorithmic Representations of Musical Structure Using IGME: The Interactive Generative Music Environment

Automating algorithmic representations of musical structure using IGME: The Interactive Generative Music Environment

Samuel J Hunt, Tom Mitchell, Chris Nash.

Corresponding author email address: Samuel.hunt@uwe.ac.uk

ABSTRACT (228)

In this paper we explore the recreation of existing musical compositions by representing the music as a series of unique musical bars, and other bars that can be replicated through various algorithmic transformations, inside the Interactive Generative Music Environment software, or IGME. This re-composition approach is intended to explore whether the pre-existing music could have been created using the processed based approaches offered by the IGME software. If music can be expressed by algorithmic processes then we propose that original works of music can be expressed or created in the same way. Such a justification can provide a rationale for creating the unique compositional processes and workflows that IGME affords to those looking to compose with generative and algorithmic music techniques, and avoid many of the pitfalls of generative music.

Music can be imported into IGME and automatically analysed to find unique bars, and bars that have been transformed from them. The overall timeline can be visualised to quickly demonstrate the structure of the music, using colour to differentiate unique musical ideas, and arrow-arcs to show the relationships between different parts. Such a process reduces the overall entropy of the music data and provides an educational insight into macro level music structures. Each of the techniques are explained and examples given. In addition, data sets have been pre-computed for several genres of music, showcasing the distribution of different types of techniques.

1. INTRODUCTION

IGME (the Interactive Generative Music Environment) is a music sequencer that supports the exploration of generative and algorithmic music techniques. Unlike code or patch-based systems, it provides an easy to use interface for exploring generative and algorithmic music techniques, that is built on common music software paradigms. Many existing generative music systems use workflows that are not familiar to non-programmer music composers. A more detailed overview of IGME (previously named IGMSE) is given in (Hunt, Mitchell and Nash, 2017 and 2018). The core design principles of IGME are:

1. Integrates algorithmic techniques for musical composition inside familiar score editing and music sequencing workflows.
2. Provides full version control, for revisiting and comparing material.
3. Uses graphical controls (WIMP) rather than code-based interfaces.
4. Takes a modular approach to composition, while retaining a linear timeline.
5. Uses a multi-layered assembly stage that assembles the final score from individual parts.

An impediment of generative music systems is that they often fail to form high level structure, and are often highly stochastic in nature (Hunt et al, 2017). This is seen in large existing systems such as Jukedeck (Langkjaer-Bain, 2018), Aiva (Zulić, 2019), and Melodrive (Collins, 2018) focusing on replacing the human completely, with cutting edge machine learning. The, overarching aim of IGME is to create a system that supports human and computer composition. The aim is that by combining the best aspects of generative music with the careful control of a human operator that more structured forms of generative music can be created. Therefore, it is worth considering how much of the music should be unique and how these ideas should be developed through the piece. Therefore, the principal aim of this research is to assess whether existing music (composed by humans) can be encoded and represented by algorithms using the tools afforded by IGME. From this we can understand what techniques other general music sequencing software should adopt, for supporting interactive generative music.

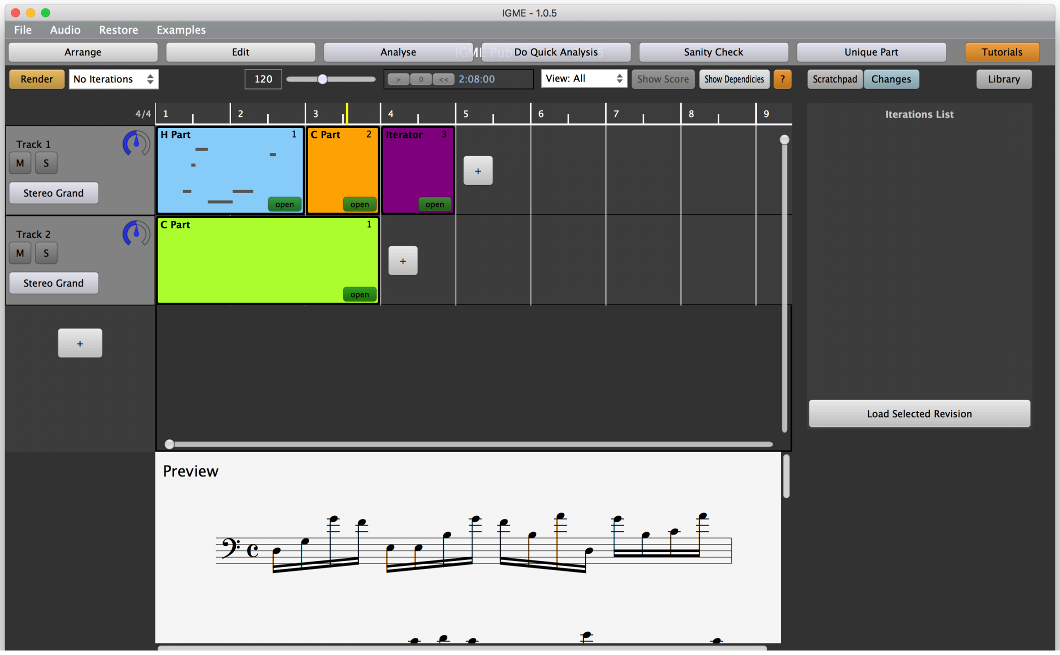


Figure 1: Arrange view inside IGME

IGME considers composition in terms of three distinct musical parts: human created content, computer generated content, or a mixture of both. A part within IGME is similar to the idea of a MIDI clip in other music software (the differences are discussed in section 3). The IGME program is divided into two main views: the arrange view (Figure 1) and edit view (Figure 2). The arrange view focuses on arranging and sequencing individual parts (e.g. MIDI clips), using design principles found in other common music sequencers. The edit view (or detail view) allows the user to edit the individual music sequences, and/or specify the algorithmic effects for each part. A range of algorithmic effects are implemented by IGME, that can either augment human composed music, or generate computer created music.

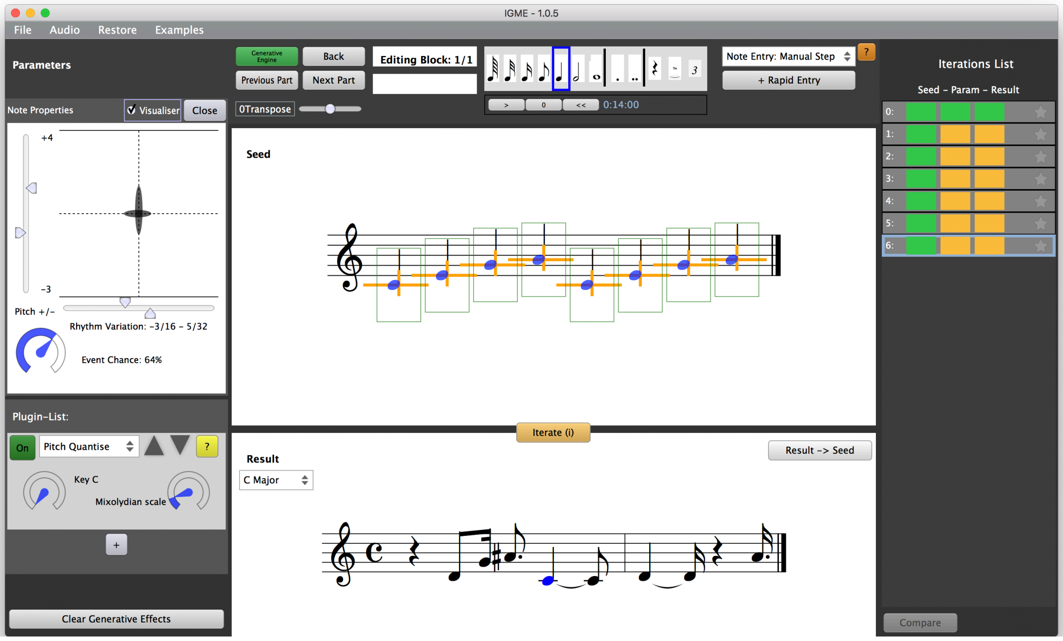


Figure 2: Edit view inside IGME.

The aim of this research is to look at how existing pieces of music can be represented and encoded using the IGME environment. The paper therefore considers two differing but similar research tasks. The first explores specific examples of music in detail, whereby the music can be represented more closely by; original ideas, and a series of transformations. The second looks at automatically analysing larger datasets to provide generalised metadata about musical structure. The concept of expressing music as patterns and processes has been explored previously by Nash (2014) using the *Manhattan environment,* and shares a number of parallels with this work.

The main body of text is broken down into five sections. Section 3 explores some of IGME's unique features that are crucial to this research. The various techniques for transforming and relating groups of musical parts are explained in section 4, each shown with examples of this process. Following this section 5 discusses the data pipeline for computing analysis automatically. Section 6 examines the output of a complete analysis of a single song, revealing insight into its musical structure. Lastly datasets are computed for various types of music and these are summarised in section 7.

2. BACKGROUND

In general, musical structure is often composed by following some set of *rules*. Moore (2001, p 433) notes that it is these rules that characterise different genres of music. Rules can be understood in terms of stylistic choices that determine or constrain elements of the music which are often inferred rather than formally defined (Herremans and Sörensen, 2012). For example, most tonal music is constrained to a given musical key (Temperley, 2007). However, certain genres of music, for example music composed using species counterpoint have well defined formal rules (Fux and Wollenberg, 1992). Furthermore; Minimalism, Serialism and other process-based forms of music are a genre of music that focuses on representing music composition as a process or series of algorithms, the research here considers music that does not identify as belonging to such a genre. In a general sense, this research attempts to find patterns in the structure of composed music.

Lerdahl and Jackendoff's (1983) generative theory of tonal music (GTTM) organises music into a four-level hierarchy; motives, phrases, periods, and larger sections. Several authors have attempted to automate the GTTM, notably Hamanaka (2006), however a full automated implementation of the GTTM remains unexplored. Rothfarb (2010) notes that the phrase level generally considers music to be 4 measures in length. The research here considers segmenting music mostly into motives, where the smallest division of hierarchy is fixed to a single measure.

The *Manhattan* music programming environment (Nash, 2014) uses a pattern-based sequencer paradigm in which code is situated in repeating musical patterns to manipulate the music during playback, as an explicit interaction model that considers music as the synthesis of patterns and processes, sympathetic to the key roles of rules and repetition throughout musical practice and history. Through a series of studies (études), the tool has been used to encapsulate pieces across various genres and eras of Western music (baroque, minimalism, romantic, popular, etc. – from Bach to Stravinsky to Hendrix) – recomposed as expressions of arranged patterns (musical seeds) and transformative or generative processes (procedural code). Used currently as a pedagogical tool, this model is designed to foster analytical thinking in students through manual analysis and reinterpretation through code, but is also the basis of other work on automated analysis and the practical exploration of data models in music.

Formal frameworks for analysing music have existed for a long term, notably Schenkerian analysis. Despite work by Marsden (2010), traditional Schenkerian analysis has only had limited success in being automated and remains too computationally expensive. More cutting-edge research in machine learning has explored automated music analysis in other ways (Huang, 2016). Deep learning, despite its promises of delivering exemplar solutions to the problem, provides a ‘black box’ approach that provides little metadata explaining the process, which is both important and useful for music. Rudin (2019) notes that more emphasis should be placed on making interpretable models for big data, rather than using black box algorithms.

Notable researchers, such as David Cope (Cope, 1991, 1996), has produced multiple works in the area of list programming for generating music. Starting with smaller fragments of music and combining them through various procedures to produce larger works. This work takes the opposite approach (starting at the end result, and working back to the start).

3. TERMINOLOGY:

A part in IGME is very similar to a MIDI clip in other musical sequencing software, however an individual part in IGME is made of 3 distinct sub-components. These are the seed, parameters, and result. The seed is the musical material that is edited by the user. The parameters are a series of processes (effects and algorithms) that are applied to the seed, to produce a result. Note the result is the musical material that is audible to the user. Without specifying any parameters, the result is identical to the seed. The seed material can also be supplied from a previous part’s result (discussed below) or by a seed generator (generative effect).

A reference part in IGME is whereby the content of a given part is referenced (or taken) from another part. In this relationship the seed material of a given part B is specified from part A’s result, therefore part B is referencing part A. Note that the reference part can have exactly the same content as its parent, or modify it (through various transformations). Looking closely at the score in Figure 3, the second bar is a direct duplicate of the first bar. Therefore, inside IGME part 2 could be notated as a part that references (in this case) part 1 (Figure 4 middle). This representation shows more explicitly the structure of the music. This could also be expressed as a *repeat* as shown in Figure 4 right. However, this common music sequencing paradigm fails to work when a bar of music is repeated in a non-consecutive manor, as shown in figure 5. Referencing can be used to represent musical structure in a more visual way, and is therefore argued as crucial concept for this work.

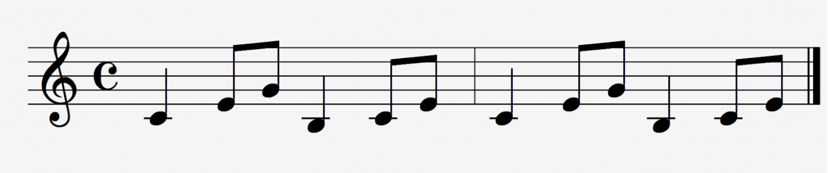


Figure 3: musical score.

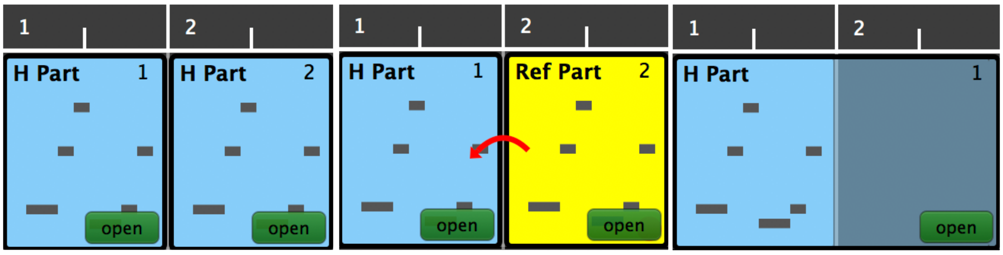


Figure 4: Left: Two parts the same. Middle: Part 2 referencing part 1. Right: Part 1 repeated once.

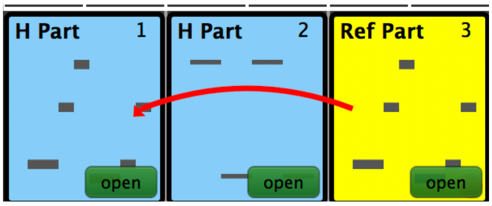


Figure 5: Part 3 is referencing part 1.

With reference parts there is a one-to-many mapping, whereby an individual part may be referenced many times, in figure 6, part 1 is referenced by parts 2, 3, and 4.

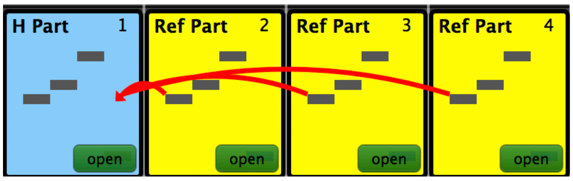


Figure 6: One part being referenced multiple times.

4. TECHNIQUES

IGME is a music composition environment, and consequently there are a range of tools for generating music through both stochastic and algorithmic techniques. Many of the stochastic techniques are not relevant in this research, as existing music cannot be expressed statistically, as the musical decisions would have been fixed during composition. Instead, a subset of the tools offered by IGME are used to determine and express musical structure. Namely the following techniques; duplication, transposition, transformations, arpeggiation and note-mapping can be automated. Each of these tools will be described in the next section alongside a working example. All of these techniques (except arpeggiation) make use of part referencing, whereby a part's initial content is taken from a previous part and then has some further process applied.

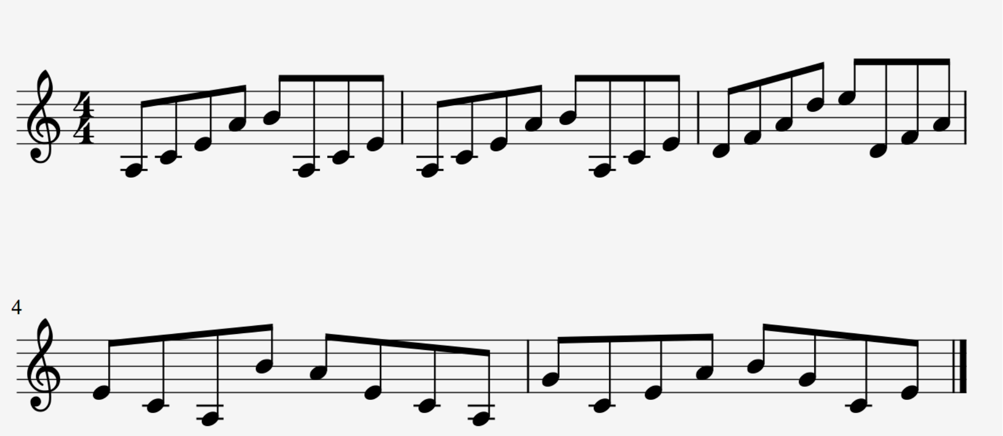


Figure 7: Score for the examples shown in this section.

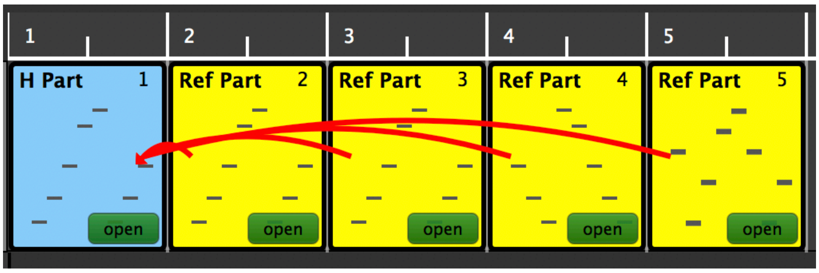


Figure 8: IGME part representation of the score in figure 3.

4.1. DUPLICATION

Is the technique of duplication (repeating) previous musical ideas, these are expressed in IGME through simple references (previous section).

4.2. TRANSPOSITION

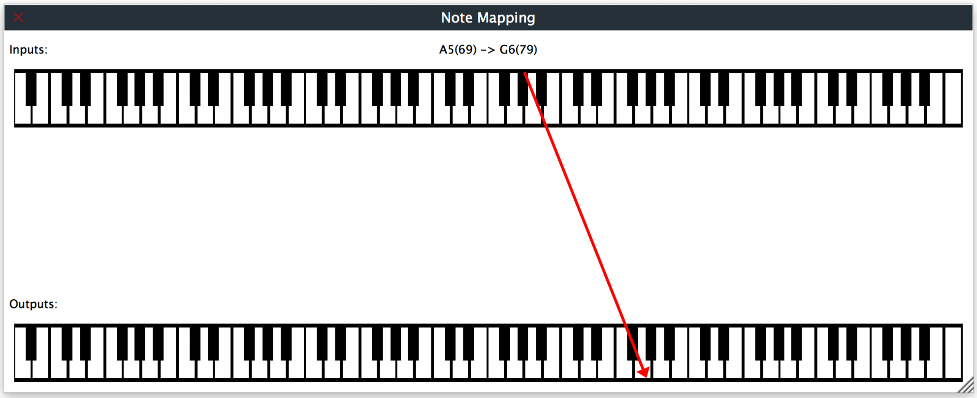
Transposition is simply the process by which all notes are chromatically transposed by a given value. In Figure 7, the second third bar of music is the first bar of music repeated and transposed by +5 semitones. Therefore, more structural semantics can be shown, if this is expressed as a reference part with a transpose process applied.

4.3. TRANSFORMS:

A transform process applies one of 4 simple procedures to a given part, these are; retrograde (playing the sequence backwards), inversion (inverting the pitches), retrograde-inversion (both together), rotate left or rotate right (note that rotating left and right together is nullified). In Figure 8 part 4 is set to reference part 1 and then have the retrograde plugin added.

4.4. NOTE MAPPING:

Note mapping is analogous to find and replace. A note map is simply a mapping that defines what notes in one sequence are replaced by in another sequence. For example, in figure 7 the pattern of notes in bar 5 is similar to bar 1, however the 2 A4's are replaced with G5's. This is expressed with a reference part and the note map plugin applied (Figure 9). With this process all occurrences of note A4 are replaced.



A5(69) -> G6(79)

Figure 9: note mapping interface.

The famous guitar hook introduction of Guns N' Roses' *Sweet Child O' Mine* (1987), provides ample opportunity to demonstrate the note map technique. The arpeggiated sequence of notes in bar 1 repeats in a block of 8. Bars 3 and 4 take the initial idea and replace the low D with an E. Bars 5-6 substitute the same note with a G, a score is provided in figure 10. Applying the note map process for bars 3 and 5, and applying reference duplication for bars 2,4,6,7, and 8, we end up with just 1 unique part, and 7 bars of transformations. This is visualised in Figure 11.

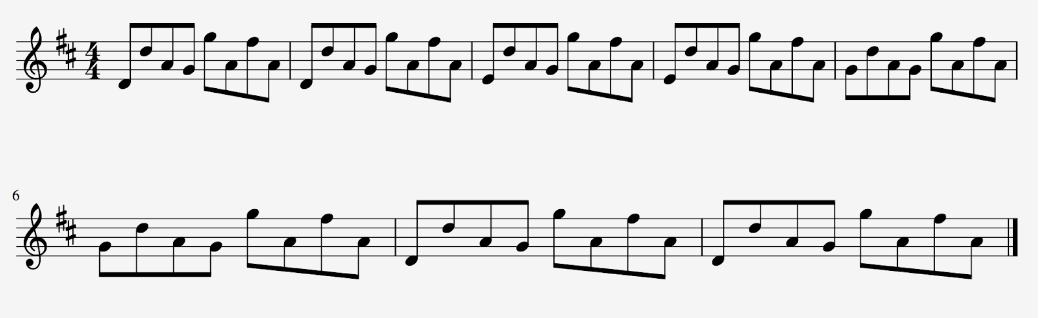


Figure 10: First 8 bars from Sweet Child O' Mine.

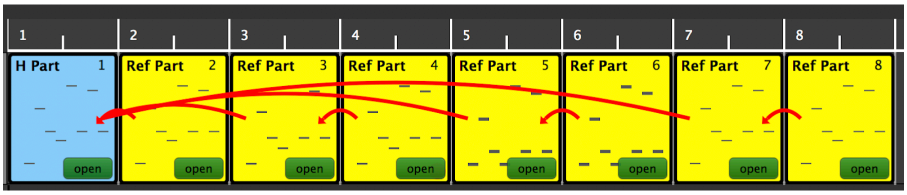


Figure 11: IGME representation of Figure 14.

4.5. ARPEGGIATION:

Unlike the other techniques discussed so far, arpeggiation attempts to reduce the overall musical data in a single part by expressing it as a collection of pitches, and the settings for an arpeggiator. For example, the sequence in Figure 12 can be encoded as 4 notes and the arpeggiator plugin with up as the play order, 1/16 for the speed, for 1 bar, and in 1 octave. Figure 13 shows the editor set up in IGME to replicate this.

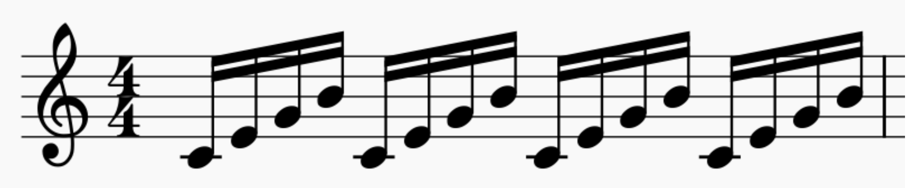


Figure 12: Simple arpeggiated idea.

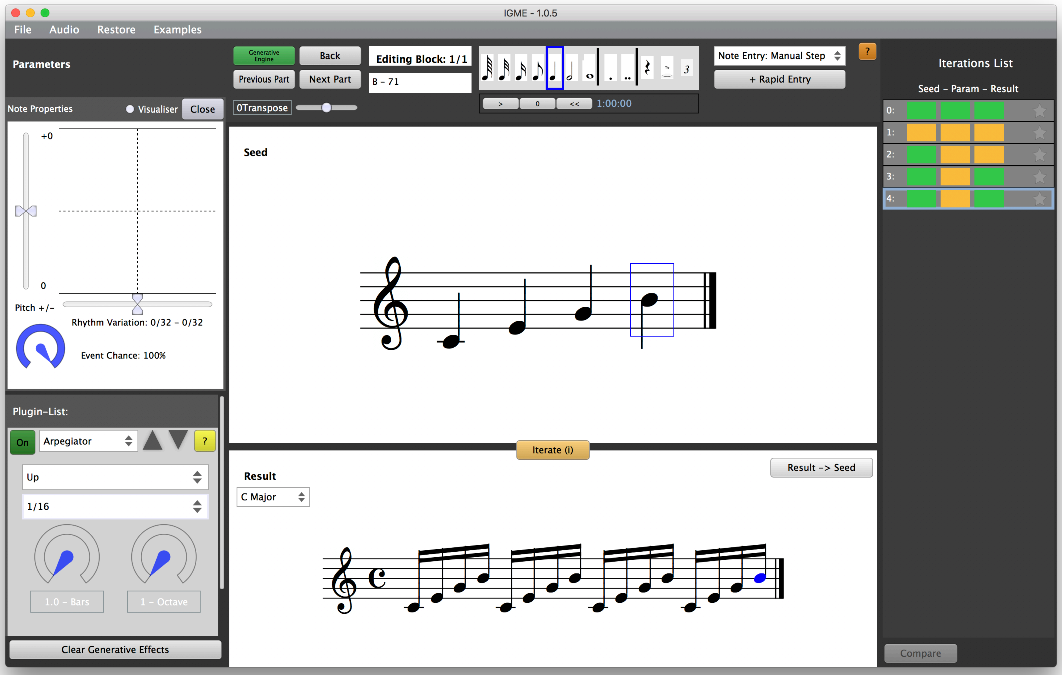


Figure 13: Part with an arpeggiator plugin applied.

5. DATA PIPELINE AND AUTOMATED ANALYSIS

An individual song in IGME is analysed by the following automated procedure. A song selected for analysis is first imported in MIDI format and decoded into IGME's internal representation (note that for the purpose of this study songs are limited to those in a 4/4-time signature, discussed further in section 8). Each MIDI track is given a track comprising of a sequence of individual parts. A track is split into parts based on bar lines, so the smallest possible part is a single bar, notes that cross bar lines are expressed such that the part takes up multiple bars of music. Note length and note onsets are quantised so that they are rounded to the nearest 1/32 note. Other details are lost by the process such as dynamic markings. Without making these modifications the complexity of doing this analysis would be implausible.

The general expression of two parts A and B is the relationship that relates B to A. This process therefore tries to find the set of procedures that modifies part A so that it produces the exact same musical output as part B. The processes outlined below automate this process using the techniques (discussed in section 4) to discover musical structure.

5.1. DUPLICATION ANALYSIS

The duplication analysis tried to find and group parts that have identical content. This works from left-to-right from the first track to the last. The process starts by taking the first part on track 1, and comparing it with every other part on the timeline. Note only parts that have the same number of events are compared, greatly reducing the overall complexity.

As the process works from left-to-right the overall number of comparisons decreases. When the first track is complete the process repeats starting with the first part on track 2. Only parts on track numbers greater than the current track need to be compared. When a match is found a reference is made between the two parts.

5.2. TRANSPOSITION ANALYSIS

The transposition analysis is similar to the duplication analysis, but the part is chromatically transposed incrementally from -12 semitones to +12 semitones before being compared with other parts. Essentially running the duplication test 24 times. This is expressed as a reference part, with a transpose plugin.

5.3. TRANSFORMS ANALYSIS

The transform analysis stage checks to see if the relationship between the two clips can be represented by a simple musical transform. This process is quicker than the first two analysis stages as parts that have already been marked as duplicate or transposed are removed from the task queue. Only parts that have the same number of events are compared, as the transformation process does not alter the quantity of different events. Given two parts, A and B, this process would iteratively compute the 4 transforms on part A, and check to see if the output matches part B, if this is true then part B is set to reference part A, and the relevant transform parameter is added to part B.

5.4. NOTE MAPPING ANALYSIS

This process looks to see if a given part can be represented by referencing another part, and substituting certain notes so that the result is the same. A number of checks are first made, these include, ensuring the two parts have the same total number of notes, the same rhythmic structure, and the part cannot be expressed by other simpler techniques. A list of notes that occur in each part are first computed, (we will call these list L1 and L2). A set of possible combinations are computed, by iteratively taking a single note from L1, and each note from L2, whereby the total number of comparisons is the size of L1 multiplied by the size of L2. A recursive function is then used to test each of the transforms on sequence A (original) and comparing it with sequence B (target). If a match is found the function exits and returns a list of the note mappings that transform part A to part B. If this process is successful then part B is set to reference part A, and the relevant note map parameter is added to part B. The note mapping process is CPU intensive and is run last.

5.5. ARPEGGIATION ANALYSIS

The arpeggiation analysis checks to see if a given part can be expressed as a smaller set of core notes and settings for an arpeggiator. The automated analysis first takes a given part and removes all the duplicate notes from the sequence, therefore leaving only a set of unique notes. The arpeggiator effect is then added, and the settings are iteratively worked through. At each iteration the output is computed and compared to the original, if it matches the original then the part is converted to an arpeggiated part.

5.6. OVERALL ANALYSIS

Pieces within IGME can either be analysed individually or in bulk. When computing an individual analysis, the parts are given a unique colour and the entire composition can be visualized. As an additional feature, once the analysis of the piece is computed. IGME can remove all but the unique ideas, therefore revealing just the raw building blocks that make up the rest of the song.

The analysis computes and represents the overall music into 2 overall categories, these are unique parts and representable parts. Within representable parts, several variants are grouped, these are duplicated parts, transposed parts, transformed parts, arpeggiated parts, and note map parts.

Duplication has a higher priority than the other techniques. In many instances a part can, be expressed as either a duplication of the same part previously, or as a transposition. For example, in a given 4 bar section there might be 2 unique parts and 2 parts that are expressed as transpositions of the first 2. When this 4-bar section repeats again, the 4 parts would be expressed as duplications in respect to the first 4 bar section.

6. EXAMPLES

6.1. SECRET OF THE FOREST

To give an overview of the automated analysis process discussed previously (before discussing analysing large corpuses), this section looks at a single piece of music in detail. Secret of the forest is a song composed by Yasunori Mitsuda (Mitsuda, 1995). The song has previously been deconstructed and analysed by Yu (2016). There are a number of sections in this piece that can be expressed and represented using the tools offered by IGME, that disseminate musical structure in the piece. Overall the piece has roughly 10% unique parts, and the remaining 90% can be expressed through the processes discussed in sections 4 and 5. Table 1 shows the overall distribution of parts found by this analysis process. No transformation parts were discovered so these are excluded from table 1. Figure 18 shows a visualisation of the overall piece. Duplicate parts are given the same colour, making patterns in the structure easier to distinguish, light blue is used to show unique parts, which are mostly present at the start of the piece.

	Number of Parts	Unique Parts	Duplicate	Transposed	Arpeggiated	Note Mapped
Counts	579	60	496	14	6	3
Percentage	100.00%	10.36%	85.66%	2.42%	1.04%	0.52%

Table 1: Analysis results for Secret of the Forest.

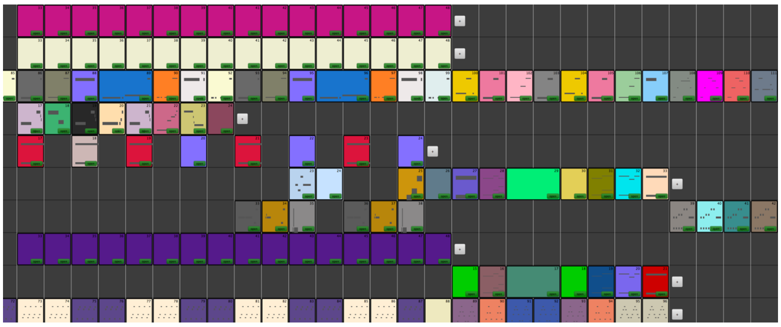


Figure 18: Visualisation of a section from the piece.

Although the analysis process performs optimally on this piece of music, there are a number of reasons why the piece cannot be analysed further. Tracks 1, 2, and 8 are percussion, and repeat a single idea throughout. Track 3 is a bass part and contains a lot of representable content, some of the content on this track is similar in structure however it is not easy to represent within IGME using current techniques, the same conclusion is true of tracks 4 also. Track 5 is almost exclusively 2 note chords, as all of the parts have the same structure this can all be represented through transforms. Tracks 6, 7, and 9, contain the bulk of the material, the unique ideas are different enough that they are mostly inexpressible by other means. 6 of the 7 parts on track 10 can all be expressed as arpeggiated parts. The final 2 tracks are mostly melodic ideas. From these 60 parts the rest of the song can be assembled.

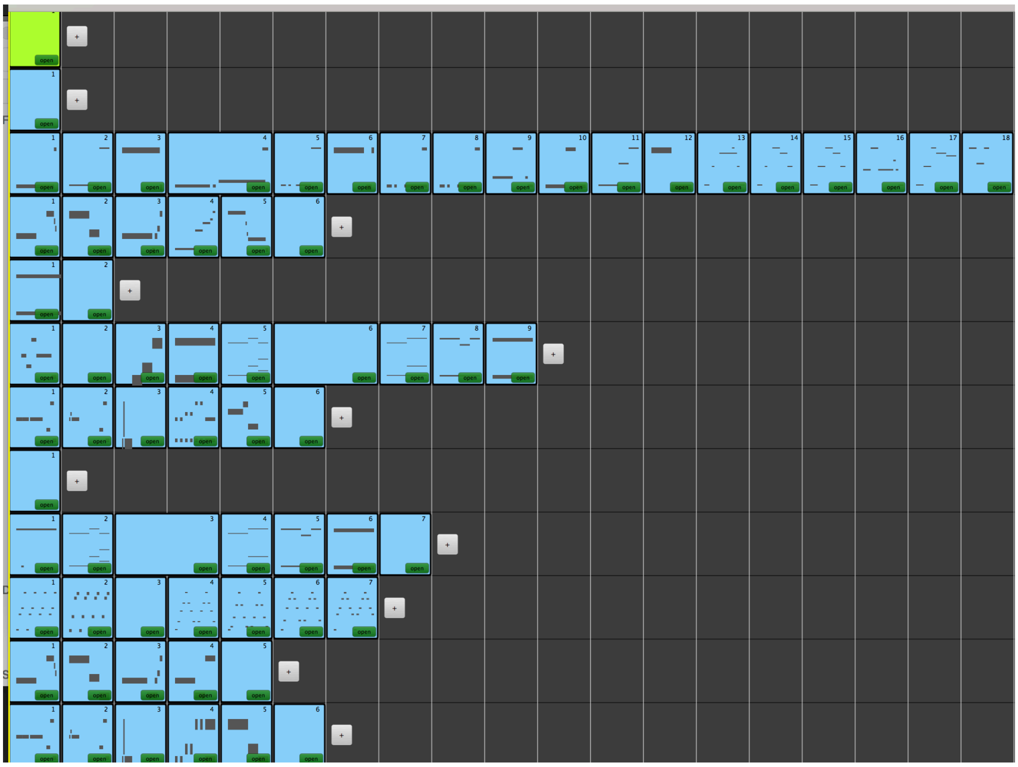


Figure 19: All of the unique ideas (building blocks) for secret of the forest.

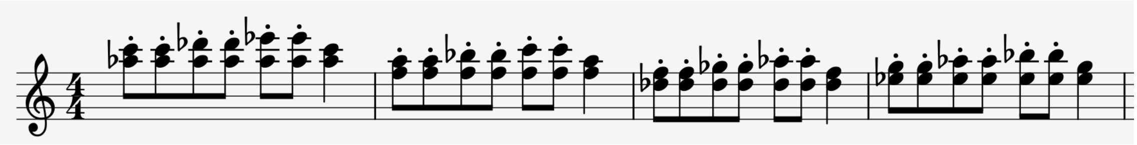


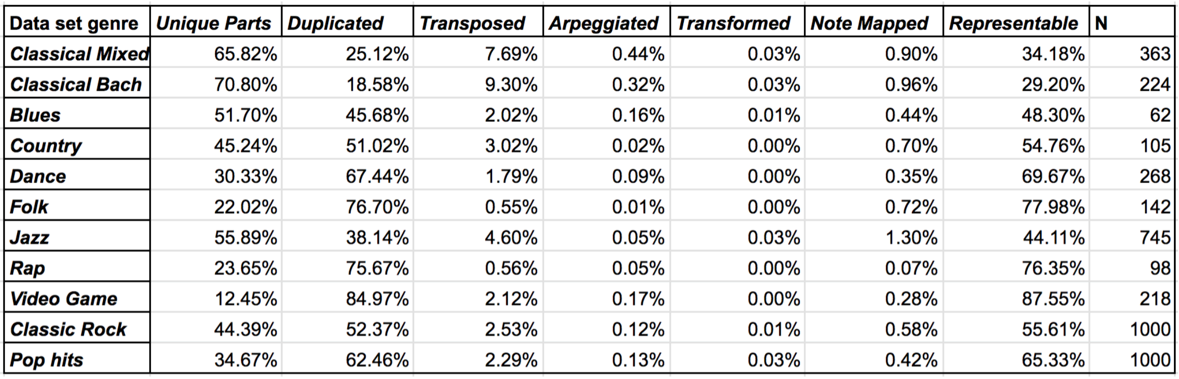
Figure 20: Musical score of repeating idea.

One of the more interesting structural ideas found when analysing the song was the repeating idea shown in figure 20, this is first used from bar 33. The same idea is repeated 4 times, but is chromatically transposed each time. The 4-bar section is then repeated 6 times throughout the piece. The repeated 4-bar sections are expressed as duplication (of the earlier section) rather than 1 part and 3 transpositions, as they did on the first occurrence.

7. DATA SETS

7.1. HUMAN COMPOSED MUSIC

To understand how the processes discussed in sections 4 and 5 can be applied to existing music it is important to analyse a large selection of it. To compute the datasets for this research, a large selection of MIDI files in different genres were gathered from various free online MIDI databases. These were then grouped by genre and analysed in bulk using the pipeline discussed in Section 5. Section 6 looked at analysing a single piece whereas this section applies the same process but for multiple pieces of work. Table 2 shows the results.



Data set genre	*Unique Parts*	*Duplicated*	*Transposed*	*Arpeggiated*	*Transformed*	*Note Mapped*	*Representable*	N
Classical Mixed	65.82%	25.12%	7.69%	0.44%	0.03%	0.90%	34.18%	363
Classical Bach	70.80%	18.58%	9.30%	0.32%	0.03%	0.96%	29.20%	224
Blues	51.70%	45.68%	2.02%	0.16%	0.01%	0.44%	48.30%	62
Country	45.24%	51.02%	3.02%	0.02%	0.00%	0.70%	54.76%	105
Dance	30.33%	67.44%	1.79%	0.09%	0.00%	0.35%	69.67%	268
Folk	22.02%	76.70%	0.55%	0.01%	0.00%	0.72%	77.98%	142
Jazz	55.89%	38.14%	4.60%	0.05%	0.03%	1.30%	44.11%	745
Rap	23.65%	75.67%	0.56%	0.05%	0.00%	0.07%	76.35%	98
Video Game	12.45%	84.97%	2.12%	0.17%	0.00%	0.28%	87.55%	218
Classic Rock	44.39%	52.37%	2.53%	0.12%	0.01%	0.58%	55.61%	1000
Pop hits	34.67%	62.46%	2.29%	0.13%	0.03%	0.42%	65.33%	1000

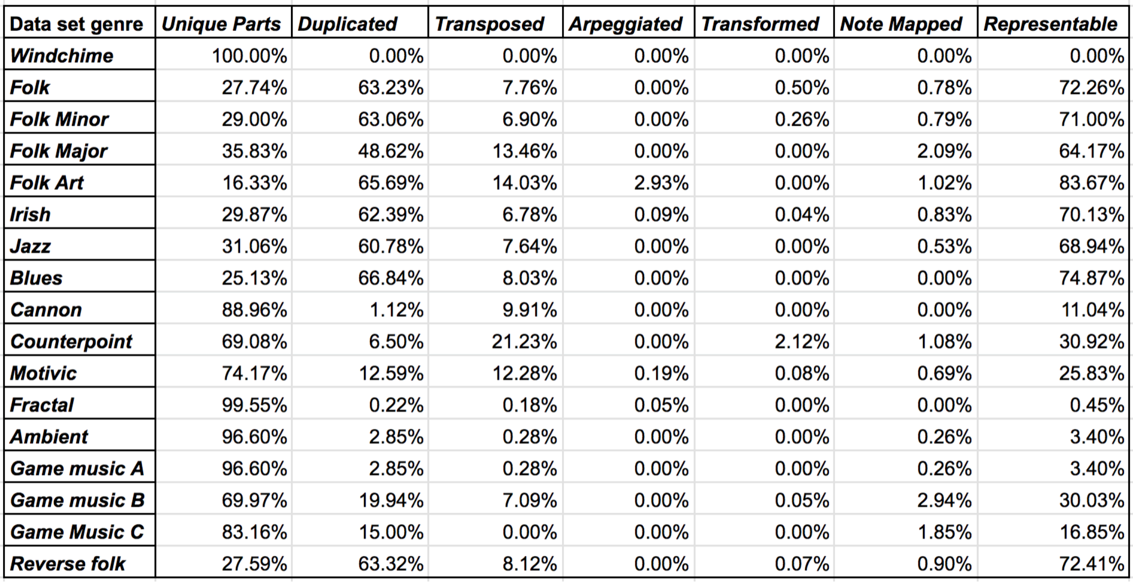
Table 2: Analysis results for various genres of music, whereby N is the number of files analysed.

The representable value is expressed as the percentage of parts that can be computed from another part (i.e. not unique). Initial observations of the data revealed that the Classical dataset scored the lowest for representability, unlike other styles of music is often instrumental, meaning that the music is perhaps more complex to accommodate for the lack of vocals. Pop, Rap, and dance music have high representable scores, perhaps as these genres of music make use of loops. Jazz has slightly more representability than classical but less than pop and rock. Video game music scores the highest overall. The results could be interpreted, that more popular forms of music (rock and pop) tend to express music in a simpler structure that conforms to the bar level hierarchy, whereas jazz and classic tend to follow more nuanced levels of structure that is not sufficiently captured by this process. Despite their low overall scores, Classical and Bach contain more transposed and arpeggiated parts then any of the other datasets.

The transformation (retrograde, inversion, rotation) technique is clearly in its current configuration either; unable to represent the music (incorrect model), or is just simply not used that often as a technique. Given the relatively high duplication score for almost all datasets, it is worth considering if current musical composition software makes this duplication either; easier to do, or make its representation obvious. Additionally, we suggest that these duplicated ideas are intentional and crucial for developing structured (non-stochastic) music.

7.2. GENERATIVE MUSIC

In addition, the output of a series of generative music programs was captured and used to create a large dataset of generative music. Based on work by (Francis, 2018) a selection of the example programs provided were compiled and run 1000 times each to produce 1000 pieces of music for each technique (see table 3). Even though the focus of this study was to test whether or not existing music could be represented by the techniques discussed in this paper, it is worth considering if and how generative music (composed by other programs) fits with this model.



Data set genre	*Unique Parts*	*Duplicated*	*Transposed*	*Arpeggiated*	*Transformed*	*Note Mapped*	*Representable*
Windchime	100.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Folk	27.74%	63.23%	7.76%	0.00%	0.50%	0.78%	72.26%
Folk Minor	29.00%	63.06%	6.90%	0.00%	0.26%	0.79%	71.00%
Folk Major	35.83%	48.62%	13.46%	0.00%	0.00%	2.09%	64.17%
Folk Art	16.33%	65.69%	14.03%	2.93%	0.00%	1.02%	83.67%
Irish	29.87%	62.39%	6.78%	0.09%	0.04%	0.83%	70.13%
Jazz	31.06%	60.78%	7.64%	0.00%	0.00%	0.53%	68.94%
Blues	25.13%	66.84%	8.03%	0.00%	0.00%	0.00%	74.87%
Cannon	88.96%	1.12%	9.91%	0.00%	0.00%	0.00%	11.04%
Counterpoint	69.08%	6.50%	21.23%	0.00%	2.12%	1.08%	30.92%
Motivic	74.17%	12.59%	12.28%	0.19%	0.08%	0.69%	25.83%
Fractal	99.55%	0.22%	0.18%	0.05%	0.00%	0.00%	0.45%
Ambient	96.60%	2.85%	0.28%	0.00%	0.00%	0.26%	3.40%
Game music A	96.60%	2.85%	0.28%	0.00%	0.00%	0.26%	3.40%
Game music B	69.97%	19.94%	7.09%	0.00%	0.05%	2.94%	30.03%
Game Music C	83.16%	15.00%	0.00%	0.00%	0.00%	1.85%	16.85%
Reverse folk	27.59%	63.32%	8.12%	0.00%	0.07%	0.90%	72.41%

Table 3: Output for different generative program types analysed.

The representability scores for the generative dataset vary widely. Firstly, the mostly stochastic techniques (fractal and windchime) have low representability scores, meaning these music types sound predictably chaotic in nature. On the other end of the scale Jazz and Blues have higher representability scores than their real-world counterparts. The counterpoint data set has a number of differences with the Bach data set, notably the generative set has less duplication, but much more transposition.

The findings in the section perhaps confirm why generative music is seen as either structureless (too stochastic) or repetitive. Therefore, a balance needs to be struck between repeating and developing existing ideas, and creating new ones. From the findings above, it would seem that this remains a challenge. It could also be suggested that generative music techniques are mostly suited to generating lower levels of musical hierarchy and that the human composers should focus on developing and arranging these lower levels to form higher level structure.

8. CONCLUSION

8.1. LIMITATIONS

The analysis processes used in the research considers musical structure to be grouped into bars, while this works for certain styles of music, much of the musical structure operates at smaller micro (rather than macro) levels. This is true of Classical music in which sequences may by asynchronous with bar lines. However as discussed previously this coarse resolution approach is designed to reduce the complexity of the research objectives. However, this automated approach paves the way for more formal methods of analysis such as Schenkerian analysis. Nash's (2014) *Manhattan* software (which has many parallels with this work) provides the ability to encode music at the micro and macro level thus providing a more complete representation of musical structure, although such representation must be encoded manually.

The system itself is still in a beta development stage and some limitations do present themselves. A major limitation is the inability to work with time signatures other than 4/4, and of course pieces that modulate to and from a different time signature. Such pieces are omitted from the data sets discussed in section 8. Many pieces of music cannot be represented sufficiently using IGME and there are two principle reasons for this. Firstly, the pattern-recognition capabilities of IGME are themselves limited, and are demonstrated here as a proof of concept. Developing these techniques for future work will undoubtedly increase the representable score of music, further highlighting the importance of patterns in music. Secondly, and for perhaps good reason certain music cannot be simply compounded into primitive rules.

8.2. FUTURE WORK

This research has several novel uses. Firstly, it allows a user interacting with the software to analyse music in a visually stimulating way. We can also use generalised metrics about music to assess why for example generative music is often seen as structureless, by analysing and comparing it with a style it is trying to replicate. This also might be used as a tool for learning a piece of music. Whereby a student can extract the individual pieces of music and practice these over, later slotting them the logical timeline to realise the full piece of music.

Ultimately the focus of this research was to assess whether existing pieces of music can be represented by a series of unique musical bars, and subsequent representations. As this paper has demonstrated this is indeed the case, with stronger emphasis for dance, folk and rap genres of music. Therefore, it is entirely possible to compose new pieces of music that intentionally use these types of processes. The number of techniques explored in this paper fail to capture all aspects of musical structure, however future work may look to address some of the shortcomings of this research.

9. REFERENCES

Collins, N. (2018) ‘there is no reason why it should ever stop’: large-scale algorithmic composition, *Journal of creative music systems*, Vol. 3, No. 1.

Cope, D. (1991) Computers and Musical Style, *Oxford University Press*, Oxford.

Cope, D. (1996) Computers and Musical Style, *AR editions Madison WI*, Wisconsin.

Cynthia, R. (2019), Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead, *Nature Machine Intelligence,* Vol. 1, No. 5, pp. 206-215.

Francis, J, R. (2018) *Algorithmic Computer Music,* 6th Ed, unpublished.

Fux, J.J. and Wollenberg, S (1992) 'Gradus ad Parnassum'(1725): Concluding Chapters, *Music Analysis*. Vol. 11, No. 2/3, pp. 209-243.

Hamanaka, M., and Hirata, K., and Tojo, Satoshi. (2006) Implementing A Generative Theory of Tonal Music*, Journal of New Music Research,* Vol. 35, No. 4, pp. 249-277.

Herrenabsm, D. and Sorensen, K. (2012) Composing first species counterpoint with a variable neighbourhood search algorithm: *Journal of Mathematics and the Arts.* Vol. 6, No. 4, pp. 169-189.

Hunt, S., Mitchell, T., & Nash, C. (2018, May). A cognitive dimensions approach for the design of an interactive generative score editor. *Paper presented at Fourth International Conference on Technologies for Music Notation and Representation,* Montréal, Canada.

Hunt, S., Mitchell, T., & Nash, C. (2017, September). Thoughts on interactive generative music composition. *Paper presented at 2nd Conference on Computer Simulation of Musical Creativity*, The Open University, Milton Keynes, UK.

Huang, A., and Wu, R (2016) Deep learning for music, Cornell University (website), available online from https://arxiv.org/abs/1606.04930 [accessed August 2019]

Langkjaer-Bain, R . (2018) Five ways data is transforming music, *Significance,* Vol. 15, No. 1, pp. 20-23.

Lerdahl, F. and Jackendoff, R. (1985) A generative theory of tonal music, Massachusetts, MIT Press.

Marsden, A. (2010) Schenkerian analysis by computer: A proof of concept, *Journal of New Music Research,* Vol. 39, No. 3, pp. 269-289.

Moore, A,F. (2001) Categorical conventions in music discourse: Style and genre, *Music and Letters*, Vol. 82, No. 3, pp. 432-442.

Moylan, W. (1987). A systematic method for the aural analysis of sound in audio reproduction/reinforcement, communications and musical contexts, *Proceedings of the 83rd Convention of the Audio Engineering Society*, New York.

Nash, C. (2014, June) Manhattan: End-User Programming for Music. *Paper presented at New Interfaces for Musical Expression (NIME),* London, UK.

Rothfarb, L, A. (2010) *Basic Formal Units (Motive, Phrase, Period), University of California, Santa Barbara (website)* available online from http://rothfarb.faculty.music.ucsb.edu/courses/160A/formal-units.html [accessed August 2019].

Temperley, D. (2007) Music and probability, Massachusetts, MIT Press.

Yu, J.M. (2016). Deconstructing: “Secret of the Forest” from Chrono Trigger, *jasonyu* (website), available online from http://jasonyu.me/secret-of-the-forest/ [Accessed November 2019].

Zulić, H. (2019) How AI can Change/Improve/Influence Music Composition, Performance and Education: Three Case Studies, *INSAM Journal of Contemporary Music, Art and Technology, Vol. 1, No. 2, pp. 100-114.*

10. DISCOGRAPHY

Guns N’ Roses. (1987), [CD], Appetite for destruction, *Universal Music.*

Mitsuda, Yasunori. (1995), [video game music], *secret of the forest*, *Square.*

3. Audio Mostly 2020: Exploring Polyrhythms, Polymeters, and Polytempi With the Universal Grid Sequencer Framework

Exploring Polyrhythms, Polymeters, and Polytempi with the Universal Grid Sequencer framework

SAMUEL J. HUNT, Creative Technologies Laboratory

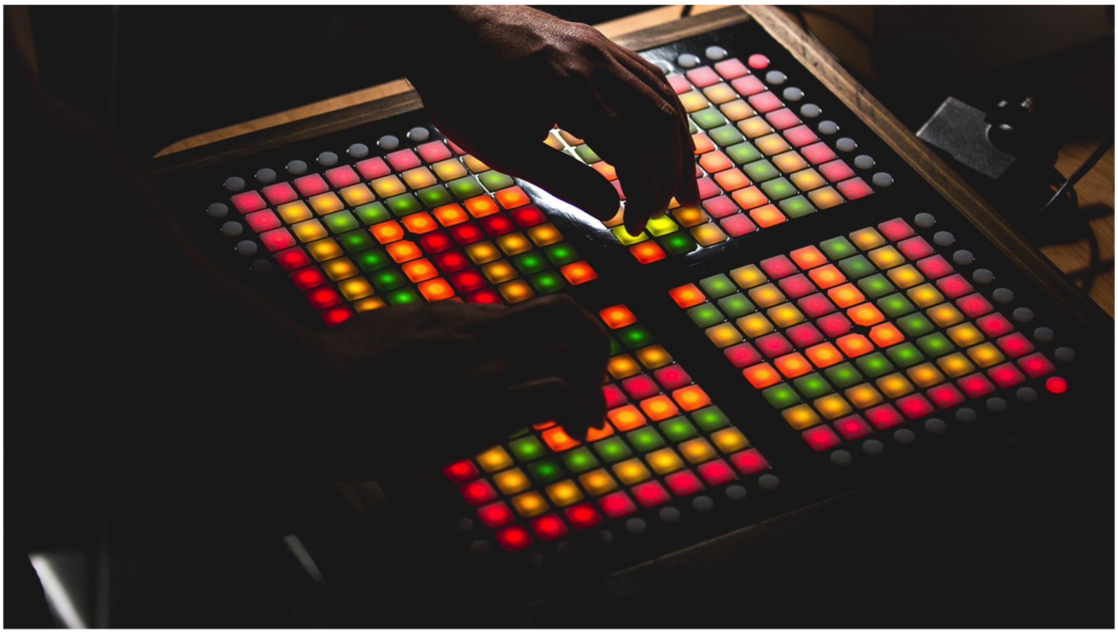


Fig. 1. Large format grid controller, made from 4 smaller grid controllers

Polyrhythms, Polymeters, and Polytempo are compositional techniques that describe pulses which are desynchronous between two or more sequences of music. Digital systems permit the sequencing of notes to a near-infinite degree of resolution, permitting an exponential number of complex rhythmic attributes in the music. Exploring such techniques within existing popular music sequencing software and notations can be challenging to generally work with and notate effectively. Step sequencers provide a simple and effective interface for exploring any arbitrary division of time into an even number of steps, with such interfaces easily expressible on grid based music controllers.

The paper therefore has two differing but related outputs. Firstly, to demonstrate a framework for working with multiple physical grid controllers forming a larger unified grid, and provide a consolidated set of tools for programming music instruments for it. Secondly, to demonstrate how such a system provides a low-entry threshold for exploring Polyrhytms, Polymeters and Polytempo relationships using desynchronised step sequencers.

CCS Concepts: • **Human-centered computing** → User interface programming; • **Applied computing** → **Sound and music computing**.

Author's address: Samuel J. Hunt, Creative Technologies Laboratory, UWE Bristol, Samuel.hunt@uwe.ac.uk.

2020. Manuscript submitted to ACM

Additional Key Words and Phrases: Polyrhythm, Polymeter, Polytempo, Grid Controllers

ACM Reference Format:

Samuel J. Hunt. 2020. Exploring Polyrhythms, Polymeters, and Polytempi with the Universal Grid Sequencer framework. 1, 1 (November 2020), 11 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Linear sequencer workflows generally enforce synchronization between tracks, whereby it is not possible to have more than one tempo or meter in concurrency [10]. For example one track at 120 bpm and another at 130, or one track in 4/4 and another in 9/8. Different types of notation allow or inhibit this in different ways. For example western score notation makes compound time permissible with different voices appearing on a single stave, or various compound notation across different staves. A piano roll notation allows virtually any discrete time division but can require tenacity and fine motor control to drag a note into a position. Step sequencer notation provides a simple way of expressing rhythms by dividing a period of time evenly into a fixed number of intervals (steps).

A step sequencer generally divides a sequence into evenly spaced steps, whereby a step corresponds to an individual note that is either on or off at that position [11]. Steps run left-to-right and voices are stacked vertically. Step sequencers, unlike traditional notation, make it easy to divide a bar of music into an arbitrary number of steps, for example dividing a bar of 4/4 into 11 identical pulses. Step sequencers have some novel properties, for example each row can be mapped; linearly, to a given scale, or to an arbitrary note value (drum sequencer). Step sequencers exist in both physical hardware and in software. We propose a simple step sequencer interface using physical hardware for exploring time, whereby we can have multiple step sequencers running together to both illustrate and audibly compare the effects of Polyrhythms, Polymeters, and Polytempi.

Grid controllers are physical interaction devices made up of a matrix of backlit buttons (see Figure 1). The interface provided by a grid controller lends itself well for representing and interacting with step sequencers. Many such controllers exist (explored in the next section) however the tools and frameworks for designing sequencers and more general purpose instruments for grid controllers are somewhat lacking. To enable us to explore time for the purpose of this project we created a general purpose framework for enabling the design of instruments for grid controllers (section 3). From this we demonstrate a step sequencer based approach using a large format grid controller for exploring complex rhythms (section 4). The large grid controller discussed in this project was built several years prior to this project and more details about its origin can be found here[1] .

2 BACKGROUND

2.1 Polyrhythms, Polymeter, and Polytempo

A polyrhythm is the simultaneous use of two or more rhythms that are not readily perceived as deriving from one another [1]. Given 2 bars of the same length and tempo (stacked vertically), a polyrhythm would be dividing the upper bar into 4 beats and the lower one into 3 beats. These beats would be in sync on the first beat and drift or phase for remainder of the bar (figure 2). A polymeter is where two sequences are played using different meters, but with the same tempo. For example a pattern that repeats a sequence over 5/4 played against one repeating over 4/4 (figure 3). After 20 pulses the original patterns will repeat again (the lowest common denominator of two time signatures). Polyrhythm and polymeter differ in that the first repeats every measure, and the latter at the phrase level [5].

[1] http://launchpadmegamini.blogspot.com/

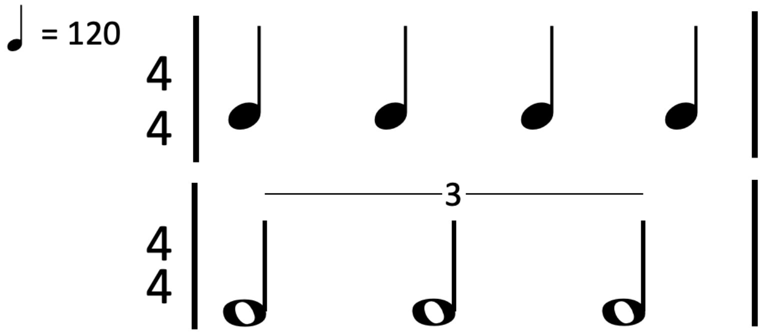


Fig. 2. 4/3 polyrhythm.

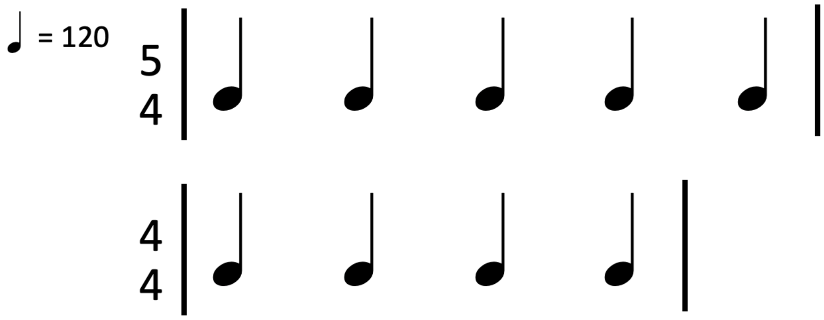


Fig. 3. 5/4 and 4/4 polymeter

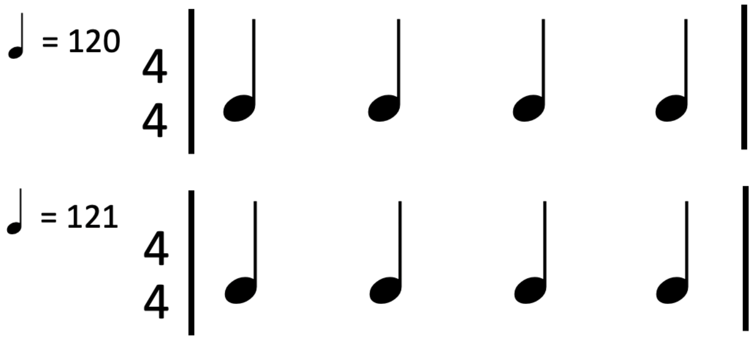


Fig. 4. two sequences using polytempo

Polytempo is whereby two sequences are played at different tempos, the point at which these sequences repeat can be significantly longer than sequences employing either polyrhythm or polymeter. For example a 1 bar repeating phrase sequence played at 120 bpm and at 121 bpm simultaneously will take 4 minutes to repeat (figure 4).

Renney and Gaster [10] state traditional notation software struggles to represent more complex manipulations of musical time. Additionally, expressing two time signatures in parallel (polymeter) or working with concurrent time (polyrhythm) is not generally well supported in notation software and thus requires undesirable 'work-arounds'. However some existing software systems do support these compositional methods.

Nash's [9] Manhattan tracker software enables a user to work with complex rhythms by subdividing rows in the tracker into a number of subdivisions. For example taking a row with a resolution of 1/32 note and dividing this into 7 smaller divisions, this effect can be repeated between tracks creating complex rhythmic structure. Malmberg's IRIS software sequencer [7], represents a step sequencer around the edge of a circle. The system supports having multiple sequencers within the circle so that polyrhytmic music can be explored. Renney and Gaster's [10] work presents a novel physical interface for exploring polyrhythms and polymeters on a 3D printed circular interface. Dorin [3] has explored polyrhythms in a more abstract form using cellular automata. More general purpose programming languages such as Supercollider [12] would allow a user to program any number of complex rhythms but requires competency of programming for an end-user. More formal methods for composing with complex rhythms is given by Dobrian [2].

2.2 Grid Controllers

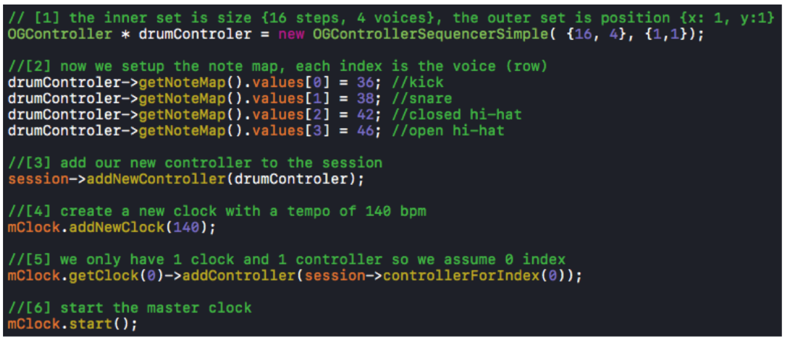
Grid controllers are generally defined as a matrix of buttons that output button state information when pushed. Many of these controllers have LED lighting behind each button so that visual feedback can be displayed. The first commercial example of a grid controller is commonly considered the Monome [4]. More modern controllers such as the Launchpad Pro[2] and Ableton Push [3] use velocity and pressure sensitive pads with full RGB feedback. Unlike traditional keyboards, grid controllers are more general purpose with specific software providing the instrumental interface and mapping for musical applications. Other than the Monome 256 (a 16 x 16 grid controller), Yamaha Tenori-on[4], and Polyend SEQ[5], larger grid controllers are rare and notably expensive.

Many purpose built musical applications exist for grid controllers, utlising Max for live patches, custom firmware, or programmed in a general purpose language. For example, Monome has a community of developers creating applications written using Max [8]. Novation's launchpad pro has the ability to write programs and embedded them directly on the device [6]. Ableton's push controllers provide specific interfaces when interfaced with Ableton Live. No general purpose framework exists for building instruments on a grid controller, but rather each device has its own set of tools and requirements. For example, a purpose built application for a Launchpad written in Max would require a significant rewrite to enable interaction with a push controller. Given the amount of common functionality between grid controllers, a framework that enables instruments to be designed for a *'general purpose grid controller'* allows developers and musicians to more easily develop and share instruments.

3 UNIVERSAL GRID SEQUENCER

The Universal Grid Sequencer (UGS) is an open source C++ framework that provides a series of musical applications for grid based controllers developed for this project. There are a number of components later discussed that make the system universal. But essentially it is a software package that allows multiple grid based controllers to be connected together as if they were one large grid controller and then position various sequencers and musical instruments on it. An instrument written specifically for one grid controller can easily be made to work with another. We briefly discuss

[2] https://novationmusic.com/en/launch/launchpad-pro
[3] https://www.ableton.com/en/push/
[4] https://www.yamaha.com/en/about/design/synapses/id_005/
[5] https://polyend.com/seq-midi-sequencer/



```
// [1] the inner set is size {16 steps, 4 voices}, the outer set is position {x: 1, y:1}
OGController * drumControler = new OGControllerSequencerSimple( {16, 4}, {1,1});

//[2] now we setup the note map, each index is the voice (row)
drumControler->getNoteMap().values[0] = 36; //kick
drumControler->getNoteMap().values[1] = 38; //snare
drumControler->getNoteMap().values[2] = 42; //closed hi-hat
drumControler->getNoteMap().values[3] = 46; //open hi-hat

//[3] add our new controller to the session
session->addNewController(drumControler);

//[4] create a new clock with a tempo of 140 bpm
mClock.addNewClock(140);

//[5] we only have 1 clock and 1 controller so we assume 0 index
mClock.getClock(0)->addController(session->controllerForIndex(0));

//[6] start the master clock
mClock.start();
```

Fig. 5. Code example for setting up a step sequencer. See AMSequencerSimpleDrumSequencer in the source code.

the core concepts as a full detailed description of the system is beyond the scope of the project, full source code is available from[6].

Firstly all MIDI devices are connected through a device object. The device object is responsible for providing the mapping between a device's physical control mapping and the virtual logical mapping employed by the system. For example, the Launchpad controller used by this project sends note on messages for each pad. Each row is represent by increments of 16 and each column by 1. These are translated by the device component into an XY coordinate position. The physical device itself can be oriented in one of four predefined positions (rotated in 90 degree increments) with the software automatically computing the transformations. The device object must also provide the reverse mapping, i.e. given a XY coordinate which pad does this relate back to. We propose that any grid based MIDI controller can be used with the system providing a device and its mapping is defined. For practical reasons we have not tested every variant of grid controllers. The basic requirements for any device is that it supports a state based button press (sending on and off messages), has LED feedback under each button and communicates via MIDI or OSC.

LED feedback is a core part of many grid based controllers, and decoupled from a pad's input. The device object also deals with mapping an arbitrary RGB colour value into a device's physical ability to represent this. Many controllers (Push, Launchpad Pro) support full RGB colour, however some controllers (the Launchpad used by this project) support only red and green with a 2 bit colour depth for each channel. In this instance the red and green components are down-sampled to be compatible.

The device manager is responsible for connecting and positioning several devices together to build an overall grid. The position, size and orientation of these devices can be arbitrary. For example, in this project we have four launchpad devices positioned in a 2 by 2 grid, with each device rotated through 90 degrees (see figure 1). At this point we have an overall grid size of 18 by 18. The device manager is responsible for translating messages from and back to individual devices. For example a message from the bottom right launchpad at its local position 0,0 becomes 9,9 in the overall grid. Likewise, sending LED feedback to position 9,9 will return that feedback to the bottom right launchpad. At this point input messages are transferred from the MIDI thread into the master clock thread using a circular buffer.

Furthermore, all the IO between devices is setup and focus can now be placed on designing controllers. A controller is simply a musical instrument or sequencer of arbitrary size and position within the grid. A session object manages an arbitrary number of controllers, and transfers messages from the device manager onto the relevant controller. For

[6] https://github.com/Sjhunt93/Universal-Grid-Sequencer

example we might have 3 controllers, a drum pad on the top right and note scaler on the top left and a 16 step sequencer occupying the bottom half of our grid. Controllers generally fit into one of 3 categories, momentary controllers, sequencer controllers, and setting controllers. The first of these is the most simple and generally maps input into output MIDI messages. For example we could have a 4 by 4 drum pad that maps each XY position to a MIDI number corresponding to specific drum sounds (e.g. kick = MIDI note 36, snare = MIDI note 38). Sequencer controllers are similar, but instead output notes at some point in the future and resemble traditional step sequencers. These respond to messages sent from clock objects that provide timing information (discussed shortly). Setting controllers send messages that control the state of other controllers, for example switching scales, transposing, selecting sequences, and others.

The master clock is responsible for transferring incoming messages into the individual controllers (through the session manager) and sending LED feedback to the physical devices, but also keeps track of and manages a number of clock objects. Clock objects send regularly pulsed messages to a given number of controllers. There is a 1-to-many mapping between clocks and controllers, however for simplicity 1 clock might control every controller ensuring synchronicity between them. For more complex examples we might have 4, 16-step sequencers each with 4 voices on our 16 by 16 grid, with each sequencer receiving a different clock. This allows each sequencer to go in and out of phase and sync with each other.

All LED feedback is sent to a LFXBuffer (LED effects buffer). Each controller has its own buffer. For example a controller with a size of 5 by 5 has a buffer of the same size. The device manager holds a master buffer that is the same size as the grid. Around 60 times a second the buffers from each controller are transferred to the master buffer and positioned accordingly. The master buffer is a representation of the physical device so therefore output messages are only sent if the state has changed which avoids overloading the communication output stream.

The system is entirely written in C++ and currently configured through changing and modifying source code which is relatively simple to do. Further iterations of the project intend to provide a drag and drop user interface and/or a domain specific language for setting up controllers. Figure 5 shows some example code for creating a step sequencer, and Figure 6 shows the output of this on the physical grid. To create a simple step sequencer we would allocate an instance of it (step 1), and also supply the two required setup parameters: a size (which for this controller determines its number of steps and voices) and a position on the grid. We then set the note numbers for each row (step 2), and add it to the session (step 3). As the controller is a sequencer type (see previous section) we need to create a clock and register this controller to it (steps 4 and 5). If we wanted to add a second sequencer we would repeat the above process. The sequences could either share the same clock or alternatively register a new one. We finally start the master clock (step 6). All MIDI output messages from the program are sent out via a virtual MIDI device and synthesised in an external program, these messages are separate to the LED Feedback (also sent as MIDI) messages sent back to the physical devices.

4 EXAMPLES

Whereas the previous section presented the system as a generalised tool for working with grid based controllers, this section explores how a large grid based controller easily supports experimenting with time using examples in polyrhythms, polymeters, and polytempi. A large grid enables us to have multiple sequences not only running at once but remain visible.

Although creating an arbitrary number of virtual step-sequences in software (for example in Max) is easily possible, physical hardware provides tactile and sensory feedback. In the examples below, 16 steps corresponds to a bar of music in 4/4 time. All example code is available from the open source repository, the accompanying video demonstrates each of

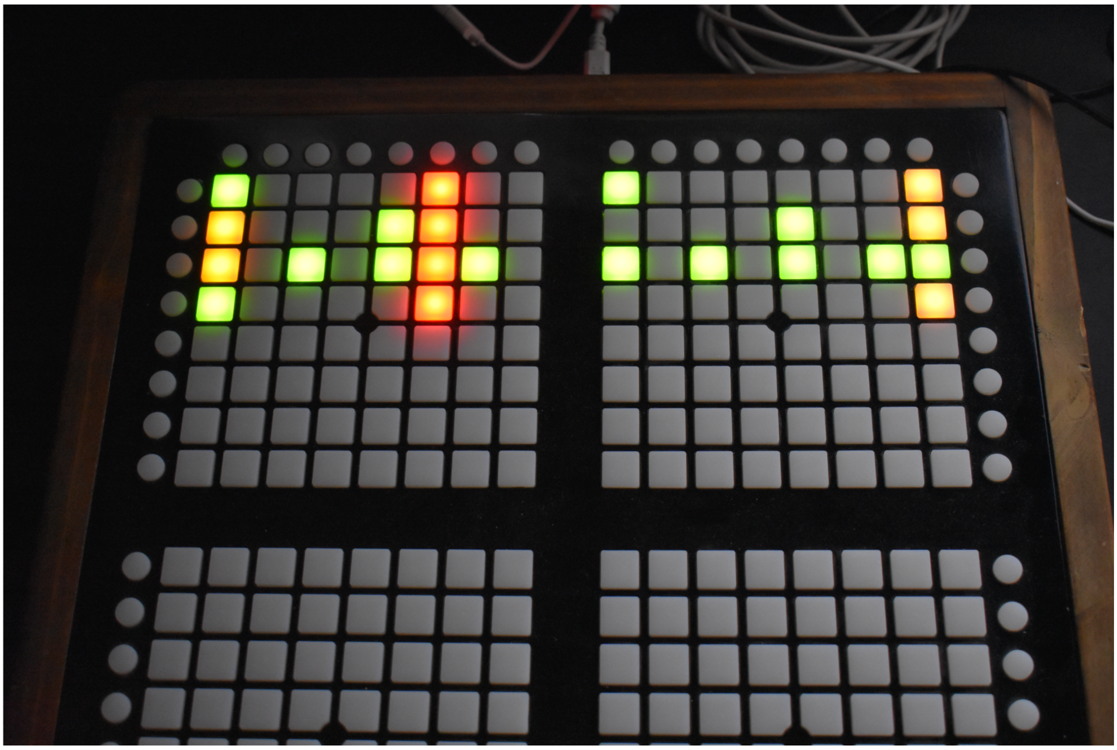


Fig. 6. Basic step sequencer, realisation of the code in figure 6

the concepts, and provides more details than is possible in paper[7]. Additionally, the range of possibilities are far greater than that demonstrated here as we focus on providing relatively simple instruments that promote demonstrability over complexity.

4.1 Polyrhythmic examples

To showcase polyrhythmic controllers (see figure 7) we need to create two (or more) step sequencers that are always in sync on the first step. In order to set up a 4 over 3 polyrhythm we would create two step sequencers one that has 16 steps and one that has 12 (any number of steps would work as long as the ratio between them is 4/3). Each step-sequencer requires a different clock, in this example the first has a tempo of 120 and the second 90 (25% slower). In any polyrhythmic container the ratio between steps (16/12) and clock speed (120/90) are proportional.

The software constructs these containers and computes relative clock speed automatically and a user simply needs to supply the number of steps for each of the two sequencers. In the example, the top of the grid has the 16 step sequencer and the bottom of the grid the 12 step sequencer, with each controller having 8 voices. LED feedback is used to show the start and end points of each sequencer. With little effort we could easily create a sequencer that is a 16/15/13/11 polyrhythm, which is considerable complex, although its musical merit remains something for the user to evaluate.

[7] https://www.youtube.com/playlist?list=PLES5ig6CvJKTYDKepwLWc37nniWoP7gwX

One downside of this representation is that (using the example shown at the start of this section), the 3/4 rhythm is shown to be shorter that the 4/4 rhythm, when in reality these are the same length. This is of course due to the limitations of needing a button for every time step. A continuous interface such as a touchscreen would allow this easily, but is not the focus of the project.

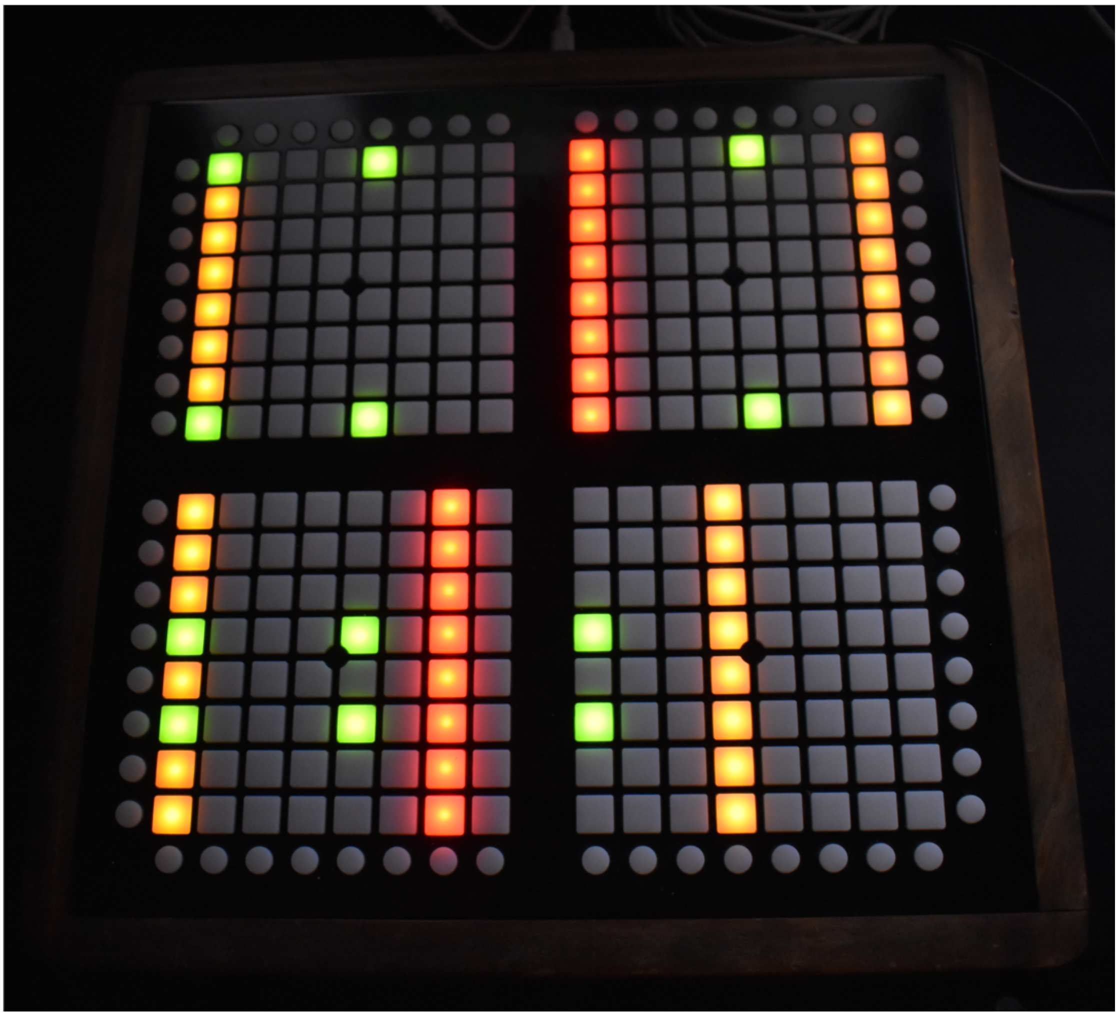


Fig. 7. 16/12 polyrhythmic sequencer, class: AMSequencer2WayPolyrhythmicContainer

4.2 Polymetric examples

Polymeterical examples are easier to configure as the sequencers share the same clock, but have a different numbers of steps between sequencers. In the example (figure 8) we have a 16 step sequencer played against a 10 step sequencer, giving us a 4/4 over 5/8 polymeter. As these are shown together visually, the user can observe the sequences going in and out of sync.

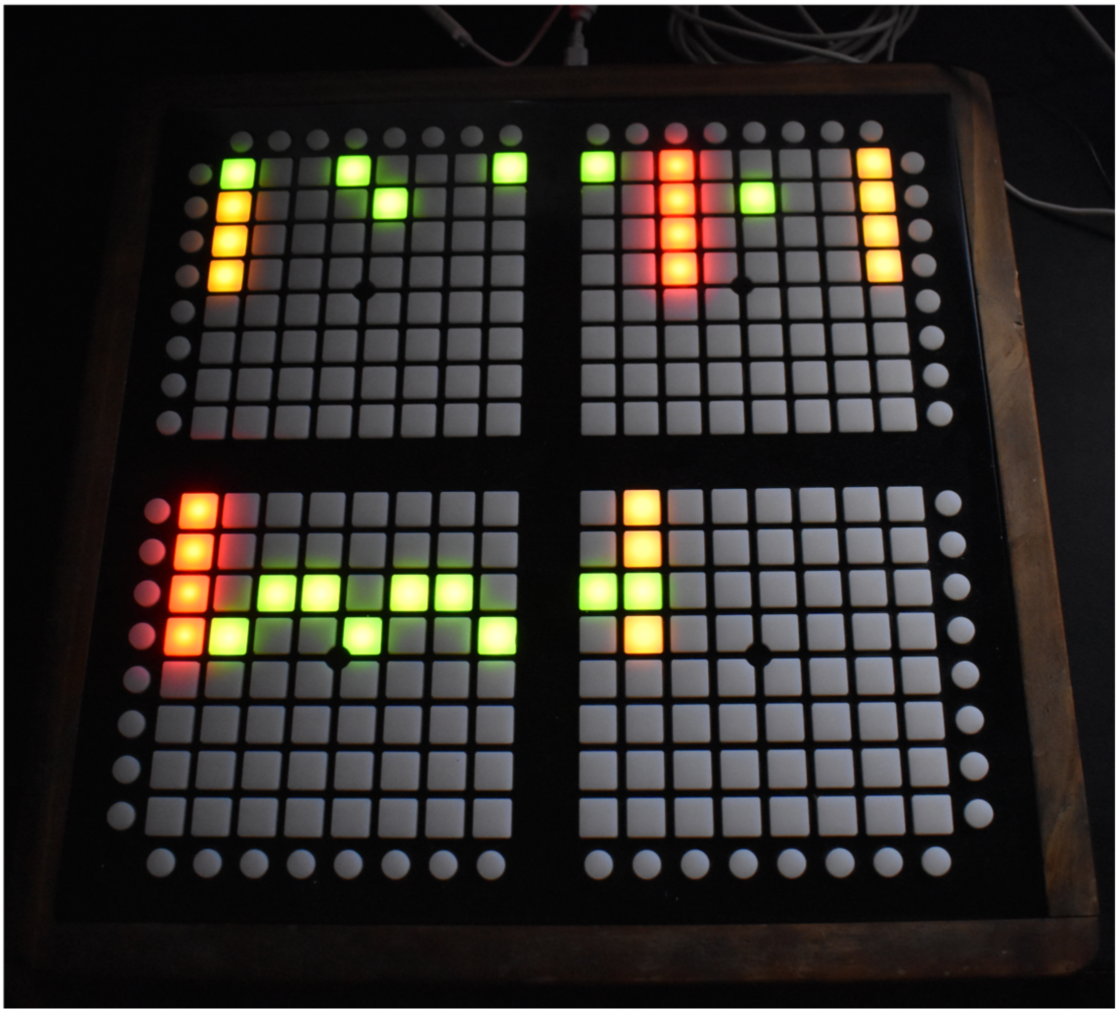


Fig. 8. 4/4 against 5/8 polymetric sequencer, class: AMSequencerPolymetricContainer

4.3 Polytempi examples

For the polytempo example we created 8 separate sequencers with a single voice (with each row outputting a separate note) and all 15 steps in length (see figure 9). Each sequencer was given its own clock with a tempo 1 bpm faster then the previous one, starting at 120. The lower rows appear to speed up as the pattern progresses, thus illustrating the concept visually. In this example it would take 4 minutes for the piece to repeat.

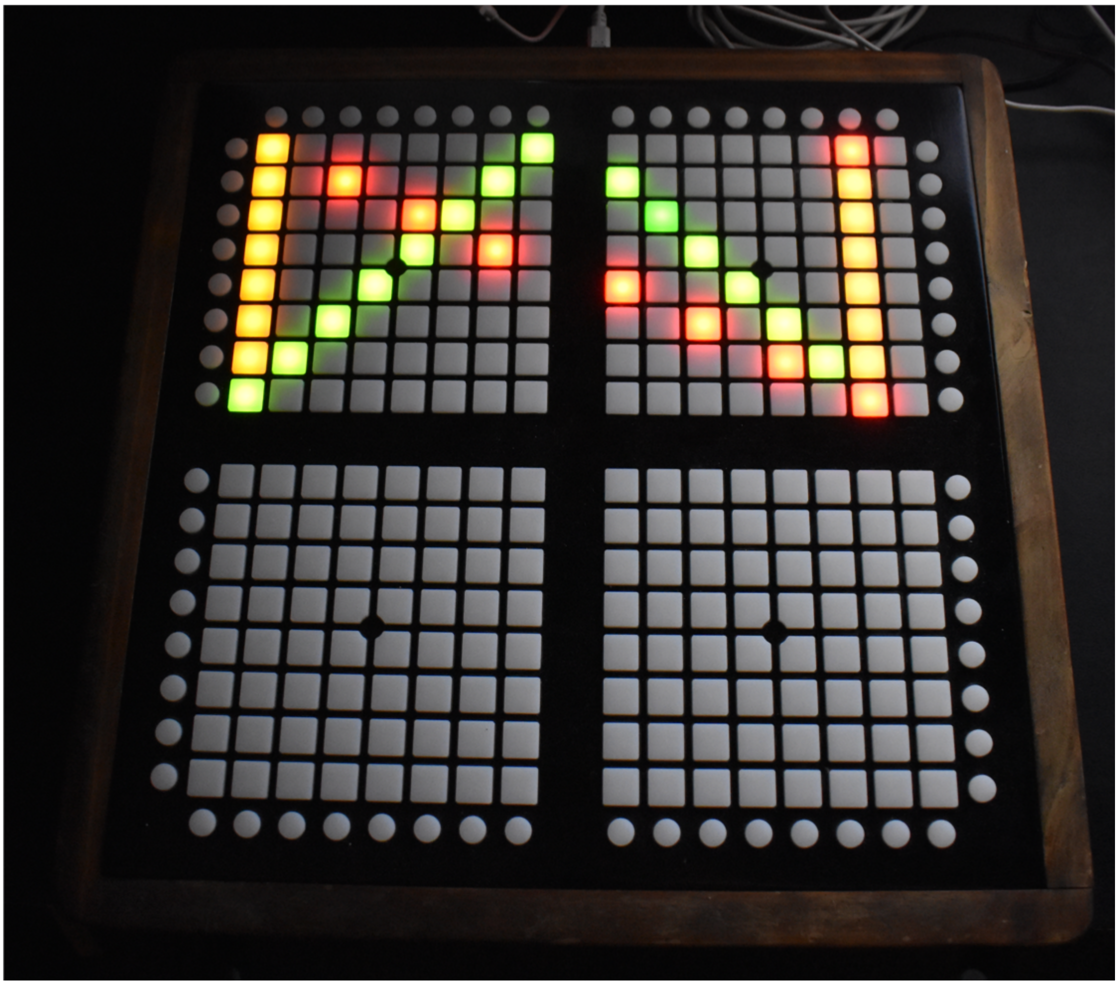


Fig. 9. 8 step sequencers running in a polytempi configuration, class: AMSequencerPolytempoContainer

5 CONCLUSION

This research sets out the initial objectives of the UGS framework and discusses its architecture. We encourage the development of the system to support more grid controllers and the creation of novel instruments. The project is provided as open source software. The framework provided us with a platform from which to develop and easily create step sequencers for exploring polyrhythms, polymeters and polytempos. With more complex examples of rhythms seemingly stochastic music was easily created, however the focus of this research was not on the musical quality of the output. Mathematically the music can be modelled and described, although it is unclear audibly whether a listener can perceive these structures, and would warrant further study and research.

Future work should therefore be primarily focused on evaluating the research with empirical user studies. Many activities are permissible, however emphasis should be placed on gauging a users understanding of complex time and rhythmic relationships, and the ease at which such interfaces can be assembled. Other studies could include more open ended exercises focusing on what a user chooses to do with these tools and to what extent they felt artistically liberated by using them.

REFERENCES

[1] Willi Apel. 2003. *The Harvard dictionary of music* (4th ed.). Harvard University Press, Cambridge, Massachusetts.

[2] Christopher Dobrian. 2012. Techniques for Polytemporal Composition. In *Proceedings of Korean Electro-Acoustic Music Society's 2012 Annual Conference (KEAMSAC2012)*. Korean Electro-Acoustic Music Society, Seoul, Korea, 1–8.

[3] Alan Dorin. 2002. LiquiPrism: Generating polyrhythms with cellular automata. In *Proceedings of the 2002 International Conference on Auditory Display*. Georgia Institute of Technology, Advanced Telecommunications Research Institute, Kyoto, Japan, 447 – 451.

[4] Jared Dunne. 2007. Monome 40h Multi-Purpose Hardware Controller.

[5] Martim Galvao. 2014. *Metric Interplay: A Case Study In Polymeter, Polyrhythm, And Polytempo*. Ph.D. Dissertation. UC Irvine.

[6] Dave Hodder. 2018. *Launchpad Pro*. Technical Report. Focusrite. https://github.com/dvhdr/launchpad-pro

[7] Viljo Malmberg et al. 2010. *Iris: A circular polyrhythmic music sequencer*. Master's thesis. Aalto University.

[8] Monome. 2020. *Grid*. Technical Report. Monome. https://monome.org/docs/grid/

[9] Chris Nash. 2014. Manhattan: End-user programming for music. In *Proceedings of the International Conference on New Interfaces for Musical Expression*. NIME, Goldsmiths, University of London, 221–226.

[10] Nathan Renney and Benedict R Gaster. 2019. Digital Expression and Representation of Rhythm. In *Proceedings of the 14th International Audio Mostly Conference: A Journey in Sound*. Association for Computing Machinery (ACM), Nottingham United Kingdom, 9–16.

[11] Richard Vogl and Peter Knees. 2017. An intelligent drum machine for electronic dance music production and performance.. In *17th International Conference on New Interfaces for Musical Expression*. NIME, Copenhagen, Denmark, 251–256.

[12] Scott Wilson, David Cottle, and Nick Collins. 2011. *The SuperCollider Book*. The MIT Press, Cambridge, Massachusetts.

4. CSMC 2020: An Analysis of Repetition in Video Game Music

An Analysis of Repetition in Video Game Music

Samuel J. Hunt

Creative Technologies Laboratory
UWE Bristol
Samuel.hunt@uwe.ac.uk

Abstract. Video game music, unlike other forms and genres of music, is comparatively young in its development, and undoubtedly heavily influenced by the hardware it was originally played on. This study observes whether the amount of repetition in video game music has been affected by limitations in hardware and physical storage size. Musical structure often derives from repetition and is one of many crucial musical elements for defining structure. We hypothesise that early videos games will contain above average repetition due to hardware limitations. Early video game soundtracks were minimally encoded, utilising hardware based chip-tune synthesis, whereas modern games can support *Hollywood style* soundtracks. This research analyses 21,391 pieces of video game music across a range of various video game consoles. In conclusion our original hypothesis was mostly disproven, and in fact the repetition structure discovered across different platforms and generations remained more-or-less consistent. There were however improvements in song length and the number of instrument tracks within a song. Overall this paper presents an initial informal analysis of repetition in video game music.

Keywords: Video game music, music analysis, music structure

1 Introduction

Despite its relative immaturity, video game hardware and audio has rapidly progressed over the last 60 years. Early hardware had primitive, if any sound, with some systems only able to emit a single beep (Chang, Kim, & Kim, 2007). Modern systems are employing dynamic and evolving soundtracks, immersive audio, and complex surround sound mixing (Hutchings & McCormack, 2019). Music analysis is a vast subject and there are many feasible research objectives for analysing video game music. However, one characteristic that perhaps captures musical structure holistically is repetition. Rahn (1993) states most musical structure derives from repetition - without repetition there is no structure and the music itself will be little more than random. The main objective of this research is to observe how repetition within video game music has progressed over the various generations of hardware. Other musical elements such as key, harmony and rhythm could also have been analysed and compared, however, in line with other work in the field (see section 2) and to keep things within the scope of this paper we focus on a singular element (repetition).

2 Literature Review and Video Game Hardware

Musical structure happens at different hierarchal levels. Lerdahl and Jackendoff (1996) consider music to be in one of a four-level hierarchy; motifs, phrases, periods, and larger sections. Within these levels repetition occurs in different ways, for example a single bar of music may be repeated several times in succession, whereas an entire section may be repeated more than once, such as a chorus section. Classical musical forms, such as ternary form (ABA), emphasises a full repeat of the first A section, after the B section. Humans are excellent pattern recognisers and even those without formal training are competent enough at recognising repeating musical sequences and motifs (Chai, 2005).

Discovering repeating patterns within music automatically using computer programs is not a novel problem, and researchers have gone about solving this in different ways. This research focuses on analysing digitally sequenced music (i.e. MIDI). Some notable research has attempted this on recorded audio (Lu, Wang, & Zhang, 2004), and with the recent developments in machine learning, interest in this has increased (Jhamtani & Berg-Kirkpatrick, 2019). Hsu, Liu, and Chen (2001) present two methods for discovering repeating patterns in music, one using a correlative matrix and the other using a tree type data structure. Their study focuses on finding patterns of music at the smallest structural level. Other methods include using a dictionary-based compression algorithm (Shih, Narayanan, & Kuo, 2001). Both of these methods are a little complex for the purpose of this study. Another simpler method used by the authors (Hunt, Mitchell, & Nash, 2019) in previous work for analysing repetition, is to simply split a piece of music into smaller segments based on bar lines and then compare these against one another. Although some nuanced details are lost by this process it permits a simple automated approach that can cover a large dataset. We employ the same methods in this research and discussed further in section 3.

Collins et al. (2008) has thoroughly explored the relationship between technical systems and their musical use in video game music. An example of her work that had similar research objectives to the work in this paper, is exploring the link between hardware limitations and the use of loops in 8-bit video game hardware (Collins, 2007). She suggests that there is some correlation between the provision of music (use of repetition) and the storage capacity of the games cartridges, but in conclusion notes *"that rather than being the consequence of the limited memory available on the systems, loops were, at least in part, an aesthetic that grew as the games became more popular and more complex"*. Noting that creative composers have invented ways in which to overcome or even to aestheticize those limitations. In summary the research was limited by focusing solely on 8-bit generation hardware. With influence of Colins' work, we conduct similar research but for a wider range of hardware.

A full technical description of the vast history of video game audio hardware is outside of the scope of this research, however, we can broadly categorise each piece of hardware into one of 3 groups using work by (Chang et al., 2007). These are; chip tune and 8-bit (Group A), synthesis and sampling (Group B), and pre-recorded and streaming music (Group C).

Chip tune and 8-bit systems (A) synthesised music in real-time using primitive on-board hardware based sound generators, similar to analogue synthesisers. Synthesis and sampling systems (B) moved away from discrete waveform generators into more general purpose synthesisers. Wavetable and FM synthesis were heavily utilised given their small memory footprint. Pre-recorded and streaming based systems (C) differ in that music could be composed and recorded externally and then played back in real time. Fritsch (2013) notes that in early video games the role of the composer often fell on the programmers themselves given that forms of storage and playback were not well formalised. Music, for the first two categories, required extensive programming for in game playback. Whereas with pre-recorded and streaming based systems the role of the composer and the sound programmer were fully decoupled.

A game's storage medium may have affected the music and audio utilised within the game. Early hardware used cartridge based media with limited storage capacity, whereas disk based media provided ample storage. For example, a Nintendo 64 cartridge provided a maximum of 64 mb whereas its closest competitor, Sony's Playstation, had 660 mb with its CD based storage (Sakamura, 1999).

There are 8 defined generations of video game consoles (Kemerer, Dunn, & Janansefat, 2017), with music from generations 3-6 used in this research. The first two generations contained minimal, if any, music as the hardware was primitively simple. The most recent generations have seen little advancements in terms of audio hardware implementation (Chang et al., 2007), but have emphasised improvements in immersive audio (surround sound formats) and dynamic generative soundscapes (Hutchings & McCormack, 2019) which create challenges for obtaining and analysing such music. Furthermore, the dataset (discussed shortly) used in this study contained few examples of music for generations 7-8.

The video games consoles whose music was analysed for this research are listed in table 1 (see appendix). The Sega Megadrive and Nintendo 64 both supported disk based media through hardware add-ons (Sega CD and N64 DD) however games using those add-ons were removed from the datasets to eliminate additional variables. For this study we only focused on home video game consoles, neither handhelds or PCs were considered.

3 Methodology and Software

To analyse the music needed for the project, the interactive generative music environment (IGME) was used to compute the repetition analysis (Hunt, Mitchell, & Nash, 2018). The same authors (Hunt et al., 2019) have completed similar work using IGME but focused on algorithmic representation of music. The dataset for the project was obtained from the online VGMusic database[1] which contains a large database (over 30,000) of MIDI files categorised by console.

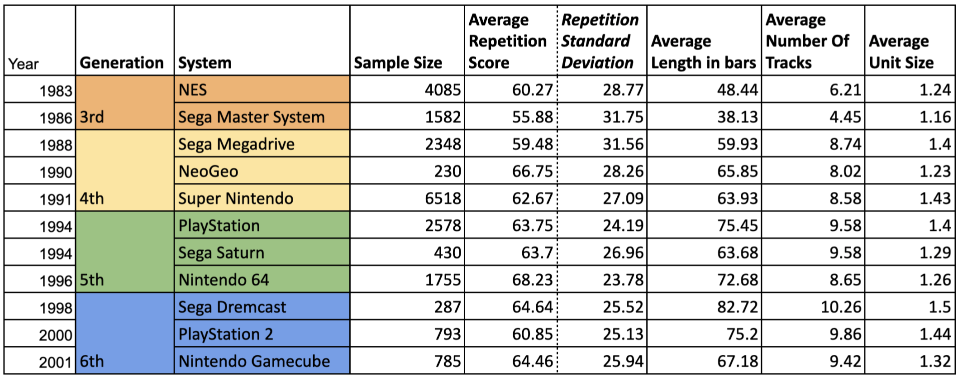
Individual songs are imported into IGME as MIDI files and then analysed. IGME only supports the analysis of files in a 4/4 time signature. The music is

[1] https://www.vgmusic.com/

segmented into smaller clips for analysis, for the purpose of this study (and as a limitation of the software) the smallest clip size is 1 bar. If notes are tied across a bar line then they are simply segmented at the next possible (non-note crossing) bar line, leaving a phrase larger than a single measure (we subsequently found this to only account for a small minority of pieces). Therefore, a clip for analysis is defined as 1 or more measures of music. A clip size average score of 1, implies that all the music in the piece could be split uniformly into 1 bar measures. Splitting into smaller units (i.e. less than a single measure) requires more complex analysis on both segmenting and subsequent analysis. Some notable work (Temperley, 2007) has explored options for this (albeit in a non-automated way), as well as the more general purpose repetition analysis methods discussed previously. Note length and onsets are quantised to the nearest 1/32 note. Other details are lost by the process, such as dynamic markings. Without making these modifications the complexity of doing this analysis would be implausible.

The repetition analysis attempts to find and group measures of music that have identical content. So that within each bar, every note has the same note length, onset time and note number; therefore, comparing every clip against every other clip on the time line. In the case of parallel duplication (two tracks doubling the same music), this increases the overall amount of repetition found as these two clips would be identical. Should two tracks be completely identical in their entirety, one is removed before starting the analysis processes. Although our definition of repetition excludes measures that are almost identical (for example a variation on a theme), it is difficult to analyse and categorise such measures, this could be an area for future research.

4 Results



Year	Generation	System	Sample Size	Average Repetition Score	*Repetition Standard Deviation*	Average Length in bars	Average Number Of Tracks	Average Unit Size
1983	3rd	NES	4085	60.27	28.77	48.44	6.21	1.24
1986		Sega Master System	1582	55.88	31.75	38.13	4.45	1.16
1988	4th	Sega Megadrive	2348	59.48	31.56	59.93	8.74	1.4
1990		NeoGeo	230	66.75	28.26	65.85	8.02	1.23
1991		Super Nintendo	6518	62.67	27.09	63.93	8.58	1.43
1994	5th	PlayStation	2578	63.75	24.19	75.45	9.58	1.4
1994		Sega Saturn	430	63.7	26.96	63.68	9.58	1.29
1996		Nintendo 64	1755	68.23	23.78	72.68	8.65	1.26
1998	6th	Sega Dremcast	287	64.64	25.52	82.72	10.26	1.5
2000		PlayStation 2	793	60.85	25.13	75.2	9.86	1.44
2001		Nintendo Gamecube	785	64.46	25.94	67.18	9.42	1.32

Fig. 1. Repetition analysis results, grouped by system.

Our original hypothesis stated older game music would contain more repetition than modern equivalents, given the hardware limitations discussed previously. The research analysed 11 groups of MIDI files (1 per platform), using the method described previously, with the results in figure 1. The repetition score is given as a mean for all music analysed per system as well as a standard deviation of the mean. The average song length in bars, the average number of MIDI tracks (after removing duplicate tracks) and the average clip size are also given.

The overall results, show that the repetition score varies little between systems. Taking an overall mean average, we find a repetition value of 62.79 and a standard deviation of 3.49. This would suggest that on average 62.79% of the musical clips in the music are a repeat of a previous clip.

One value that changes significantly over time is the average song length as well as the number of permissible tracks. It would appear that advancements in a game's storage size allowed for longer song lengths, and advancements in audio hardware increased polyphony. One interesting observation is that when comparing the Nintendo 64 and Playstation, they have similar values for repetition and song length. We hypothesised that Playstation music would have longer song lengths given how its storage capacity was 10x that of the Nintendo. Although the Playstation has marginally longer songs and higher track counts, it appears that these storage limitations had little effect on a songs structure.

5 Conclusion

In conclusion we found little variance in musical repetition between video game consoles and their subsequent hardware. Although older hardware itself did not specifically contribute to repetition, it limited the amount of songs that a individual game could contain. Such an observation would however require future study. Collins (2007) whose research was similar to that discussed here, observed that despite some quite rigid technological constraints provided only *'loose pressures'* on the development of audio for 8-bit video game hardware.

The style of music composed for a video game is hugely dependent on the type of game and is something overlooked in this study. Future work may focus on the differences between such genres although this would require significant organisation of the dataset. A limitation of using MIDI files is that two independent music lines or voices might be encoded on a single MIDI track, this may therefore have an adverse effect on the results. Is it however considerably difficult to separate voices once they are encoded without tediously analysing and manually separating the content. Also worth noting, is the smaller timbre characteristics of the audio when played on its respective physical hardware undoubtedly creates small nuances in the structure of the music. These are not sufficiently captured by MIDI encoding. Although repetition is an important musical characteristic it does not describe all the complexities of musical structure. Future work may also include additional analysis processes, such as: rhythm, use of key, instrumentation, polyphony, harmony and others.

References

Chai, W. (2005). *Automated analysis of musical structure* (Unpublished doctoral dissertation). Massachusetts Institute of Technology.

Chang, K., Kim, G., & Kim, T. (2007). Video game console audio: evolution and future trends. In *Computer graphics, imaging and visualisation (cgiv 2007)* (pp. 97–102). Bangkok, Thailand: IEEE.

Collins, K. (2007). In the loop: Creativity and constraint in 8-bit video game audio. *Twentieth-century music*, *4*(2), 209.

Collins, K., et al. (2008). *Game sound: an introduction to the history, theory, and practice of video game music and sound design.* Mit Press.

Fritsch, M. (2013). History of video game music. In *Music and game* (pp. 11–40). Newyork: Springer.

Hsu, J.-L., Liu, C.-C., & Chen, A. L. (2001). Discovering nontrivial repeating patterns in music data. *IEEE Transactions on multimedia*, *3*(3), 311–325.

Hunt, S., Mitchell, T., & Nash, C. (2018). A cognitive dimensions approach for the design of an interactive generative score editor. In *Fourth international conference on technologies for music notation and representation, montréal, canada.* (pp. 1–8). Montreal, Canada: TENOR.

Hunt, S., Mitchell, T., & Nash, C. (2019). Automating algorithmic representations of musical structure using igme: The interactive generative music environment. In *Innovation in music 2019* (Vol. 1, pp. 1–18). University of West London: Innovation In Music.

Hutchings, P., & McCormack, J. (2019). Adaptive music composition for games. *IEEE Transactions on Games*, 1 –1.

Jhamtani, H., & Berg-Kirkpatrick, T. (2019). Modeling self-repetition in music generation using generative adversarial networks. In *International conference on machine learning.* Long Beach, California: International Conference on Machine Learning.

Kemerer, C. F., Dunn, B. K., & Janansefat, S. (2017). Winners-take-some dynamics in digital platform markets: A reexamination of the video game console wars. *University of Pittsburgh, PA.*

Lerdahl, F., & Jackendoff, R. S. (1996). *A generative theory of tonal music.* Cambridge, Massachusetts: MIT press.

Lu, L., Wang, M., & Zhang, H.-J. (2004). Repeating pattern discovery and structure analysis from acoustic music data. In *Proceedings of the 6th acm sigmm international workshop on multimedia information retrieval* (pp. 275–282). New York NY USA: ACM SIGMM.

Rahn, J. (1993). Repetition. *Contemporary Music Review*, *7*(2), 49–57.

Sakamura, K. (1999). Entertainment and edutainment. *IEEE Micro*, *19*(6), 15–19.

Shih, H.-H., Narayanan, S. S., & Kuo, C.-C. (2001). Automatic main melody extraction from midi files with a modified lempel-ziv algorithm. In *Proceedings of 2001 international symposium on intelligent multimedia, video and speech processing. isimp 2001 (ieee cat. no. 01ex489)* (pp. 9–12). Hong Kong, China: IEEE.

Temperley, D. (2007). *Music and probability.* Cambridge, Massachusetts: Mit Press.

Appendix

Table 1. A table summarising the various video game consoles used in this study.

Generation	**Console**	**Audio group Type**	**Media Type**	**Year of release**
3rd	Sega Master System	B	Cartridge	1986
	Nintendo entertainment system	A	Cartridge	1983
4th	Super Nintendo	B	Cartridge	1991
	Sega Megadrive	B	Cartridge	1988
	SNK NEOGeo	B	Cartridge	1990
5th	Playstation	C	Disk	1994
	Sega Saturn	C	Disk	1994
	Nintendo 64	C	Cartridge	1996
6th	Playstation 2	C	Disk	2000
	Sega Dreamcast	C	Disk	1998
	Nintendo Gamecube	C	Disk	2001

The end.