Approaches to AI for Musical Performance

Brett Milford
MCTN

School of Agricultural, Computational and
Environmental Sciences
University of Southern Queensland

November 2018

Submitted in partial fulfillment of
the requirements of the award of
Master of Computing Technology

Abstract

Computers play an essential role in the composition, engineering, performance and distribution aspects of the modern music production paradigm, however there exists significant barriers to the wide spread use and adoption of AI techniques in the broader music community. I present a design for a rhythm generation application and in doing so establish a method by which a computer may play the role of a performing musician. As a precursor to this, a history and discussion of pertinent issues related to the computational model for music is presented, highlighting key values a new design should espouse. A design and implementation for a novel rhythm generation system based off Markov Models and exploiting the values highlighted is presented and an evaluation method for testing the efficacy of this system is devised. In an instance of this evaluation presented in this paper (Milford, 2019, Chapter 5), the implementation averaged better than a purely random algorithm however the population of data collected was insufficient to draw a conclusive result. A wealth of knowledge was gained from the process which showed a number of promising attributes for the design. In particular, it was recommended that further research should consider a hybrid approach of the design presented here, with training techniques typical of deep learning methods.

Certification of Thesis

The work presented in this thesis is, to the best of my knowledge and belief, the original work of *Brett Milford*, except as acknowledged in the text. The material in this thesis has not been submitted, either in whole or in part, for a degree at this or any other university.

(Brett Milford) Date

Signature of Supervisors Date

Acknowledgements

I would like to thank Associate Professor Richard Watson for his ongoing support in the development of this dissertation, and Dr Xiaohui Tao for his help in finalising this project. I would also like to thank my family and my partner Ms Kate Collis for her personal support in seeing me through this degree.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This dissertation presents a study and system design for the implementation of Artificial Intelligence (AI) techniques in the extension of musical performance. As a precursor to this, the themes present in music programming languages and the inherent difficulties of adequately representing and computing musical abstractions is examined and discussed. Subsequently it delves into the literature in the field of Music AI to inform and guide the theory pertaining to the development of one such system. The impetus for this research is to investigate how a computer may play the role of a human in undertaking the various activities of a performing musician. As such this dissertation focuses on the design elements and considerations necessary for emulating the functions of a musician's performance.

Computers play an essential role in composition, engineering, performance and distribution within the modern music production paradigm. In these roles, they typically work to emulate pre-existing non-digital systems. There is however, a second extensive history in avant-garde musical circles of employing the processing and algorithmic capabilities of computers to provide unique musical ideas and sounds in compositions (Taruskin, 2019, Chapter 10). In this regard there is a precedent for extending the capabilities of the musician through computational means. With the application of modern AI and Machine Learning (ML) techniques for data processing and information extraction, research in this cross-disciplinary field will extend our knowledge and understanding of both the music and AI domains.

Currently there exists significant barriers to the wide spread use and adoption of AI techniques, including issues of how best to represent and manipulate musical structures in a manner which captures the abstract and multi-layered

characteristics of music (Balaban, 2003). Furthermore, whilst music may bear some similar traits to problems of Natural Language Processing (NLP) or Computer Vision, modern AI tools developed and refined for these areas are unlikely to produce coherent results without thorough optimisation or re-engineering for the music domain (Hutchings, 2018). In light of this, I posit that it is possible to produce a system which both performs better than random chance, and to the typical listener may be perceived as human in origin.

Ultimately my motivations align closely with Ingalls (1981) ethos for the Smalltalk language – "Any barrier that exists between the user and some part of the system will eventually be a barrier to creative expression". These words resonate more broadly than the bounds within which they were conceived, as research in the field of music and AI will play an important role in bridging the divide between scientific and creative domains.

1.1 Scope, Limitations, and Content

This study approaches the field of AI from the context of music, and as such presents and focuses on pertinent issues in reconciling the intricacies of music with computational methods, as opposed to retrofitting preexisting AI tools to music problems. Because of this, emphasis is attributed to the musical implications and outcomes rather than the underlying computer science theories. The scope of the implementation, it's design and the techniques used were limited in depth by the time constraints imposed on this project.

With this in mind, the project was limited in scope to producing a simple system for generation of stylistic but novel rhythms. This decision was made because it highlights a specific function that a musician may perform and is a crucial element of a technique employed in jazz and popular idioms called 'Comping' (Witmer, 2003). The implementation was designed to utilise features and abstractions of the Euterpea library (Hudak, 2014) and was limited to the production of Musical Instrument Digital Interface (MIDI) (Burnand, 2001) instructions. All other components of the tested system (such as production of the sonic artefacts) were deferred to other computer applications.

Given the defined scope, this dissertation contributes the following knowledge:

- A review and contextualisation of the computer-music and Music AI fields.

- A system design and architecture for a rhythm generation application.
- An implementation of this design.
- An evaluation design for assessing Music AI systems.
- An analysis of data from the evaluation of the implementation conducted.

The next chapter will present historical and contextual information from the computer-music domain, in particular, analysing the lineage of current systems and highlighting issues of taxonomy and representation which need to be considered in contemporary work. It will explore the theory and design of Music AI systems and inform the design and evaluation choices in the proceeding chapters. Chapter 3 will articulate the details of a system design and architecture for a rhythm generation application, drawing on theory and research presented in the preceding chapters. Chapter 4 will detail the evaluation methods that will be used to determine the efficacy of the implementation from the previous chapter. In particular it contributes a method of empirical assessment for similar systems. Chapter 5 will present the results of the evaluation method described in the previous chapter and analyse this data for themes and trends, to ascertain the effectiveness of the approaches used in achieving the desired outcomes. Finally, Chapter 6 will draw together my conclusions from the study, highlighting the achievements and implications of this work as well as the avenues for further research.

Chapter 2

Music Programming Languages and AI Techniques

This chapter reviews the contemporary literature involving the developments in music oriented programming languages as well as the research conducted in incorporating AI techniques into musical problem domains. A review of both areas is necessary to fully understand the issues faced in developing Music AI systems, and serves to inform both the system design and evaluation choices articulated in the following chapters. The first section provides a brief history of music programming languages and their legacy on language conceptualisation and design. The second looks at models and constructs for music in computing. The final section reviews definitions of AI and its subcategories, and is followed by an analysis of three categories of Music AI systems.

2.1 Music Programming Languages

2.1.1 The MUSIC-N Languages

Programming languages designed expressly for the manipulation of music have an extensive history. Thompson (2018) demonstrates a proliferation of music programming languages from as early as the mid 1980's. These languages vary greatly in structure and scope and various authors (Lazzarini, 2013; Roads, 1996; Bresson and Giavitto, 2014) have provided a discussion of the dominant languages in this field which have proved most influential in the development of computer music.

The original 'MUSIC' languages (MUSIC, MUSIC II, MUSIC III, ..., V,

or MUSIC-N, as they are commonly collectively referred to) (Manning, 2013) played a dominant role in early computer-music work. Lazzarini (2013) discusses the MUSIC-N languages; their history, development and their influence in pioneering what is now a common architecture of modern low level music languages. This architecture includes the segregation of sound generation routines and sound composition routines. A direct, modern descendent of these programs is Csound (Csound, n.d) which has much of the same features and designs of the original MUSIC-N languages. Lazzarini also discusses two modern visual programming languages, Max (Cycling '74, n.d) (named after the original author of MUSIC – Max Matthews) and Pure Data (Pure Data, n.d) which, although less obvious, are also descendent from the MUSIC-N decades. The seminal book of Roads (1996) expands upon the importance of these languages and their design, highlighting the influence of pre-existing musical devices such as music sequencers, on subsequent languages. The text as a whole provides in-depth discussion of computer audio concepts and is considered to be the essential source material.

Roads exposition on the MUSIC-N and similar languages engages in a detailed discussion of the workings of this architecture. Input components to this system are A) an Orchestra file and B) a Score file. The Orchestra file provides a description of the instruments in the sound synthesis language, whilst the Score file orchestrates the sounds of these instruments. The sound synthesis language enables the creation of elaborate unit generators which are distinct functional units that describe the methods of sound processing in a detailed, low level manner. These are often combined in various ways to produce the sound of the instruments (Roads, 1996, Chapter 17). Recently the synthesis concepts fundamental to these languages have been broadened to include burgeoning areas of sound synthesis and creation; as such Nishino et al. (2016) formalised their approach to a particular field of synthesis called Granular Synthesis.

Nishino et al. utilise concepts similar to the unit-generator found in Csound to approach the problem of Microsound Synthesis (Roads, 2004, Chapter 5) – a particular form of synthesis related to Granular Synthesis (Roads, 2004, Chapter 3), in which audio is sliced in to very small parts and reconstructed in a pattern or order different to the original. Nishino et al. (2016) describe their programming model for Microsound Synthesis as "simpler and terser" than existing unit-generator languages and in turn provide an insight into the de-

sign considerations that go into the creation of music programming languages, including a critical discussion of the design decisions of existing languages.

2.1.2 Visual Programming Languages

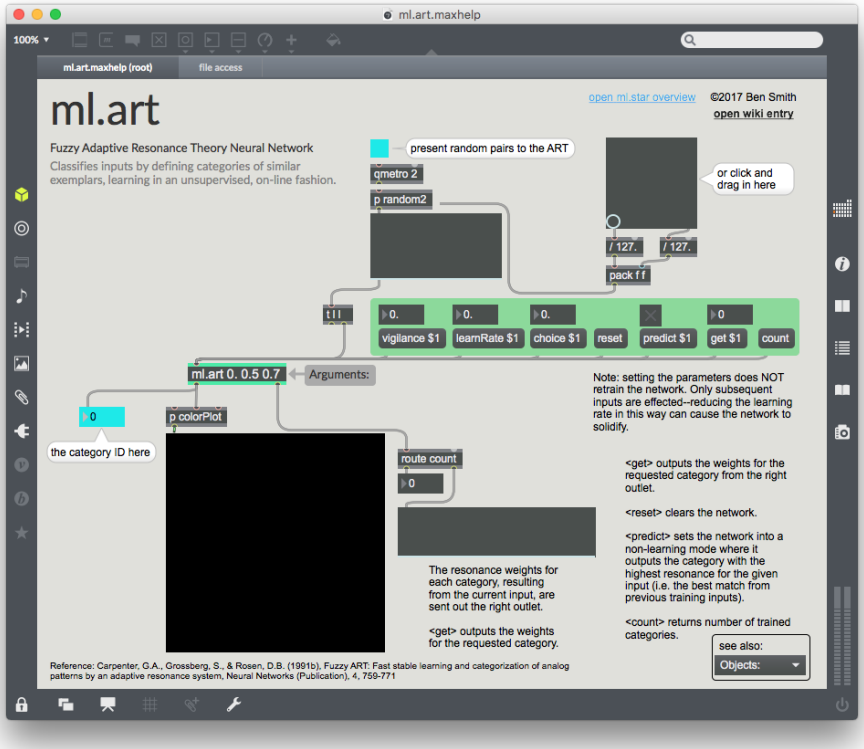

Figure 2.1: An example of the patch chord motif used in Max "Patches" to connect data and audio to functions and Digital Signal Processing (DSP) units.

Visual programming languages have provided another dominant metaphor for the design of common music programming languages of which Max (Cycling '74, n.d) and Pure Data (Pure Data, n.d) are the most common. To understand the genealogy of their design Roads discusses the design and history of music sequencers, which he describes as rudimentary "computer performers", dating back as far as the 13th Century (Roads, 1996, p. 662). Max and Pure Data, which is an Open-source fork of an early version of Max, simulate the physical motifs of patch chords and electrical signals found in early sequencers and analogue electric synthesises (Figure 2.1). These similarities lessen the cognitive load for users transitioning from music studio hardware to a software based system, and at the same time expose fundamental computer program-

ming concepts such as variables and conditionals to the user, and transparently encapsulate higher level language concepts such as functional abstraction and object oriented design.

Bresson and Giavitto (2014) further the discussion of visual programming and declarative languages with the introduction of their real-time, reactive extension of the Open Music – a visual, Domain Specific Language (DSL) designed for music composition. Their article compares and contrasts the programming model of 'Open Music' with that of Max, Pure Data and Open Sound World (CNMAT, n.d) contributing extensively to the discourse on music programming languages. It delves into Open Music, which is implemented in Common Lisp and shares features with similar DSLs (Hudak, 2014). Bresson and Giavitto have observed that the verge between 'compositional' architectures such as Csound, which are designed with separate composition and performance phases in mind, and 'performative' designs such as Max, which are characterised by a shorter feedback loop and the ability to modify output on the fly, are a "frustrating limitation" in developing creative music projects. They stipulate that "their convergence would allow for a tighter coupling between composition and performance"(Bresson and Giavitto, 2014, p. 3), which forms the justification for the design choices of their language.

2.1.3 Algorithmic Composition

Algorithmic composition is a category of musical composition and study that developed alongside early computing and programming environments, taking design and aesthetic queues from the underlying systems. Roads (1996, Chapter 18) reviews the history of algorithmic composition systems, which the author defines as "systems designed to produce music autonomously given in a procedural manner". Although formalised as a school of study in the second half of the twentieth century, examples that mimic the underlying concepts of this style of composition have been documented much earlier in the Western Classical music tradition under the term Aleatory Music (Hedges, 1978). The dominant example being 18th and 19th Century "Musical Dice Games", notable ones being written by composers such as Mozart, where pre-written fragments of waltzes were selected based on a dice roll and performed in sequence (Ruttkay, 1997).

Roads (1996, Chapter 18) divided the field into Deterministic and Stochastic methods and looked explicitly at three systems designed by composer-

programmers such as Iannis Xanikas, and the methods and work-flows involved. Here Roads discusses philosophical issues amongst musicians of how these techniques and tools should be used. For example, is the output of such systems considered original music of the composer or is it simply 'found music'? Should the composer rewrite portions of the music after it is output, or should be composer change the algorithm and recompute the score? Though seemingly irrelevant from a computer science lens, these questions still plague composer-programmers today. Supko (2015) details one such journey in developing, what he describes as "a silicon-based life form to help make music that mere carbon-based life forms could never imagine on their own". In particular, Supko feels strongly for refining work that is output from their system by hand. He views his creation as an intelligent, and indiscriminate information processor, that ultimately serves to compliment human creativity rather than supplant it. There are similar arguments for the purist's approach of altering the algorithm and not the output, however, more importantly, when computer scientists are inevitably asked to consider the ethical consequences of their AI research (Gibney, 2018), the history of reflection in the computer-music paradigm may serve to rationalise and contextualise future discussion around artificial intelligence.

Roads (1996, Chapter 19) catalogues and describes a number of strategies for algorithmic composition in his chapter by the same name. These include systems theory, stochastic processes, language grammars, constraint programming, expert systems and neural networks. Roads work lays the ground for what would later transition into the umbrella term of AI and Machine Learning. For instance Toiviainen (2000) classifies representational approaches, such as Language Grammars or Expert Systems, as traditional Symbolic AI, whilst the author appears to advocate for Connectionism AI techniques which would now be referred to as Artificial Neural Networks (ANN). Algorithmic composition remains a salient approach to understanding and building up complex and intelligent systems in addition to the plethora of modern ML techniques.

2.2 Computational Model for Music Applications

The nature of music is a phenomenon which is multi-faceted, many-layered, and continually changing. As such, the development of a unified computational

model for music which would allow researchers to reason about components of one or more systems from a fundamental standpoint is yet to be achieved. This is an issue that was highlighted by Balaban (2003).

Balaban explains how research in the field of computer music is directed towards the development of narrow systems for treating highly specific problems, such as disparate systems for music composition, performance, analysis, training and musical information classification. Each of these systems leads to the inherent development of computational models which, for instance, introduce theoretical frameworks or techniques from other fields, such as Artificial Neural Networks, Concrete Abstraction from Lambda Calculus, Systemic Grammar approach from Linguistics (Balaban, 2003). In developing these systems, a non-trivial degree of musical knowledge and intuition are embedded within these systems but are formed from no discernible foundation.

This can be observed more readily in the few dominant standards in widespread use in the computer-music space. Pulse-Code Modulation (PCM) (Fine, 2008) audio is one of these few standards; it provides a description of the physical phenomenon, the oscillation of atmospheric pressure to produce sound waves, which provides the raw material from which music can be produced but makes no pretence at being an accurate description of music itself. Another example is Musical Instrument Digital Interface (MIDI) (MIDI Association, n.d) which has become the de facto standard for the digital description of music which reduces music to a time series database of events. For systems which deal in musical data, these are two standards in which they may claim parity, however every other layer; architectural design choice, or instance of embedded musical knowledge or understanding, is siloed by the lack of research and commonality in the computational basis for music itself.

Features of systematic decoupling required to refine a computational model of music can be found in emerging projects which seek to rethink the music language paradigm. McCartney (2002), the author of SuperCollider, a music programming language and sound synthesis engine, provides this perspective of computer music tools.

> The kinds of ideas one wishes to express, however, can be quite different and lead to very different tools. If one is interested in realizing a score that represents a piece of music as a fixed artifact, then a traditional orchestra/score model (Csound, MUSIC-N) will suffice. Motivations for the design of SuperCollider were the ability

> to realise sound processes that were different every time they are played, to write pieces in a way that describes a range of possibilities rather than a fixed entity, and to facilitate live improvisation by a composer/performer. (McCartney, 2002, p. 1)

McCartney's point touches on the tensions and inadequacies of current standing solutions. The authors design rationale for SuperCollider contains a number of systematic improvements over the programming environments introduced thus far. Those improvements include, a fundamental decoupling of sound generation systems and scoring languages into a client server architecture, the use of an open source protocol for communication between components, a familiar C like syntax for the scoring language, and a C++ application programming interface for creating sound generation new units. These improvements have enabled the development of projects such as Overtone (Aaron and Rose, 2018) which reuse SuperCollider's sound generation components, and combine them with relevant features and libraries of its parent language Clojure (Clojure, n.d). Already this has seen the adoption of external Clojure libraries such as Quil (Quil, n.d), and ShaderTone (ShaderTone, n.d) to coordinate visual elements with audio.

In the context of Balaban (2003), SuperCollider offers progress in the way of decoupling elements of music creation. In assessing SuperColider's scoring language against Balaban's criteria it yields some shortcomings. The scoring language excels as a language for performance, and many forms of composition, however it is not a reusable description of music that is useful for many types of analysis as well as training and classification scenarios. Furthermore it diverges significantly from Western Classical notation and aims to offer abstractions for programmaticly controlling synthesizers and as such supports the latter better.

Due to the decoupled nature of SuperCollider and its open standards communication protocols, its components can, and are being reused in more general purpose languages. Conceivably, one could develop a model which abstracts over different layers of musical elements (from signal processing to notation systems), but similarly decouples itself from any particular sound synthesis implementation details (like SuperCollider's sound synthesis engine). In turn these diverse use cases of analysis or training and classification could be catered for by separate languages as is the case with Overtone (Aaron and Rose, 2018) whilst still using large portions of the system. To produce a truly robust model, Balaban (2003) highlights a number of key concepts and elements which would

need to be addressed; this includes the investigation of levels of primitives, parameters and abstraction methods. Balaban suggests a promising avenue for establishing this would include research into "musical schemas" (Leman, 2012).

Janin et al. (2013) present a recent endeavour to reconcile the difficulties found in defining a terse musical model with the realities of requirements for concrete computation, but with a decidedly different approach. They present a mathematically defined proposal for combining space modelling and time programming into a single feature called spatio-temporal tiled programming. Hudak (2008) describes a similar concept, in a paper which defines atomic rules for Polymorphic Temporal Media, with the justification that considerable work had taken place in embedding semantic descriptions in multimedia (e.g. via XML, UML, and the Semantic Web) but not on formalising the semantics of concrete media. Hudak's work, whilst media generic (the author provides examples of the application of its principles to both music and animation), does have a concrete implementation in the Haskell-based library – Euterpea (Hudak, 2014). Janin et al. (2013) makes strong comparisons both to Hudak's thesis and the Euterpea library, stating that "These languages provide strong abstraction design principles that can be used efficiently when programming musical applications" (Janin et al., 2013, p. 23). The authors work differs fundamentally in that it aims to further generalise over the atomics of sequential and parallel spatial dimensions through the introduction of 'tiling semi-groups', to allow the combination of musical objects in both dimensions with a single operator, the 'tiled product'. Janin's work (Janin et al., 2013; Janin, 2016) appears to focus solely on the music domain and it is unclear if these principles could be applied to other forms of temporal media. Furthermore, the proofs remain purely academic and whilst rules may be able to be retrofitted onto a library such as Euterpea, there is no apparent implementation of this theory to date.

The functional programming paradigm presents clear advantages for the representation and composition of musical processes and systems (Janin et al., 2013; Janin, 2016; Hudak, 2008, 2014; Ingalls, 2018). Programming languages have the tendency to fundamentally influence the design of the systems they build. Languages in the pure functional paradigm promote the strong encapsulation of data, the design of routines without side effects, reusable functions with the properties of function composition (Mills, 1975), and in turn the production of detailed levels of abstraction. As highlighted, music in the general

sense exhibits all these features; for example, it can be represented at numerous levels, from low level signals to high level notation systems and its operations are individually deterministic and able to be arbitrarily composed. An example of the effective exploitation of pure functional techniques in the music language design is Faust – Functional Audio Stream (GRAME, 2017).

The Faust programming model combines functional programming with features of synchronous and signal processing languages such as Lustre (Verimag, n.d), Esterel (Berry et al., n.d) or SIGNAL (Polychrony, n.d) and it compiles to standard C++ for performance and integration with common application frameworks. It utilises algebraic block-diagrams for function composition and specifies 5 binary composition operators which define how discrete signal processors are merged (GRAME, 2017, Chapter 3). The Faust model focuses specifically on the Digital Signal Processing (DSP) problem domain, and produces highly optimised implementations, however it does not provide higher level representations required for a robust model and diverse use-cases. Euterpea is a DSL written in Haskell which does provide higher level musical representations, furthermore it has features with semantic similarities to Faust such as arrow combinators.

Euterpea (Hudak, 2014) is a language for expressing and manipulating musical structures at 'Note' and 'Signal' levels. Evolving out of Haskore (Hudak, 1996), a project of research into semantics and features of temporal media; Euterpea aims to differentiate itself from systems by utilising the abstraction capabilities of Haskell. In working towards Balaban's ideal of a computational basis for music, Euterpea presents the most pragmatic and considered approach both for the theoretical considerations of building music systems, and the practical considerations of implementing and integrating diverse frameworks and systems.

2.3 Definitions of Artificial Intelligence

Artificial Intelligence is a broad field of enquiry with an extensive history of development alongside that of computational machines. Paths of enquiry in this field have remained relatively consistent whilst nomenclature and taxonomy of particular methods have evolved considerably. The following definitions and their aliases aim to capture the categorisation most commonly in use, whilst highlighting historical terminology which is found throughout the literature.

Artificial Intelligence is the broadest categorisation in use. Subsequent categories are generally considered narrower definitions, encompassed by the umbrella of AI. Early approaches to AI involved the development of expert systems which derived their knowledge base from encoded rule base patterns (also referred to as rule base systems and categorised as Symbolic AI). A number of effective Music AI systems were developed in this manner including the commercial system Band in a Box (Rolland and Ganascia, 2013).

Machine Learning (ML) is defined as a subcategory of AI algorithms and techniques which model the problem domain based on learning mechanisms. Within this category Deep Learning is routinely used for classification and prediction tasks, and utilises a repertoire of ML techniques based on Artificial Neural Networks (ANN), sometimes referred to as Connectionism AI.

The key aspect of labelling approaches as Deep Learning, is the utilisation of multi-layered processing, or multi-levelled abstractions for defining complex representations in terms of simpler ones. Machine Learning and Deep Learning have received renewed attention due to advances in data capacity and management, and the increase in parallel computing performance found in modern multi-core and General-Purpose GPU machines. The GPGPUs allow the computation of the traditionally performance intensive, highly parallel algorithms, whilst the CPUs allow the utilisation of large data sets for training inferencing models, by providing and coordinating access to ancillary data input/output devices such as network interface cards and storage controllers. Other techniques and algorithms which are typically categorised as machine learning (but not ANN) include genetic and linear regression algorithms, dynamic programming and other deterministic methods (Briot et al., 2017).

Briot et al. (2017) provides a thorough analysis of the field for Deep Learning techniques as applied to music. My survey approaches the Music AI field from the music perspective, categorising approaches based on their context to music rather than to AI. Furthermore, a ML centric approach to Music AI would not only forgo the observation of other interesting techniques, but would ignore qualitative aspects of building Music AI. As Hutchings (2018) observes, the models designed and refined for natural language processing (one of the more common problem domains for ML techniques) "probably aren't going to be very effective 'out-of-the-box' tools for music analysis and generation". This is due to both the fundamental differences between music and language and the reality that training data of reasonable accuracy and depth is heav-

ily skewed toward genre's of Western Classical traditions, thus limiting the inferencing abilities of models developed by this method.

2.4 Music Artificial Intelligence

The application of AI in music has an extensive history which progressed alongside the development of Algorithmic Composition. Roads (1985) asserts that the field of AI research consists of two broad categories; cognitive science: devoted to the development of theories of human intelligence, and applied AI: the engineering aspects of AI. Both Roads and Laske (1992) were heavily influenced by cognitive scientist Marvin Minsky (Minsky, 1974) in developing aspects of and approaches to AI. Roads highlights a lack of flexibility with commercial hardware and software as the primary barrier to exploration of this space, however his distinctions are fundamental in focusing and clarifying the lens of Music AI research.

Marsden (2000) elaborates on the definition of Music AI and approaches used within the discipline. He highlights issues with the representation of knowledge and with symbolic and subsymbolic (Leman, 1989) representation and processing in modelling musical behaviour. Marsden points out it is impossible for Music AI to mimic human intelligence in full as he asserts, "its impossible in principle for computers to pass a musical version of the Turing test" (Marsden, 2000, p. 15). This paper provides an interesting, contemporary discussion of the field and talks in-depth about symbolic and data representation approaches including musical grammars, and Connectionism AI approaches such as neural networks. Marsden's work is particularly pertinent as it highlights similar tensions of musical representation and processing experienced by contemporaries such as Balaban.

Broadly speaking the field of Music AI research breaks into three narrow fields, with methods and approaches and evaluative frameworks specific to their area. These are 'Composition AI', 'Performance AI' and 'Representation and Identification AI'. The following sections will touch on the techniques used in each field to gain an understanding of the context in which my methods will be based.

2.4.1 Composition AI

A core outcome of composition AI is the production of static musical artefacts or data. In the abstract, systems typically take some form of input, derive some knowledge, then use this to produce the desired output. Most approaches achieve this in diverse ways which are tailored to the particular features they wish to focus on.

Phon-Amnuaisuk (2003), presents an approach to the intelligent harmonisation of chorales. Chorales are a particular style of composition which involves the use of two or more 'voices' in the design of melody, harmony and counterpoint. Chorales are unique in that they are a very direct application of problem solving under constraints. At each point in the chorale the composer must satisfy a set of constraints such as harmonisation of melody, voicing of chords in a manner that requires the least movement by each part (voice leading) and alignment of the chordal progression with melodic and lyrical cadences etc. The author's approach involves taking rules (e.g. the chord transition from the sub-dominant to the dominant chord must occur via voice leading and the melody must be in the tenor line) and directing the application of these rules in searching for potential harmonisation of a given monophonic line.

This pattern of design is consistent with systems typically found in Music AI. In this instance the author contributes a well structured approach to defining music theory rules of choral harmonisation. Colombo et al. (2016) utilise a similar structure, but harnesses ANN for the knowledge derivation component of their system.

In designing this approach Colombo et al. acknowledge that they sought a solution which was both easily trainable and capable of reproducing the long-range temporal dependencies found in music (e.g. a song structure over a range of minutes to hours). This system involves input and training on a large corpus of melodies to generate output which is "coherent with the style they have been trained on". In particular it employs gated recurrent unit networks (Chung et al., 2014) – an optimised form of recurrent neural networks which "have been shown to be particularly efficient in learning complex sequential activations with arbitrary long time lags" (Colombo et al., 2016, p.1).

Quick (2014) presents a number of innovations in design and implementation of a Music AI system. The author's system employs a system of musical grammar definitions, as well as formulas for creating an alphabet of chord sequences. Of particular interest is the experimental methodology used, which

employs the musical equivalent of a 'Turing Test', in which participants are surveyed on their perception of the music being man or machine made. Quick's method highlights the effectiveness of a language grammar inspired approach which is deterministic and readily comprehensible in nature than, for instance, Deep Learning techniques.

2.4.2 Performance AI

The field of Performance AI is diverse in its approaches and outcomes, and does not necessarily generalise to the input/output patterns found in Composition AI. For instance Widmer (2001), focuses on applying AI study to develop models of 'musical expression'; whilst Pachet (2003) presents an interactive music performance system which adapts to its inputs in real-time. The uniting theme in this section of research is the analysis and modelling of music performance in real-time.

Widmer (2001) discusses the application of Machine Learning, to empirical musicology, in order to "study the phenomenon of Expressive Music Performance (or 'Musical Expression') and to inductively build formal models of (particular aspects of) expressive performance from real performances by human musicians" (Widmer, 2001, p.149). The author's model aims to utilise machine learning's data mining and analysis capabilities to investigate patterns across a large corpus of data, and develop insights and algorithms for more intelligent music applications. This approach treats musical expression as a Big Data problem and the data collection model appears to favour instruments with highly discrete rhythmic and tonal qualities (e.g. piano, rather than trombone). Igarashi et al. (2003) study musical expression with a method that is agreeable to all types of instruments by analysing the respiration characteristics of musicians.

Igarashi et al. (2003) investigates respiration in musical performances with the objective of defining rules pertaining its characteristics in various musical contexts. The study discovered, that players tend to exhale at the beginning of new large musical structures, and inhale at key changes. The study utilised Inductive Logic Programming (ILP) to analyse performance data. ILP provides a framework of inductive inference, based on predicate logic which utilises similar technologies to expert-systems designed in languages such as Prolog.

Miranda (2003) approaches the tangential issue of understanding how an agreed corpora of music are developed and utilises a method of computational

simulation to achieve enlightening results. Miranda presents a simulation of basic autonomous agents interacting to produce a common repertoire of intonations. The study is motivated by an interest to simulate the real-time development of communicative sounds within a society from scratch and achieves this through modelling and computational methods. The author finds it is in fact possible to simulate the development of musical intonations within a society of agents, and that they do, after sufficient interaction, develop a shared model and understanding. The work highlights the implications this has for musicology disciplines as well as our understanding of mimetic learning.

Pachet's system *The Continuator* (2003) seeks to reconcile interactive, performance oriented systems with composition oriented systems. In the development of this, and complementary projects at Flow Machines (Sony, n.d), Pachet and his colleagues aim to extend the technical ability of musicians with stylistically consistent, automatically learnt material. The author explains his motivations as follows:

> The undertaking can be seen as a way to turn musical instruments from passive objects into active, autonomous systems, with which one can interact using high-level controls, much in the same way Claude Nougaro, through the blink of an eye, can control, or influence his pianist Maurice Vander. (Pachet, 2003, p. 119)

The system is based on a Markov Model of musical styles that is augmented to account for efficient real-time learning and development of style. Markov Models are a statistical method for modelling a system or series of events, in which it is assumed that future states depend only on the current state and not on events that occurred before it (Rabiner, 1989). Thematic material is learnt in real-time whilst users may direct the system in elements such as form and range with tactile parameters. The implementation is particularly effective in highly improvised styles and demonstrates the effectiveness of probability and Markov Model based systems in producing qualitatively good results. A similar performance oriented system was developed by Baird et al. (1993), but with a different emphasis on qualities of musicianship. Baird et al., looked at implementing a music Performance AI for the Macintosh II. The system reads incoming MIDI data and matches it to a score in order to determine where the musician is located in the playback of a piece. The intention is to take the computer from being a "virtuosic tape recorder" to being an adept ensemble

musician that can react to changes in time and other musical events. The article discusses a tracking algorithm that takes MIDI data and interprets it correctly to the score, as well as the methods it employs to work with fluctuations in tempo. Whilst the code and implementation details are unavailable, the discussion aids in the development of architectures and approaches to solving difficult problems in the real-time space.

The influence of Baird et al. (1993) can be found in numerous studies and systems including De Mantaras and Arcos's *SaxEx* (2002). The system is an approach Performance AI, utilising case based reasoning which works with monophonic recordings to develop models of performance from tacit knowledge learned through observation. The authors divide the field of Music AI into compositional, improvisational and performance systems. Their system produces improvisations based on the material learnt, however the study does not specify how their system (or any other 'improvisation system') is fundamentally different from a composition system. Improvisation in music has traditionally been a performance art, requiring attention to context, real-time events and expressive nuance, which can not be captured in a static composition, and typically varies considerably with each performance.

In a related study, Ramirez and Hazan (2005) develop a system for learning such expressive nuance from recordings of a similar genre. Ramirez and Hazan describe an approach to learning expressive performance rules for monophonic Jazz standards. It uses a melodic transcription system which extracts a set of acoustic features from audio recordings, then applies genetic algorithms to induce rules of expressive performance. It aggregates the rules produced during different runs that are of musical interest and have good prediction accuracy. This work is explored further in another study (Ramirez and Hazan, 2006) which presents a machine learning approach to modelling the knowledge applied by a musician when performing a score. It describes a tool for both generating and explaining expressive music performances of monophonic jazz melodies. The system consists of a melodic transcription component, a machine learning component and a melody synthesis component which generates expressive monophonic output.

These papers present state of the art examples of systems and approaches to music Performance AI, and contribute to our knowledge and understanding of systems and processes in both the music domain and AI research.

2.4.3 Representation and Identification AI

Music Representation and Identification AI is primarily concerned with developing, deriving and constructing musical knowledge. In contrast with Composition and Performance AI, Representation and Identification AIs do not necessarily contribute new sonic artefacts, but instead hone and refine tools for the analysis and interrogation of musical constructs. These systems often contribute to, or form part of a broader system, but also find application in musicology disciplines for researching and drawing conclusions across broad repertoires.

The nature of this style of AI lends itself to investigating the difficult issues of computational models for music highlighted by Balaban (2003), in particular Bod (2003),Chew (2003) and Conklin (2003) grapple with related issues in their work. Bod (2003) takes an in-depth look at developing a parsing model based on generic assumptions of how the mind perceives patterns. It combines the two common approaches to parsing, simplest structures first, and probable structures first into a method which looks for the simplest structure in a group of the most probable structures, or looks for the most probable structures in a group of the simplest structures. This study looks to unify approaches to natural language processing and Music AI with the assertion that the faculties for each (and potentially visual domains) share commonalities. This approach utilised Data-Oriented Parsing, which learns grammar by extracting subtrees from a tree-bank, and processes data by combining these sub-trees to analyse the new input.

Chew (2003) utilises a boundary search algorithm for determining points of modulation in a piece using the Spiral Array (Chew, 2000), an abstract geometric model for describing tonality and harmonic relationships. Chew describes their findings succinctly in the following:

> Comparisons between the choices of an expert listener and the algorithm indicates that in human cognition, a dynamic interplay exists between memory and present knowledge, thus maximizing the opportunity for the information to coalesce into meaningful patterns. (Chew, 2003, p. 18)

Conklin (2003) describes a new method for discovering patterns in the vertical and horizontal dimensions of polyphonic music. The method works to both detect common structure, but limit the patterns found to those that

have musical or statistical significance. This research also defines an ontology for the description of a musical abstract data type which has similarities with Haskore (Hudak, 1996) and the Open Music environment.

Each of these authors contributes a perspective to Balaban's discussion of the computational basis of music and in doing so highlight how perception and cognition play a unique role in developing Music AI systems. The following authors further this discourse in establishing cognition as a driving factor in the design of these AI systems.

Dannenberg and Hu (2003) derive descriptions of music from audio recording inputs. The paper highlights that "recognition and understanding" are not well defined in the field of music and even though music exhibits internal logic and rich structures, no general theory of semantics exists. The paper asserts the premise that an important part of music understanding is the identification of repetition within music, which in turn generates structure.

Pikrakis et al. (2003) provide a method for recognising musical patterns in monophonic audio using Discrete Observation Hidden Markov Models. Hidden Markov Models differ from simpler Markov Models such as Markov Chains, by way of the fact that each state within the series of events is not directly observable (Rabiner, 1989). In essence Pikrakis et al. (2003) presents a system for recognising musical patterns based on the premise that certain musical patterns have been shaped and categorised through practice and experience over many years; similar in notion to the simulated development of a shared repertoire presented by Miranda (2003).

Povel (2003) presents a concise description of a computational model for the processing of tonal melodies. The model is based on the assumption that a tone sequence is represented in terms of a chord progression underlying the sequence.

Spevak et al. (2003) explores the ambiguity found in attempting to determine melodic segmentation. The study utilises a corpus of melodies, annotated with melodic segments by hand by musicians to inform a probabilistic framework for modelling ambiguity. A system for determining boundaries that integrates both the segment boundaries and lengths that the musicians preferred is presented.

Spiro (2003) investigates the usage of note-length as the sole input to a model of the "perception of music heard", which aims to determine time signature and phrasing. The author developed two models, one using rule-based

grammar and one using a combination of rule-based grammar and memory-based approach. The author finds that second method is the most successful and improves predictions over previous approaches.

Temperley (2003) reinterprets the 'key-profile model' approach to determining musical keys of Krumhansl (1990), using a Bayesian probabilistic model. This sheds light on a number of issues: the psychological motivation for the key-profile model, aspects of musical cognition; metrical analysis, and issues of ambiguity around expectation of tonality.

A common difficulty in all areas of music is in developing a concrete and shared understanding of how music sounds. Dixon et al. (2003); Tidemann (2011) investigate this problem of perception by linking musical attributes to visual attributes. Though tempo and dynamics are relatively rudimentary elements in music, Dixon et al. (2003) highlight just how fundamental they are in the perception of a 'human' and 'musical' performance. They present a system to visualise different performance aspects (such as tempo and dynamics) which would typically be registered by listeners on a subconscious level. To achieve this a real time algorithm to determine and track multiple hypotheses of the current tempo, and update these hypotheses dynamically is presented in the paper. The paper demonstrates that these features allow the system to accurately depict relevant changes to fundamental aspects of widely varying performances of the same piece.

Similarly Tidemann (2011) seeks to understand the expressiveness of music through its perceptual links to movement. The article presents an architecture that couples musical input with body movement. The study focuses on a simulated humanoid robot that learns to play the drums like a typical drummer, taking into account both visual and auditory cues. The study employs artificial neural networks to simulate the imitation learning process.

Music AI is an expansive and diverse field of research. The proliferation of music programming languages and system designs offers numerous possibilities for the development of various approaches. This chapter has presented and contextualised the relevant work in the computer-music and Music AI domains and in doings so offers this summary of design considerations for Music AI systems:

- Representation – Due consideration needs to be given to the computational model for music. This in turn informs the data model and function that may be performed.

- Perception and cognition play an important part in developing a shared concept of how music sounds.
- A system inline with the aims of this project would be classified as a Composition AI and should utilise a structure exemplary of systems designed in this vein.

Chapter 3

A Rhythm Generation Application

This chapter will articulate the design choices made in the development of a Music AI oriented system. Primarily the system design draws on the background established in Chapter 2, with refinements made to support answering the research questions established in Chapter 1. In particular this chapter will cover the analysis undertaken to establish the bounds of the system which has been implemented, including the tools and techniques used and the system design and implementation details.

3.1 Requirements Analysis

The aims of this project as outlined in Chapter 1 can be summarised as follows:

- Investigate how a computer can play the role of a musician.
- Design a system which to the typical listener may be perceived as human in origin.

A pragmatic approach to investigate the possibilities for computers playing the role of a musician involves breaking down their activities into small and specific functions. Numerous studies take this approach, and it is perhaps the prevailing method of inquiry in this domain. For instance Miranda (2003) takes this approach by simulating a commonly accepted learning technique for musicians – imitation, which forms the basis of many pedagogical methods (Suzuki, 2013; Kohut, 1992). A number of studies have had success in producing designs

which emulated a facet of a musician's role (Spiro, 2003; Spevak et al., 2003; Pikrakis et al., 2003; De Mantaras and Arcos, 2002; Baird et al., 1993; Pachet, 2003), and this approach will be utilised in designing a system which aims to be perceived as human. Notably some of the most compelling systems make use of case based reasoning, or probabilistic algorithms such as Markov Models, grammar based rules, or a combination of these techniques (Phon-Amnuaisuk, 2003; Quick, 2014; Pachet, 2003); Pachet's Continuator (Sony CSL, 2012) being the most common exemplar for these styles of systems. Given the efficacy of these approaches the system should utilise a Markov Model style algorithm in its design.

Chapter 2 highlighted numerous possibilities for the choice of language and data model. To recapitulate what was established in Chapter 2 the following captures the key points which should be considered in the design of Music AI:

- Representation – Due consideration needs to be given to the computational model for music. This in turn informs the data model and function that may be performed.
- Perception and cognition play an important part in developing a shared concept of how music sounds.
- A system inline with the aims of this project would be classified as a Composition AI and should utilise a structure exemplary of systems designed in this vein.

These requirements are broad enough that they leave open a number of options. For instance an implementation in *Faust* (Ingalls, 2018) would be ideal if considering sound synthesis within the system. For systems considering note level semantics *Euterpea* provides a DSL within the Haskell language with a robust data model which translates well to typical music abstractions (Hudak, 2014, 2008, 1996). Haskell as a language allows clear and succinct implementations of algorithms and the Euterpea library, through use of Haskell's powerful type, data, and type-class systems, provides a clear and rich implementation of Western Classical music notation in code. Given the timeline of this project, and the scope declared in section 1.1, it was decided that this system should only focus on the absolute necessary components – the production of MIDI instructions. All other components such as the production of audio samples and sound should be considered out of scope and deferred to ancillary computer applications.

3.2 System Design

The key design goals, as established from the Aims, Requirements and Scope are:

1. The system defers some function of performing musicians to the computer – thus extending the musician's capabilities, or freeing up the composer/performer to attend to other details.

2. Produces rhythmic patterns – this project is not concerned with melodic or harmonic generation, sound synthesis or any other component of music production.

3. Produces variations over temporal scale of greater than 10 seconds – This is to ensure adequate time to express rhythmic variation and differentiate itself from purely random choice (i.e. at short enough lengths, a random rhythmic pattern may be indistinguishable from a human made pattern). Ideally this would extend to scales typical of songs (∼3min), however because of inherent repetition in typical song structures and forms, much shorter lengths are acceptable.

With this in mind, my approach will utilise probability and Markov Model algorithms common in AI and ML disciplines and consistent with previous studies examined (Pachet, 2003; Pikrakis et al., 2003), with a codified description of common music theory principles to generate structured and varied rhythms. Specifically this approach uses Markov Chains, trained on a corpus of rhythmic patterns derived from real music, to generate stylistically similar but novel patterns to satisfy the key design goals.

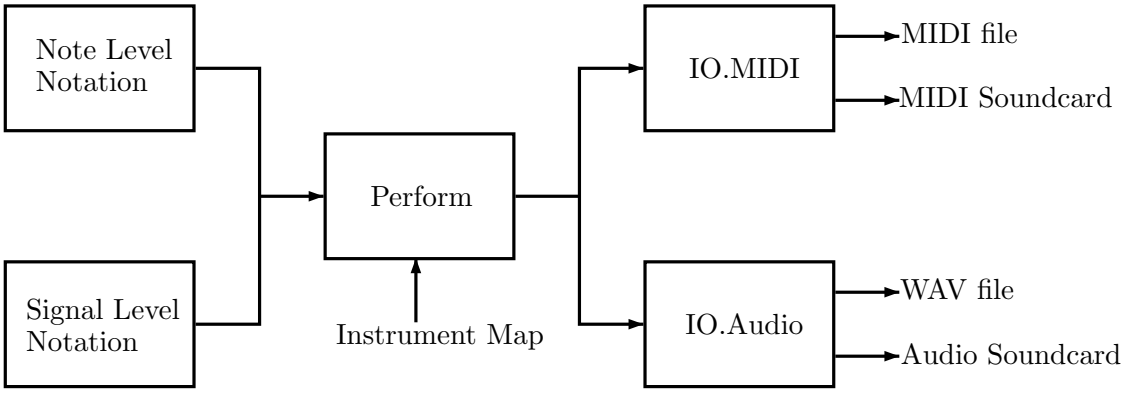

Figure 3.1: Implementation System Diagram

The implementation of this project is written in Haskell and makes use of the Euterpea library for music notation and MIDI Input/Output (I/O). Euterpea provides two main abstractions for manipulating music. The first is the Note level abstraction which is concerned with music notation, that is specifying notes, rhythms and other musical directions. The second is the Signal level abstraction which is concerned with the representation and generation of sound. These two systems can be used in an ad hoc manner to produce or receive I/O, or can be encapsulated together in an I/O function with attributes including tempo and instrumentation to produce a more detailed output. The flow described here is captured in Figure 3.1.

This project will make use of the Note level API and utilise pre-existing, real sound samples in the production of audio recordings for evaluation phase of the project, as opposed to making use of synthesised sounds from Euterpea. To transform this notation into sound, the resulting MIDI stream will be directed into a software based sampler to create the sounds, which, in turn, will be directed into a Digital Audio Workstation (DAW) – an application for recording, mixing and processing audio (Case, 2014) – for mixing and processing inline with other sound sources. Everything pertaining to how the rhythms are performed including the timing and velocity with which a particular instrument is struck and larger rhythmic structure, pattern and architecture, is determined by the system and expressed in the Note level notation.

3.3 Tools and Techniques

Haskell is the language of choice in this implementation. In producing this application the "Haskell Stack"(Haskell Stack, 2019) tool chain was used to manage software dependencies and compiler versions. Git (Chacon and Straub, n.d) has been used for version control and the source code and build documentation for this project is available online (Milford, 2018). Currently the implementation makes use of version 8.0.2 of the Glasgow Haskell Compiler (Gamari, n.d), and Euterpea 2.0.2 library code and its dependencies. In developing this application an agile methodology was used to enable a process of experimentation and exploration of techniques. This approach, along with the characteristic strengths in abstraction of Haskell, and the robust data model of music provided by Euterpea enabled the exploration of different algorithms and techniques over the same data with minimal modifications. As detailed

in Chapter 2, numerous alternative software stacks exist for developing music software however these characteristics of the Haskell-Euterpea software stack and techniques employed have proven essential to the exploratory nature of this project.

3.4 Implementation and Approach

The following section will detail key features of the code developed, in particular the implementation of Markov Models for rhythm generation as well as supplemental approaches, based on probabilistic algorithms which formed part of the experimental phase of this project.

This implementation makes use of the Note level API (Euterpea, 2018) to generate MIDI instructions. The core construct of the Note API is the polymorphic "Music" type which is used to develop a complex hierarchy of types and relationships. Furthermore it allows the use of two key operators; ":+:", to perform sequential composition – analogous with a horizontal run of notes in stave notation and ":=:", to perform parallel composition – equivalent to vertically grouped notes in stave notation.

In addition to "Music", the following types and value constructors from the Note API are used in the subsequent code listings:

Pitch A type pseudonym for specifying a pitch as a tuple of the Pitch Class (e.g F sharp) and the Octave.

AbsPitch A type pseudonym for specifying pitch as an absolute number.

Dur A number specifying the length of an event.

Prim Value constructor of Primitive values for Music types.

Note, Rest Value constructor for Primitive types which are notes, or rests.

Listing 3.1: Percussion Class

```
class Percs a where
      perc' :: PercussionSound -> a -> Music Pitch

instance Percs (Music a) where
      perc' _ (Prim (Rest d)) = rest d
```

```
instance Percs Dur where
      perc' ps d = instrument Percussion $
                   note d (pitch (fromEnum ps + 35))

-- Bind a rhythm to an instrument
dk :: Percs a => PercussionSound -> [a] -> Music Pitch
dk ps ds = line (map (perc' ps) ds)

-- sounds :: [Dur -> Music Pitch]
sounds = [ptp BassDrum1, ptp AcousticSnare]

-- percMap :: [Dur -> Music Pitch] -> [Dur] -> Music Pitch
percMap (s:ss) ds = line (map s ds) :=: percMap ss ds

-- Turn PercussionSound into Primitive a
ptp ps d = flip Note (percToPitch ps) d

-- Construct rhythmic structures
type Mphrase = [Bar]
type Bar = [Beats]
type Beats = [Beat]
type Beat = Dur
```

Listing 3.2: MIDI mapper

```
percToAbsPitch :: PercussionSound -> AbsPitch
percToAbsPitch ps = fromEnum ps + 35

percToPitch :: PercussionSound -> Pitch
percToPitch ps = pitch (fromEnum ps + 35)
```

The code in Listing 3.1 sets up the core structures which enables clean abstractions to be used in specifying inputs. In particular it extends Euterpea's "Music" type to accommodate percussion instruments. The MIDI specification enumerates pitches and sounds from 0 to 127 and as part of this percussion sounds start at 35. The percussion class abstracts over these details Listing 3.2 handles the actions of converting it to the necessary format for MIDI output.

Listing 3.3: Measures

```
{-| To encode behaviour in the performance of
    rhythmic instruments based on the following assumptions:

    - Simple, quadruple meter (4/4)
    - Beat hirarchy is as follows: 1,3,2,4
    - Phrases are roughly 2^n measures long (2,4,8,16)
-}

data FourFour = FourFour { beatOne :: Dur
```

```
                        , beatTwo :: Dur
                        , beatThree :: Dur
                        , beatFour :: Dur
                        } deriving (Show)


data QuarterBeat a = QuarterBeat Music
                   deriving (Show)

mkQuarterBeat :: Music a -> QuarterBeat
mkQuarterBeat m = if dur m == 1 % 4 then m else error "
    beatOne, is too short or too long"

bar :: FourFour -> Music a
bar b(one two three four) = one :+: two :+: three :+: four

-- An enumerated version
```

Listing 3.4: An enumerated version of Listing 3.3

```
rkick 4 = perc BassDrum1 den :+: perc BassDrum1 sn
rkick 5 = perc BassDrum1 sn :+: perc BassDrum1 den
rkick 6 = tempo (3/2) (perc BassDrum1 en :+: perc BassDrum1
    en :+: perc BassDrum1 en)

rsnare :: Int -> Music Pitch
rsnare 1 = rest qn
rsnare 2 = perc AcousticSnare qn
rsnare 3 = perc AcousticSnare en :+: perc AcousticSnare en
rsnare 4 = perc AcousticSnare den :+: perc AcousticSnare sn
rsnare 5 = perc AcousticSnare sn :+: perc AcousticSnare den
rsnare 6 = tempo (3/2) (perc AcousticSnare en :+: perc
    AcousticSnare en :+: perc AcousticSnare en)

rhihat :: Int -> Music Pitch
rhihat 1 = rest qn
rhihat 2 = perc ClosedHiHat qn
rhihat 3 = perc ClosedHiHat en :+: perc ClosedHiHat en
rhihat 4 = perc ClosedHiHat den :+: perc ClosedHiHat sn
rhihat 5 = perc ClosedHiHat sn :+: perc ClosedHiHat den
rhihat 6 = tempo (3/2) (perc ClosedHiHat en :+: perc
    ClosedHiHat en :+: perc ClosedHiHat en)

rhitom :: Int -> Music Pitch
rhitom 1 = rest qn
rhitom 2 = perc HighTom qn
rhitom 3 = perc HighTom en :+: perc HighTom en
rhitom 4 = perc HighTom den :+: perc HighTom sn
rhitom 5 = perc HighTom sn :+: perc HighTom den
```

```
rhitom 6 = tempo (3/2) (perc HighTom en :+: perc HighTom en
    :+: perc HighTom en)
```

Listing 3.3 outlines the approach to construct abstractions for better encoding of music measures. Listing 3.4 shows an enumerated version of Listing 3.3 which provides the same features.

Listing 3.5: Random Music

```
{-| Uniform Distribution
     List of percussion sounds, list of durations, seed int
udRhythm :: [PercussionSound] -> [a] -> Int -> Music Pitch
-}
udRhythm [] _ _ = rest wn
udRhythm (p:ps) rs n = dk p (map (selectRhythm rs) (r n))
    :=: udRhythm ps rs (n + 256)
  where
    r n = randomRs (0,length(rs)-1) (mkStdGen n)

selectRhythm rs i = rs !! i
```

Listing 3.6: Example usage of Random Music

```
myRhythms = [qn,en,sn,hn]
myInstruments = [BassDrum1,OpenHiHat,ClosedHiHat,
    AcousticSnare]

-- Uniform distribution
myudRhythm = udRhythm myInstruments myRhythms 42
```

Several approaches were tested including a purely random method which served as a baseline with which to evaluate other methods. Listing 3.6 demonstrates use of random, uniform distribution rhythm generator API. It takes a list of instruments, a list of durations and a seed number for the random number generator as parameters. Listing 3.5 makes use of Haskell standard library functions (randomRs, mkStdGen) to produce uniformly distributed random integers with the parameters of range (fixed to the length of the instrument list) and the seed integer. "udRhythm" uses these integers to produce a list of rhythms for each instrument in the instrument list.

Listing 3.7: Markov Model Music

```
mc' ps n i = M.run n ps 0 (mkStdGen i)
mcm' pss n i = concat (M.runMulti n pss 0 (mkStdGen i))
mcmcm' pss n i = concat $ concat (M.runMulti n pss 0 (
    mkStdGen i))
```

```
rhythmGen' :: (PercussionSound, Mphrase) -> Int -> Int  ->
    Music Pitch
rhythmGen' (s,p) n i = dk s $ mcmcm' p n i

rhythmGen :: [(PercussionSound, Mphrase)] -> Int -> Int ->
    Music Pitch
rhythmGen [] _ _ = rest wn
rhythmGen ((s,p):ds) n i = rhythmGen' (s,p) n i :=:
    rhythmGen ds n i
```

Listing 3.8: Example usage of Markov Model Music

```
bea01,bea02,bea03,bea04,bea05,bea06 :: [Beat]
bea01 = [qn]
bea02 = [en,en]
bea03 = [en,en]
bea04 = [en,en]
bea05 = [qn,qn,qn,qn]
bea06 = [ten,ten,ten]

bar01,bar02 :: Bar
bar01 = [bea01,bea03,bea06,bea01]
bar02 = [bea02,bea01,bea05,bea04]

phr01,phr02 :: Mphrase
phr01 = [bar01,bar02]

testRhythmGen = rhythmGen [(BassDrum1,phr01),(AcousticSnare,
    phr02)] 3 42
```

The implementation in Listing 3.7 was arrived at after extensive iterations. In its distilled form the structures from Listing 3.1 allow you to define Beat, Bar (measure), and Phrase (termed Mphrase) structures. The "rhythmGen" function exploits these structures to generate rhythmic patterns which can reference arbitrarily long phrase histories. Listing 3.8 provides an example of usage of the "rhythmGen" API. This first parameter takes a list of instrument (PercussionSound) and phrase (Mphrase) tuples which is used to construct a Markov Chain for each instrument. The instrument and phrases are coupled here so that musically appropriate phrases for each instrument are use in generating the output (unlike the random implementation). The second parameter "n" is passed to Data.MarkovChain module to determine size of prediction context, which in this instance is the number of values from the phrase list. The third parameter, "i" is a seed passed to "mkStdGen" to produced a random number generator. Listing 3.7 makes use of Haskell "markov-chain" package

(which provides the Data.MarkovChain module) and its API (run, runMulti) to produce a list of durations for each instrument.

This chapter has detailed and justified a design and implementation for a novel rhythm generation application. The following chapter will present the evaluation process designed for the assessment of the efficacy of this system. It will include details of auxiliary systems utilised in the creation of sound artefacts, details of the data collection methods used, and analysis techniques that will be performed on the resulting data set.

Chapter 4

Evaluation method for AI Generated Music

This chapter will detail the evaluation process designed for assessing the system and implementation established in Chapter 3. Chapter 1 highlighted that this study approaches AI from the Music perspective, and is concerned primarily with the musical qualities and perceptions of the system's output. This is an important detail as it informs the questions the evaluation should seek to answer. In particular the evaluation should determine:

- Is the system distinguishable from human derived work?
- To what degree is it perceived to be human (or not)?

Furthermore the evaluation design should acknowledge the background of the listener. There could be grounds to include the collection of demographic information regarding prior musical training within the scope of this evaluation, however this information may prove irrelevant. One should not need to be an expert listener in order to be able to quantify if something sounds human or non-human, additionally the system aims to be agreeable to the typical listener, whether a trained musician or not. In a similarly designed experiment the author found that "Music theory training showed no correlation with average scores given to each composer"(Quick, 2014, pp. 165–167), where *composer* aligns with both human and computer based sources.

There are currently no standard metrics or experimental procedures for assessing performance and qualities of algorithms in this domain, however there are a number of studies from which to draw inspiration (Quick, 2014; Colombo

et al., 2016; Gifford et al., 2017; Halkiopoulos and Boutsinas, 2012; Hutchings and McCormack, 2017), and which have been consulted in the overall design of this assessment including the survey design, data collection methods and statistical analysis.

The aim of this evaluation is to quantify the effectiveness of the system based on a person's assessment of the sonic artefacts produced. Participants will be asked to rate artefacts on a five point linear scale of 'Human' to 'Computer' produced. This evaluation method draws similarities with the well known 'Turing Test', however, by contrast it makes use of a scale for answers, rather than binary Computer/Human options. This will allow me to determine the degree of effectiveness or ineffectiveness of the system with respect to random and human options, which will serve as a baseline for the analysis.

The evaluation will take place as a survey in which participants will be presented with a recording and asked to assess this recording on a scale of one to five, one being 'Probably Human', two being 'Maybe Human', three being 'Undecided', four being 'Maybe Computer' and five being 'Probably Computer'. The user will be presented with a total of nine audio samples, which will fall into one of three categories:

1. Random – The artefacts produced are the product of random selection of duration and instrumentation within the boundaries of a predefined tempo and instrumentation.

2. RhythmGen – Artefacts produced are the product of a corpora of structured data supplied to the Markov Model implementation, again utilising the same predefined tempo and instrumentation.

3. Human – Artefacts produced are done so utilising a human input device. Rhythms will be produced at the same tempo and using the same instrumentation as in both computer generation cases.

Recordings used will be between 10 and 20 seconds in length. While similar experiments have made use of shorter recordings (Quick, 2014, p. 162) I noted that at lengths less than this eliciting perceivable variation for purely rhythmic phrases is difficult for common stylistic patterns.

In addition to this, audio samples produced will utilise the same instrument samples and acoustic processing including the addition of reverb, to

make recordings sound normal and representative of typical listening environments and experiences. Random, and RhythmGen samples will be produced by bussing MIDI data from the system into Ableton Live (Ableton Inc., n.da), a commercial DAW (See Figure 3.1). The Human made recordings will utilise a hardware human interface device – Ableton Push (Ableton Inc., n.db), designed for producing MIDI data through a touch based interface, which will also be bussed into the same DAW. From there the data will undergo a minimal processing chain which will map the MIDI data to sound samples, add reverb to the sounds, and bring the samples up to the same base volume level. This is designed to give all samples similar acoustic properties so as to avoid a listener bias towards certain recordings.

Finally, the rhythmic data supplied to RhythmGen and the recordings produced for the human component will be novel and original, so the participants should not have prior exposure to these exact audio records. However the samples will be indicative of a particular musical style and genre, which will be primarily derived from styles present in popular music idioms to avoid listener biases due to lack of training or exposure.

This evaluation design lends itself to a statistical analysis in which the distribution and variation of responses may be examined to extract information about the efficacy of the system. In doing so, this study will be able to quantify the effectiveness of its approach in pursuit of the aim to determine "how a computer may play the role of a human in undertaking various activities of a performing musician" (Milford, 2019, p. 1). The following chapter will present an analysis of the data collected and a discussion of the implications of these findings.

Chapter 5

Results and Analysis

This chapter presents the results from the evaluation procedure described in Chapter 4. Graph and table figures have been provided to aid in the discussion and analysis of the raw data and highlight its statistical features. This chapter provides a discussion on these features as well as the implications of the findings observed.

5.1 Results

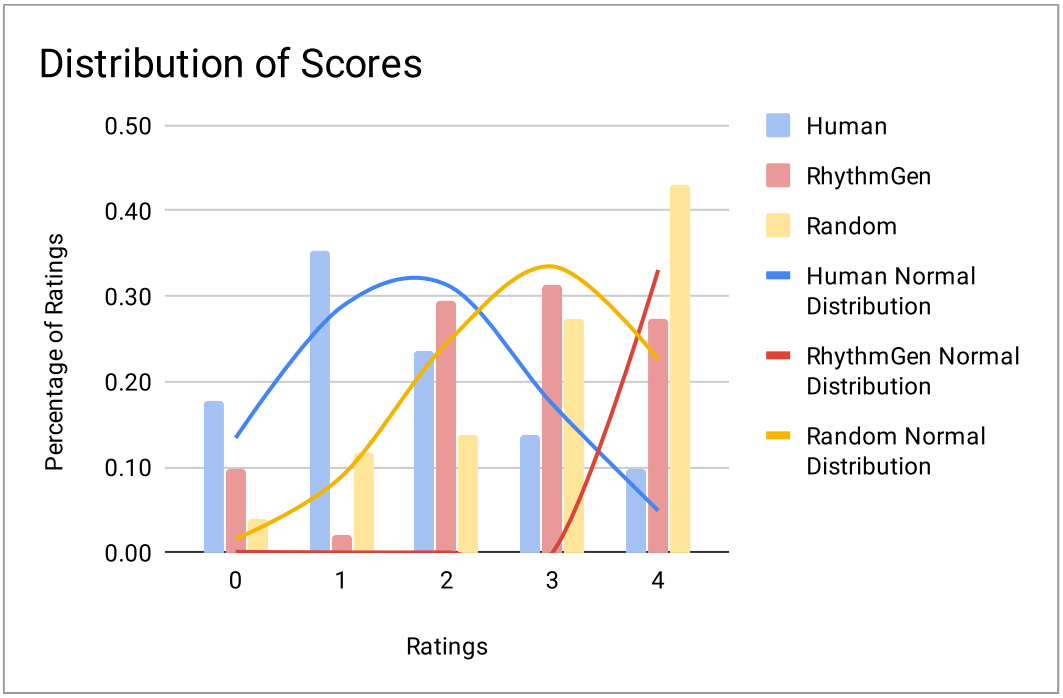

Figure 5.1: Frequency and Distribution (0 = 'Probably Human', 1 = 'Maybe Human', 2 = 'Unsure', 3 = 'Maybe Computer', 4 = 'Probably Computer')

A total of seventeen surveys were collected consisting of respondents from personal and professional social circles. A summary of the raw data is presented in Figure 5.1, which superimposes the function of normal distribution highlighting the differences present in the data set. The distributions of the raw data exposes clear differences between human and random sources. However the distribution of the RhythGen source scores was inconsistent and diffuse. Because of this the averages of participants scores were analysed as well to provide further insight into the data collected.

Mean Values		
Human	Rhythmgen	Random
1.62	2.69	2.94

Table 5.1: Mean scores of each composer

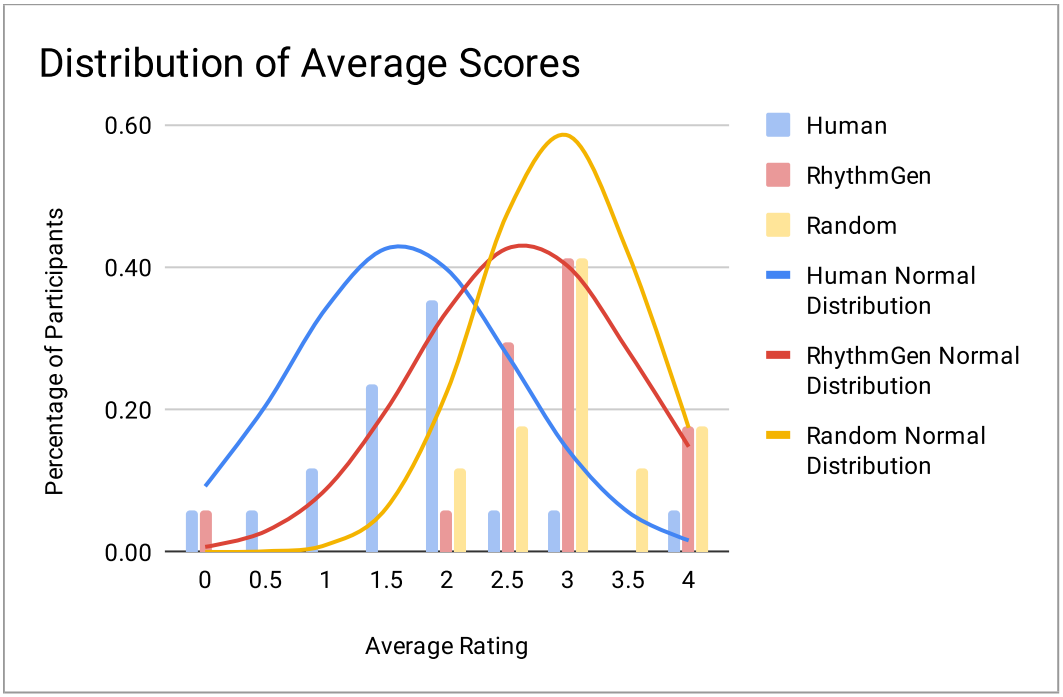

Figure 5.2: Distribution of Average Scores

Mean scores for each source are presented in Table 5.1. Human produced audio recordings yielded the lowest score as expected. The value however was closer to "Unsure" (2) than "Probably Human" (0) potentially indicating perceptual issues with the audio recordings or some other part of the assessment method. Random scored the highest, (closest to "Probably Computer") and the RhythmGen source scored better than this, positioned in the middle of

the group. Figure 5.2 presents the distribution of the averages of participants scores, expressing this relationship.

T-Test Comparisons			
	Human/Random	Human/RhythmGen	Rhythmgen/Random
Score	1.36×10^{-7}	2.78×10^{-6}	1.93×10^{-1}
Average	4.71×10^{-5}	2.01×10^{-4}	3.32×10^{-1}

Table 5.2: p values from paired double tailed t-test

Table 5.2 presents the p-values obtained from a paired, two-tailed Student T-Test on both the raw data and the averages of participants' scores. Both Human/RhythmGen and Random/Human paired tests produced statistically significant scores ($p < 0.01$), indicating distinctive populations of data and a high degree of confidence for their mean scores. RhythmGen/Random however, produced inconclusive results ($p > 0.01$) to reject the null hypothesis.

5.2 Discussion

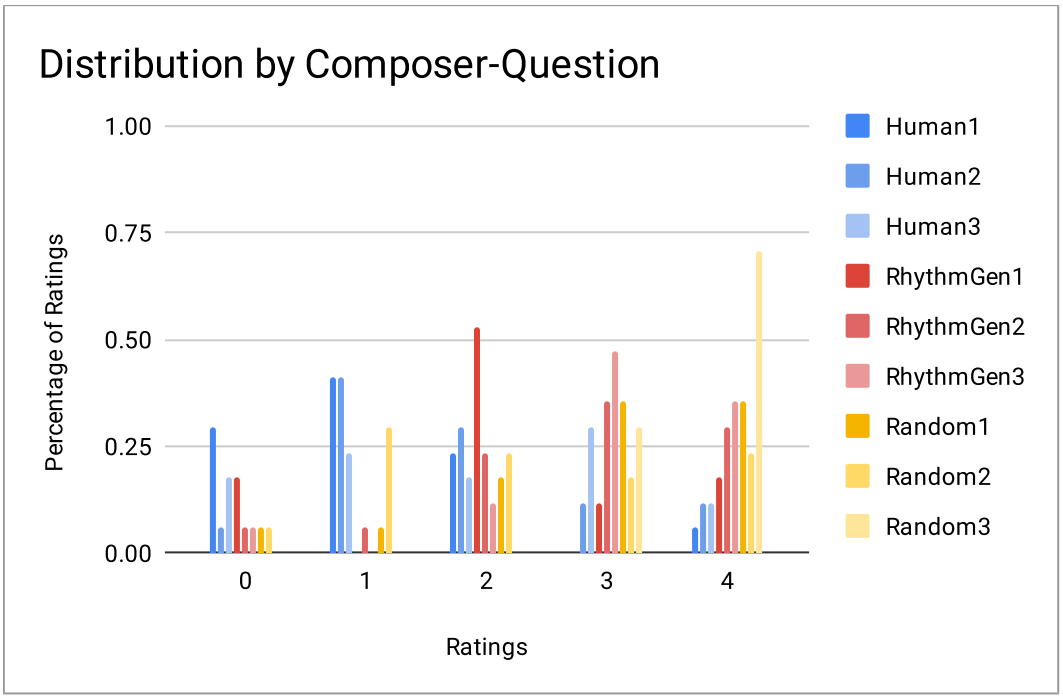

Figure 5.3: Distribution by Composer-Question (0 = 'Probably Human', 1 = 'Maybe Human', 2 = 'Unsure', 3 = 'Maybe Computer', 4 = 'Probably Computer')

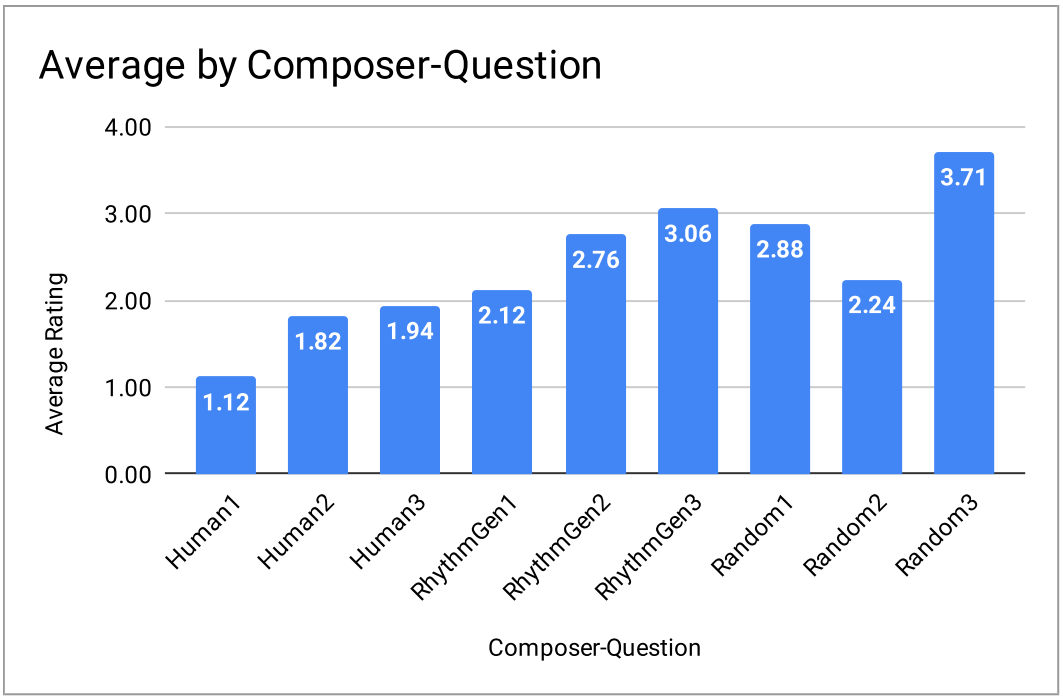

Figure 5.4: Average By Composer-Question

The assessment conducted was limited due to the time constraints of the project. A larger population would be necessary to gain a definitive answer on the outcome of the approaches presented in this paper. The preliminary data however, does provide a number of insights into the system outputs and the evaluation design. A number of unexpected trends were expressed in the data. Because of this, supplemental figures 5.3 and 5.4 are supplied as a method of visually breaking down the scoring on a per-question basis.

It was observed that certain recordings appeared to consistently score the opposite of their related artefacts. For instance *Human3* scores are distributed further right than *Human1* and *Human2*, whilst in stark contrast *Random2* is distributed further left than its similarly composed recordings. This is highlighted clearly in Figure 5.4 where *Random2* averages nearly an entire point lower than the next closest score from the same source and *Human3* averages almost an entire point higher than the lowest human score. Its difficult to comment on the RhythmGen scores however it is possible that the inconsistencies on both ends of the spectrum made the perception of the RhythmGen sources ambiguous, leading to the wide gamut of scores.

The inconsistencies observed in the data suggest the recordings presented were not truly indicative of their source. The human produced sources were not processed from their input apart from the procedures explained in Chapter 4.

Often recordings produced in this manner undergo quantisation – a process by which slight timing variations are corrected towards the global session tempo whilst correcting for stylistic considerations. In the absence of this the human produced recordings had a tendency to speed up and slow down or forgo successive beats. By contrast the computer produced recordings did not exhibit this behaviour and the difference would have been noticeable in an exclusively rhythmic audio mix. In the case of the human composed sources these cues may have been misinterpreted by the listener as being foreign or computer in origin as these attributes are usually edited out in published recordings or largely mitigated by extensive rehearsal.

Similar to Human and Random sources, the first RhythmGen recording performed well, and better then its similar recordings. A number of constraints were observed on the patterns that could be expressed to the algorithm. Primarily due to unresolved type class issues, it was not possible to express purely rhythmic rests alongside rhythmic beats. That is, it was only possible to specify when to play, and not when not to play. A number of methods were employed to mitigate this issue however, ultimately this limitation may have produced less than ideal outputs. In the context of the data collected it is difficult to tell if these subtleties were perceived as such, and if this constraint had an actual bearing on the outcome.

Some Random recordings, particularly *Random2*, were routinely perceived as human. The author notes that these recordings tended to have a likeness in style to the drum fills often found in jazz genres. As the Random algorithm had no concept of beats or measures, the patterns produced tended to become "additive rhythms" – a style in which groupings are aurally formed on a beat by beat basis and that are rarely subject to repetition. In addition to this, *Random2* by chance appeared to start with a degree of measured time likely contributing to its lower than average score. From the data collected it appears that randomly produced sequences of rhythms are not necessarily perceived as such. Ironically a smarter algorithm for producing random *sounding* rhythms may be required, which would take into consideration the full array of musical experiences the listener might be accustom to.

The results of this assessment are inconclusive, but provide valuable information for both system and experiment designs. Ideally this experiment would be performed again with at least twice the number of respondents. Utilising a number of audio recordings from each source was useful in reasoning about

participants responses to similar sources. Finally care should be taken to select recordings which are indicative of their source, or another method should be devised to inform listeners expectations. In the case of rhythmic centric systems this may prove difficult, as typical listeners may not be accustom to thinking academically about the rhythmic aspects of music they hear on a regular basis.

Chapter 6

Conclusion

This dissertation has presented a study and system design for the implementation of a selection of AI techniques to extend and compliment musical performance. Specifically the aim of this study has been to determine how a computer can play the role of a human in undertaking the activities of a performing musician. In doing so the etymology of Music AI and its ancillary domains was discussed and used to inform the framework of reasoning by which the system design in implementation was guided.

The work of Balaban (2003) was influential in guiding the choice of implementation language, and highlighting deeper considerations of the computational basis for music, which music programming languages and systems face. Hudak (2008) and Janin et al. (2013) further influenced the theory supporting the system design, in presenting a considered approach which addressed many of the challenges expressed by Balaban.

The design for a novel rhythm generation application, informed by the prior art of proponents of the field such as Quick (2014) and Pachet (2003) was presented. The implementation of this, with consideration for the factors presented at the end of Chapter 2, led to the development of a lightweight and flexible system, which supported the exploration of numerous approaches.

From the outset this dissertation emphasised the musical aspects of Music AI systems. An evaluation procedure designed to assess the efficacy of the system from a musical perspective was developed. It drew on assessment designs from similar research to create an evaluation similar to a "Turing Test", but with a five point linear scale for responses to help evaluate the system with greater accuracy.

Through this endeavour, this dissertation contributes the following knowl-

edge:

- A review and contextualisation of the computer-music and Music AI fields.
- A system design and architecture for a rhythm generation application.
- An implementation of this design.
- An evaluation design for assessing Music AI systems.
- An analysis of data from the evaluation of the implementation conducted.

An evaluation of this system was conducted however, the depth of the study was not sufficient to conclusively determine its effectiveness. The results did show some promising traits, and provided a wealth of information for the refinement of future assessment endeavours.

The outcomes of this paper alone highlight numerous avenues for further research, particularly in refining and formalising a methodology for this field of research. A standard assessment instrument would foster a refined approach to music oriented AI systems and create the opportunity for the dissemination of generalised findings and the direct comparisons of methods, system design and architectures. The system implementation in this paper utilised non-machine learning approaches because it was found that they can be highly effective; future research should seek to incorporate these techniques (expert systems, grammar based approaches etc.) with training techniques from deep learning to harness a hybrid style approach to this problem domain. Finally, this research highlighted that thought needs to be given to understanding music, its unique cognitive processes and the implications this has for the design of its computational models.

In summation, these efforts show computers have a tangible role in music performance. With diligent care for the intricacies of musical cognition, the accelerating pace of AI research will present novel and interesting opportunities for the Music Artificial Intelligence domain.

References

Sam Aaron and Jeff Rose. Overtone: Collaborative programmable music. `https://overtone.github.io`, 2018. Viewed 1 March 2019.

Ableton Inc. Live. `https://www.ableton.com/en/live/`, n.da. Viewed 1 March 2019.

Ableton Inc. Push. `https://www.ableton.com/en/push/`, n.db. Viewed 1 March 2019.

Bridget Baird, Donald Blevins, and Noel Zahler. Artificial intelligence and music: Implementing an interactive computer performer. *Computer Music Journal*, 17(2):73–79, 1993.

Mira Balaban. Structure and interpretation of music concepts: Music from a computational perspective. In *Music and Artificial Intelligence: Second Internatinoal Conference, ICMAI 2002, Edinburgh,Scotland, UK, September 12-14, 2002, Proceedings*, pages 1–3. Springer, 2003.

Gerard Berry, Xavier Fornari, and Jean-Paul Marmorat. The esterel language. `https://www-sop.inria.fr/meije/esterel/esterel-eng.html`, n.d. Viewed 1 March 2019.

Rens Bod. A general parsing model for music and language. In *Music and Artificial Intelligence: Second Internatinoal Conference, ICMAI 2002, Edinburgh,Scotland, UK, September 12-14, 2002, Proceedings*, pages 5–17. Springer, 2003.

Jean Bresson and Jean-Louis Giavitto. A reactive extension of the openmusic visual programming language. *Journal of Visual Languages & Computing*, 25(4):363–375, 2014. ISSN 1045-926X.

Jean-Pierre Briot, Gaëtan Hadjeres, and François Pachet. Deep learning techniques for music generation - A survey. *CoRR*, abs/1709.01620, 2017.

David Burnand. MIDI. "http://www.oxfordmusiconline.com/grovemusic/view/10.1093/gmo/9781561592630.001.0001/omo-9781561592630-e-0000042823", 2001. Viewed 1 March 2019.

Alex Case. Digital audio workstation. "http://www.oxfordmusiconline.com/grovemusic/view/10.1093/gmo/9781561592630.001.0001/omo-9781561592630-e-1002256346", 2014. Viewed 1 March 2019.

Scott Chacon and Ben Straub. Git. https://git-scm.com/, n.d. Viewed 1 March 2019.

Elaine Chew. *Towards a mathematical model of tonality*. PhD thesis, Massachusetts Institute of Technology, 2000.

Elaine Chew. The spiral array: An algorithm for determining key boundaries. In *Music and Artificial Intelligence: Second Internatinoal Conference, ICMAI 2002, Edinburgh,Scotland, UK, September 12-14, 2002, Proceedings*, pages 18–31. Springer, 2003.

Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

Clojure. Clojure. https://clojure.org/, n.d. Viewed 1 March 2019.

CNMAT. Open sound world. http://osw.sourceforge.net/, n.d. Viewed 1 March 2019.

Florian Colombo, Samuel P. Muscinelli, Alexander Seeholzer, Johanni Brea, and Wulfram Gerstner. Algorithmic composition of melodies with deep recurrent neural networks. *CoRR*, abs/1606.07251, 2016.

Darrell Conklin. Representation and discovery of vertical patterns in music. In *Music and Artificial Intelligence: Second Internatinoal Conference, ICMAI 2002, Edinburgh,Scotland, UK, September 12-14, 2002, Proceedings*, pages 32–42. Springer, 2003.

Csound. Csound. https://csound.com/, n.d. Viewed 1 March 2019.

Cycling '74. Max. https://cycling74.com/, n.d. Viewed 1 March 2019.

Roger Dannenberg and Ning Hu. Discovering musical structure in audio recordings. In *Music and Artificial Intelligence: Second Internatinoal Conference, ICMAI 2002, Edinburgh,Scotland, UK, September 12-14, 2002, Proceedings*, pages 43–57. Springer, 2003.

Ramon De Mantaras and Josep Arcos. AI and music: From composition to expressive performance. *AI magazine*, 23(3):43, 2002.

Simon Dixon, Werner Goebl, and Gerhard Widmer. Real time tracking and visualisation of musical expression. In *Music and Artificial Intelligence: Second Internatinoal Conference, ICMAI 2002, Edinburgh,Scotland, UK, September 12-14, 2002, Proceedings*, pages 58–68. Springer, 2003.

Euterpea. Note-level API. https://gitlab.com/brettmilford/rhythmgn.git, July 2018. Viewed 1 March 2019.

Thomas Fine. The dawn of commercial digital recording. *ARSC Journal*, 39 (1):1–17, 2008.

Ben Gamari. The Glasgow Haskell Compiler. https://www.haskell.org/ghc/, n.d. Viewed 1 March 2019.

Elizabeth Gibney. The ethics of computer science: this researcher has a controversial proposal. https://www.nature.com/articles/d41586-018-05791-w, 2018. Viewed 1 March 2019.

Toby Gifford, Shelly Knotts, Stefano Kalonaris, and Jon McCormack. Evaluating improvisational interfaces. 2017.

GRAME. *FAUST Quick Reference*. Centre National de Creation Musicale, 2017.

Constantinos Halkiopoulos and Basilis Boutsinas. Automatic interactive music improvization based on data mining. *International Journal on Artificial Intelligence Tools*, 21(4), 2012.

Haskell Stack. The Haskell Tool Stack. https://docs.haskellstack.org, March 2019. Viewed 1 March 2019.

Stephen A. Hedges. Dice music in the eighteenth century. *Music & Letters*, 59 (2):180–187, 1978.

Paul Hudak. Haskore music tutorial. In *International School on Advanced Functional Programming*, pages 38–67. Springer, 1996.

Paul Hudak. A sound and complete axiomatization of polymorphic temporal media. *Journal of Logic and Algebraic Programming,(submitted)*, 2008.

Paul Hudak. *The Haskell School of Music – From Signals to Symphonies*. (Version 2.6), January 2014.

Patrick Hutchings. Music and machine learning. `http://ai.sensilab.monash.edu/2018/08/23/Neural-Music/`, August 2018. Viewed 1 November 2018.

Patrick Hutchings and Jon McCormack. Using autonomous agents to improvise music compositions in real-time. In *International Conference on Evolutionary and Biologically Inspired Music and Art*, pages 114–127. Springer, 2017.

Soh Igarashi, Tomonobu Ozaki, and Koichi Furukawa. Respiration reflecting musical expression: Analysis of respiration during musical performance by inductive logic programming. In *Music and Artificial Intelligence: Second Internatinoal Conference, ICMAI 2002, Edinburgh,Scotland, UK, September 12-14, 2002, Proceedings*, pages 94–106. Springer, 2003.

Daniel Ingalls. Design principles behind smalltalk. `http://www.cs.virginia.edu/~evans/cs655/readings/smalltalk.html`, August 1981. Viewed 1 November 2018.

Daniel Ingalls. Faust programming language: About. `http://faust.grame.fr/about/`, August 2018. Viewed 1 November 2018.

David Janin. A robust algebraic framework for high-level music writing and programming. In *Technologies for Music Notation and Representation (TENOR), May 2016, Cambridge, United Kingdom*. HAL, 2016.

David Janin, Florent Berthaut, Myriam Desainte-Catherine, Yann Orlarey, and Sylvain Salvati. The t-calculus: towards a structured programing of (musical) time and space. In *Proceedings of the first ACM SIGPLAN workshop on Functional art, music, modeling & design*, pages 23–34. ACM, 2013.

David Kohut. *Musical Performance*. Stipes Pub Llc, 1992.

Carol L Krumhansl. *Cognitive foundations of musical pitch*, volume 17. Oxford University Press, 1990.

Otto Laske. Artificial intelligence and music: A cornerstone of cognitive musicology. In Mira Balaban, Kermal Ebcioğlu, and Otto Laske, editors, *Understanding Music with AI*, pages 2–28. MIT Press, Cambridge, MA, USA, 1992.

Victor Lazzarini. The development of computer music programming systems. *Journal of New Music Research*, 42(1):97–110, 2013.

Marc Leman. Symbolic and subsymbolic information processing in models of musical communication and cognition. *Journal of New Music Research*, 18 (1-2):141–160, 1989.

Marc Leman. *Music and schema theory: Cognitive foundations of systematic musicology*, volume 31. Springer Science & Business Media, 2012.

Peter Manning. *Electronic and computer music*. Oxford University Press, 2013.

Alan Marsden. Music, intelligence and artificiality. In *Readings in Music and Artificial Intelligence*, pages 15–28. Routledge, 2000.

James McCartney. Rethinking the computer music language: Supercollider. *Computer Music Journal*, 26(4):61–68, 2002.

MIDI Association. Midi association. `https://www.midi.org/specifications-old/item/the-midi-1-0-specification`, n.d. Viewed 1 March 2019.

Brett Milford. Rhythmgen. `https://gitlab.com/brettmilford/rhythmgn.git`, 2018. Viewed 1 March 2019.

Brett Milford. *Approaches to AI for Musical Performance*. PhD thesis, University of Southern Queensland, 2019.

Harlan D Mills. The new math of computer programming. *Communications of the ACM*, 18(1):43–48, 1975.

Marvin Minsky. A framework for representing knowledge. 1974.

Eduardo Miranda. Mimetic development of intonation. In *Music and Artificial Intelligence: Second Internatinoal Conference, ICMAI 2002, Edinburgh,Scotland, UK, September 12-14, 2002, Proceedings*, pages 107–118. Springer, 2003.

Hiroki Nishino, Naotoshi Osaka, and Ryohei Nakatsu. The microsound synthesis framework in the LC computer music programming language. *Computer Music Journal*, 2016.

François Pachet. Interacting with a musical learning system: The continuator. In *Music and Artificial Intelligence: Second Internatinoal Conference, ICMAI 2002, Edinburgh,Scotland, UK, September 12-14, 2002, Proceedings*, pages 119–132. Springer, 2003.

Somnuk Phon-Amnuaisuk. Control language for harmonisation process. In *Music and Artificial Intelligence: Second Internatinoal Conference, ICMAI 2002, Edinburgh,Scotland, UK, September 12-14, 2002, Proceedings*, pages 155–167. Springer, 2003.

Aggelos Pikrakis, Sergios Theodoridis, and Dimitris Kamarotos. Recoginition of isolated musical patterns using hidden markov models. In *Music and Artificial Intelligence: Second Internatinoal Conference, ICMAI 2002, Edinburgh,Scotland, UK, September 12-14, 2002, Proceedings*, pages 133–143. Springer, 2003.

Polychrony. Polychrony. `http://polychrony.inria.fr/`, n.d. Viewed 1 March 2019.

Dirk-Jan Povel. A model for the perception of tonal melodies. In *Music and Artificial Intelligence: Second Internatinoal Conference, ICMAI 2002, Edinburgh,Scotland, UK, September 12-14, 2002, Proceedings*, pages 144–154. Springer, 2003.

Pure Data. Pure data. `https://puredata.info/`, n.d. Viewed 1 March 2019.

Donya Quick. *Kulitta: A Framework for Automated Music Composition*. PhD thesis, Yale University, 2014.

Quil. Quil. `http://www.quil.info/`, n.d. Viewed 1 March 2019.

Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

Rafael Ramirez and Amaury Hazan. Understanding expressive music performance using genetic algorithms. In *Workshops on Applications of Evolutionary Computation*, pages 508–516. Springer, 2005.

Rafael Ramirez and Amaury Hazan. A tool for generating and explaining expressive music performances of monophonic jazz melodies. *International Journal on Artificial Intelligence Tools*, 15(4):673–691, 2006.

Curtis Roads. Research in music and artificial intelligence. *ACM Computing Surveys (CSUR)*, 17(2):163–190, 1985.

Curtis Roads. *The computer music tutorial*. MIT press, 1996.

Curtis Roads. *Microsound*. MIT press, 2004.

Pierre-Yves Rolland and Jean-Gabriel Ganascia. Musical pattern exttraction and similarity assessment. In *Readings in Music and Artificial Intelligence*, pages 115–144. Routledge, 2013.

Zsofia Ruttkay. Composing mozart variations with dice. *Teaching Statistics*, 19(1):18–19, 1997.

ShaderTone. Shadertone. `https://github.com/overtone/shadertone`, n.d. Viewed 1 March 2019.

Sony. Flow machines. `http://www.flow-machines.com/`, n.d. Viewed 1 March 2019.

Sony CSL. Musical turing test with the continuator on vpro channel. `https://youtu.be/ynPWOMzossI`, 2012. Viewed 1 March 2019.

Christian Spevak, Belinda Thom, and Karin Hothker. Evaluating melodic segmentation. In *Music and Artificial Intelligence: Second Internatinoal Conference, ICMAI 2002, Edinburgh,Scotland, UK, September 12-14, 2002, Proceedings*, pages 168–182. Springer, 2003.

Neta Spiro. Combining grammar-based and memory-based models of perception of time signature and phase. In *Music and Artificial Intelligence: Second*

Internatinoal Conference, ICMAI 2002, Edinburgh,Scotland, UK, September 12-14, 2002, Proceedings, pages 183–194. Springer, 2003.

John Supko. How I Taught My computer to write its own Music. http://nautil.us/issue/21/information/how-i-taught-my-computer-to-write-its-own-music, 2015.

Shin'ichi Suzuki. *Nurtured by Love*. Alfred Music, 2013. ISBN 0739090445.

Richard Taruskin. *Music in the Late Twentieth Century*. Oxford University Press, 2019. Viewed 17 March 2019.

David Temperley. A bayesian approach to key-finding. In *Music and Artificial Intelligence: Second Internatinoal Conference, ICMAI 2002, Edinburgh,Scotland, UK, September 12-14, 2002, Proceedings*, pages 195–206. Springer, 2003.

Tim Thompson. Programming languages used for music. http://www.nosuch.com/tjt/plum.html, 2018. Viewed 1 November 2018.

Axel Tidemann. An artificial intelligence architecture for musical expressiveness that learns by imitation. In *11th International Conference on New Interfaces for Musical Expression, NIME 2011, Oslo, Norway, May 30 - June 1, 2011*, pages 268–271, 2011.

Petri Toiviainen. Symbolic AI versus connectionism in music research. In *Readings in Music and Artificial Intelligence*, pages 47–67. Routledge, 2000.

Verimag. Lustre v6. http://www-verimag.imag.fr/Lustre-V6.html, n.d. Viewed 1 March 2019.

Gerhard Widmer. Using ai and machine learning to study expressive music performance: project survey and first report. *AI Communications*, 14(3): 149–162, 2001.

Robert Witmer. Comp. http://www.oxfordmusiconline.com/grovemusic/view/10.1093/gmo/9781561592630.001.0001/omo-9781561592630-e-2000098100, January 2003. Viewed 1 November 2018.

Appendix A

Program Code

Working application code of the system developed, including build documentation, is available at: `https://gitlab.com/brettmilford/rhythmgen.git`

Listing A.1: RhythmGn.hs

```
module RhythmGn (
  module RhythmGn.Lib,
  module RhythmGn.Random,
  module RhythmGn.SelfSimilar,
  module RhythmGn.Markov,
  )
  where
import RhythmGn.Lib
import RhythmGn.Random
import RhythmGn.SelfSimilar
import RhythmGn.Markov
```

Listing A.2: Lib.hs

```
{-# LANGUAGE TypeOperators #-}
{-# LANGUAGE TypeSynonymInstances #-}
{-# LANGUAGE FlexibleInstances #-}
module RhythmGn.Lib
where
import Euterpea

-- Support for triplets

tbn,twn,thn,tqn,ten,tsn,ttn,tsfn :: Dur
tbnr,twnr,thnr,tqnr,tenr,tsnr,ttnr,tsfnr :: Music Pitch

tbn  = 4/3;  tbnr  = rest tbn
```

```
twn   = 2/3;   twnr    = rest twn
thn   = 1/3;   thnr    = rest thn
tqn   = 1/6;   tqnr    = rest tqn
ten   = 1/12;  tenr    = rest ten
tsn   = 1/24;  tsnr    = rest tsn
ttn   = 1/48;  ttnr    = rest ttn
tsfn  = 1/96;  tsfnr   = rest tsfn

{-| better conversion from percussion sound

percToAbsPitch' :: PercussionSound -> AbsPitch
percToAbsPitch' ps = absPitch $ pitch (fromEnum ps + 35)

-}
percToAbsPitch :: PercussionSound -> AbsPitch
percToAbsPitch ps = fromEnum ps + 35

percToPitch :: PercussionSound -> Pitch
percToPitch ps = pitch (fromEnum ps + 35)


{-| Handle durations and rests with the same function
    Adhoc polymorphism
-}

class Percs a where
      perc' :: PercussionSound -> a -> Music Pitch

instance Percs (Music a) where
      perc' _ (Prim (Rest d)) = rest d

instance Percs Dur where
      perc' ps d = instrument Percussion $
                        note d (pitch (fromEnum ps + 35))

{-|
    perc' :: PercussionSound -> Music a -> Music Pitch
    perc' _ (Prim (Rest d)) = rest d
    perc' ps d = instrument Percussion $ note (dur d) (
       pitch (fromEnum ps + 35))
-}

-- Bind a rhythm to an instrument
dk :: Percs a => PercussionSound -> [a] -> Music Pitch
dk ps ds = line (map (perc' ps) ds)
```

```
-- Construct rhythmic structures
type Mphrase = [Bar]
type Bar = [Beats]
type Beats = [Beat]
type Beat = Dur

{-| TODO: WIP, make polymorphic over Rests as well
    data Beat = Dur | Music Pitch deriving (Show, Eq,
       Ord)
-}
type TimeSignature = (Int,Dur)

-- ptp :: PercussionSound -> Dur -> Music Pitch
-- ptp ps d = Prim (Note d (percToPitch ps))

-- sounds :: [Dur -> Music Pitch]
sounds = [ptp BassDrum1, ptp AcousticSnare]
-- durs :: [a -> Primitive a]
-- durs = [Note qn, Note en, Rest tn]
-- this approach will map all rests to their own line...
-- percMap :: [Dur -> Music Pitch] -> [Dur] -> Music
   Pitch
percMap (s:ss) ds = line (map s ds) :=: percMap ss ds

-- Turn PercussionSound into Primitive a
ptp ps d = flip Note (percToPitch ps) d

{-| To encode behaviour in the performance of
    rhythmic instruments based on the following
       assumptions:

    - Simple, quadruple meter (4/4)
    - Beat hirarchy is as follows: 1,3,2,4
    - Phrases are roughly 2^n measures long (2,4,8,16)
-}

data FourFour = FourFour { beatOne :: Dur
                         , beatTwo :: Dur
                         , beatThree :: Dur
                         , beatFour :: Dur
                         } deriving (Show)


data QuarterBeat a = QuarterBeat Music
```

```
                    deriving (Show)

mkQuarterBeat :: Music a -> QuarterBeat
mkQuarterBeat m = if dur m == 1 % 4 then m else error "
    beatOne, is too short or too long"

bar :: FourFour -> Music a
bar b(one two three four) = one :+: two :+: three :+:
    four

-- An enumerated version
rkick :: Int -> Music Pitch
rkick 1 = rest qn
rkick 2 = perc BassDrum1 qn
rkick 3 = perc BassDrum1 en :+: perc BassDrum1 en
rkick 4 = perc BassDrum1 den :+: perc BassDrum1 sn
rkick 5 = perc BassDrum1 sn :+: perc BassDrum1 den
rkick 6 = tempo (3/2) (perc BassDrum1 en :+: perc
    BassDrum1 en :+: perc BassDrum1 en)

rsnare :: Int -> Music Pitch
rsnare 1 = rest qn
rsnare 2 = perc AcousticSnare qn
rsnare 3 = perc AcousticSnare en :+: perc AcousticSnare
    en
rsnare 4 = perc AcousticSnare den :+: perc AcousticSnare
     sn
rsnare 5 = perc AcousticSnare sn :+: perc AcousticSnare
    den
rsnare 6 = tempo (3/2) (perc AcousticSnare en :+: perc
    AcousticSnare en :+: perc AcousticSnare en)

rhihat :: Int -> Music Pitch
rhihat 1 = rest qn
rhihat 2 = perc ClosedHiHat qn
rhihat 3 = perc ClosedHiHat en :+: perc ClosedHiHat en
rhihat 4 = perc ClosedHiHat den :+: perc ClosedHiHat sn
rhihat 5 = perc ClosedHiHat sn :+: perc ClosedHiHat den
rhihat 6 = tempo (3/2) (perc ClosedHiHat en :+: perc
    ClosedHiHat en :+: perc ClosedHiHat en)

rhitom :: Int -> Music Pitch
rhitom 1 = rest qn
rhitom 2 = perc HighTom qn
rhitom 3 = perc HighTom en :+: perc HighTom en
```

```
rhitom 4 = perc HighTom den :+: perc HighTom sn
rhitom 5 = perc HighTom sn :+: perc HighTom den
rhitom 6 = tempo (3/2) (perc HighTom en :+: perc HighTom
    en :+: perc HighTom en)
```

Listing A.3: Random.hs

```
module RhythmGn.Random where
import Euterpea
import System.Random
import System.Random.Distributions
import RhythmGn.Lib

-- selectRhythm :: [a] -> Int -> a
selectRhythm rs i = rs !! i

{-| Uniform Distribution
    List of percussion sounds, list of durations, seed
       int
udRhythm :: [PercussionSound] -> [a] -> Int -> Music
   Pitch
-}
udRhythm [] _ _ = rest wn
udRhythm (p:ps) rs n = dk p (map (selectRhythm rs) (r n)
   ) :=: udRhythm ps rs (n + 256)
  where
    r n = randomRs (0,length(rs)-1) (mkStdGen n)

-- tuplet in the form (lower bound, range), a (random)
   float
toRange :: (Int,Int) -> Float -> Int
toRange (l,r) x = round (fromIntegral(l) * x +
   fromIntegral(r))

{-| Linear distribution
ldRhythm :: [PercussionSound] -> [a] -> Int -> Music
   Pitch
-}
ldRhythm [] _ _ = rest wn
ldRhythm (p:ps) rs n = dk p (map (selectRhythm rs) l)
   :=: ldRhythm ps rs (n+256)
  where l   = map (toRange (0,length(rs)-1)) rs1
        rs1 = rands linear (mkStdGen n)

{-| Exponential distribution
edRhythm :: [PercussionSound] -> [a] -> Int -> Float ->
```

```
    Music Pitch
-}
edRhythm [] _ _ _ = rest wn
edRhythm (p:ps) rs n lam = dk p (map (selectRhythm rs) l
    )
                                :=: edRhythm ps rs (n+245)
                                    lam
  where l   = map (toRange (0,length(rs)-1)) rs1
        rs1 = rands (exponential lam) (mkStdGen n)

{-| Gaussian distribution
gdRhythm :: [PercussionSound] -> [a] -> Int -> Float ->
    Float -> Music Pitch
-}
gdRhythm [] _ _ _ _ = rest wn
gdRhythm (p:ps) rs n sig m = dk p (map (selectRhythm rs)
    l)
                                  :=: gdRhythm ps rs (n+256)
                                      sig m
  where l   = map (toRange (0,length(rs)-1)) rs1
        rs1 = rands (gaussian sig m) (mkStdGen n)

-- TODO: Impliment rhythm selection within beat unit
```

Listing A.4: SelfSimilar.hs

```
module RhythmGn.SelfSimilar where
import Euterpea

data Cluster  = Cluster SNote [Cluster]
type SNote    = (Dur,AbsPitch)

selfSim       :: [SNote] -> Cluster
selfSim pat  = Cluster (0,0) (map mkCluster pat)
    where mkCluster note =
            Cluster note (map (mkCluster . addMult note)
                pat)

addMult                    :: SNote -> SNote -> SNote
addMult (d0,p0) (d1,p1)  = (d0*d1,p0+p1)

fringe                          :: Int -> Cluster -> [SNote
    ]
fringe 0 (Cluster note cls)  = [note]
fringe n (Cluster note cls)  = concatMap (fringe (n-1))
    cls
```

```
simToMusic      :: [SNote] -> Music Pitch
simToMusic      = line . map mkNote

mkNote          :: (Dur,AbsPitch) -> Music Pitch
mkNote (d,ap)   = note d (pitch ap)

-- ss :: [SNote] -> Int -> AbsPitch -> Dur -> Music a
ss pat n tr te = transpose tr $ tempo te $ simToMusic $
    fringe n $ selfSim pat
```

Listing A.5: Markov.hs

```
{-# LANGUAGE ConstraintKinds #-}

module RhythmGn.Markov where
import Euterpea
import System.Random
import Data.MarkovChain as M
import RhythmGn.Lib

mc' ps n i = M.run n ps 0 (mkStdGen i)
mcm' pss n i = concat (M.runMulti n pss 0 (mkStdGen i))
mcmcm' pss n i = concat $ concat (M.runMulti n pss 0 (
    mkStdGen i))

rhythmGen' :: (PercussionSound, Mphrase) -> Int -> Int
    -> Music Pitch
rhythmGen' (s,p) n i = dk s $ mcmcm' p n i

rhythmGen :: [(PercussionSound, Mphrase)] -> Int -> Int
    -> Music Pitch
rhythmGen [] _ _ = rest wn
rhythmGen ((s,p):ds) n i = rhythmGen' (s,p) n i :=:
    rhythmGen ds n i


{-| TODO: Impliment [Dur] chunking based on
    timesignature
      Turn [Dur] into Phrase based on TimeSignature
      i.e. chunk a list to beat & bar length

mkPhrase :: TimeSignature -> [Dur] -> Phrase
mkPhrase (i,d) ds = mkBars i $ mkBeats d ds
```

```
mkBar :: Int -> [Beat] -> Bar
mkBar i (b:bs) = mkbar' i-1 bs b
          where
          mkbar' _ [] b      = error "not enough beats to
              make a bar"
          mkbar' 0 _ b       = b
          mkbar' i (m:ms) b = mkbar' i-1 ms m:b

mkBeat :: Dur -> [Dur] -> (Beat,[Dur])
mkBeat l (d:ds)
          | d == l     = ([d],ds)
          | d < b      = mkbeat' l-d ds [d]
          | otherwise = error errmsg
          where
          errmsg = "The rhythm crosses the beat"
          mkbeat' l (d:ds) b
                    | d == l = (d:b,ds)
                    | d < l = mkbeat' l-d ds d:b
                    | otherwise = error errmsg

mkBeats :: Int -> [Dur] ->
mkBeats l ds = mkBeat l ds
-}
```

Listing A.6: Examples.hs

```
module Examples where
import Euterpea
import RhythmGn

{-| Simple, non-generative rhythm
    The play this with 'play rhythmicPattern'
-}

kick = perc BassDrum1 en
ohhat = perc OpenHiHat en
chhat = perc ClosedHiHat sn
snare = perc AcousticSnare en

kickLine = line[kick, rest qn, kick, kick, rest dqn]
snareLine = line[rest qn, snare, rest qn, snare]
chhatLine = line[rest sn, chhat, chhat, chhat]
rhythmicPattern = kickLine :=: snareLine :=: times 4
   chhatLine

{-| Self Similar Generator
```

```
    See RhythmGn.SelfSimilar for more information
    Usage: `> play tma`
-}

smp1 :: [SNote]
smp1 = [(en,36),(qn,0),(en,36),(en,36),(dqn,0)]
smp2 = [(qn,0),(en,38),(qn,0),(en,38)]
smp3 = [(sn,0),(sn,42),(sn,42),(sn,42),(sn,0),(sn,42),(
   sn,42),(sn,42),(sn,0),(sn,42),(sn,42),(sn,42),(sn,0)
   ,(sn,42),(sn,42),(sn,42)]

tm1 = instrument Percussion (ss smp1 4 0 (1/100))
tm2 = instrument Percussion (ss smp2 4 0 (1/100))
tm3 = instrument Percussion (ss smp3 4 0 (1/100))
tma = tm1 :=: tm2 :=: tm3

{-| Random Generator with probability distributions
    See RhythmGn.Random for more info
    Usage: `play myudRhythm`
-}

-- N.B Rests in the same list cause a type error
-- e.g. myRhythms = [qnr,en,sn,hnr]
myRhythms = [qn,en,sn,hn]
myInstruments = [BassDrum1,OpenHiHat,ClosedHiHat,
   AcousticSnare]

-- Uniform distribution
myudRhythm = udRhythm myInstruments myRhythms 42

-- Linear distribution
myldRhythm = ldRhythm myInstruments myRhythms 42

-- Exponential distribution
myedRhythm = edRhythm myInstruments myRhythms 42 0.34

-- Gaussian distribution
mygdRhythm = gdRhythm myInstruments myRhythms 42 0.34 0

{-| Markov Chains
    See RhythmGn.Markov for more information
    Usage: `> play testRhythmGen`
-}

bea01,bea02,bea03,bea04,bea05,bea06 :: [Beat]
```

```
bea01 = [qn]
bea02 = [en,en]
bea03 = [en,en]
bea04 = [en,en]
bea05 = [qn,qn,qn,qn]
bea06 = [ten,ten,ten]

bar01,bar02 :: Bar
bar01 = [bea01,bea03,bea06,bea01]
bar02 = [bea02,bea01,bea05,bea04]

phr01,phr02 :: Mphrase
phr01 = [bar01,bar02]
phr02 = [bar02,bar01]

{-| Phrases must corespond to instruments,
    it doesn't make sense to select the instrument
    randomly as in the random implementation
-}
testRhythmGen = rhythmGen [(BassDrum1,phr01),(
   AcousticSnare,phr02)] 3 42
```