1 Machine Musicianship

Machine Musicianship is both an exploration of the theoretical foundations of analyzing, performing, and composing music with computers, and a tutorial in writing software to pursue those goals. The theoretical foundations are derived from the fields of music theory, computer music, music cognition, and artificial intelligence. The intended audience includes practitioners in those fields, as well as composers and interested performers.

The training of musicians begins by teaching basic musical concepts, a collection of knowledge commonly known as musicianship. These concepts underlie the musical skills of listening, performance, and composition. Computer programs designed to implement any of these skills—that is, to make sense of what is heard, perform music expressively, or compose convincing pieces—can similarly benefit from a musician's fundamental level of musicianship.

To be sure, there are many worthy computer music programs that have no basic musical knowledge at all. The usual technique is to implement thoroughly that part of musicianship required for the task at hand. Notation programs must know how many beats belong in a bar; sequencers must be able to transpose enharmonically. In this text we will explore how a more systematic foundation of musical knowledge can further extend such programs' range of use as well as improve their communication with human musicians.

Consider a simple example of this level of functionality: music sequencers can transpose enharmonically, but they cannot execute a command such as ''transpose the selected measures to the subdominant.'' The reason for this limitation is that typical programs have

no access to a description of the music in terms of relative harmonic function. Such an extension would be quite straightforward for current sequencers—and there may even be some that do it—though I have not seen any. Even better would be a sequencer that could transpose to the subdominant without the user having to inform the program as to which tonal center is current. That facility is more computationally demanding, but still well within the reach of established algorithms. Examples such as these can be generated at will and doubtless have occurred to anyone who has used music software in any depth. The point is that such programs can become more useful simply by better accommodating the practices of fundamental musicianship.

This book explores the technology of implementing musical concepts in computer programs and how resulting applications can be used to accomplish tasks ranging from the solution of simple musical problems through live performance of interactive music compositions to the design and implementation of musically responsive installations and web sites. These concepts are programmed using both C++ and Max, a graphic programming environment developed by Miller Puckette and David Zicarelli (Dobrian 1997). Some experience with one or both of these is assumed if readers wish to extend the example programs on their own. The accompanying CD-ROM includes working versions of the examples, as well as source code and a hypertext document showing how the code leads to the programs' musical functionality.

Machine Musicianship is not intended as a programming tutorial, however. The processes described in these pages constitute a computational approach to music analysis, composition, and performance that may engage practitioners in those fields whether they are programmers or not. I present the practical examples with programming information in order to help those who wish to write their own, but they can also be used as stand-alone applications by those who do not. It is my hope that interested musicians may even profit from simply reading the text without any use of a computer at all.

1.1 The Motivation for Machine Musicianship

Designing computer programs that will recognize and reason about human musical concepts enables the creation of applications for performance, education, and production that resonate with and reinforce the basic nature of human musicianship. Access to functions such as phrase boundary recognition makes possible operations that simply cannot be accomplished without such capabilities. The realization of norms for the expressive shaping of a phrase by a machine performer, for example, can only be applied once a phrase has been identified as a phrase in the first place. Further, realizing these concepts algorithmically allows us to augment human musicianship with processes and representations that only a computer could implement. A complete record of the program's ''listening experience'' is immediately available and can be used both to evaluate the algorithm's performance and to direct further analysis.

Beyond the pedagogical and practical value, I believe that there are compelling musical reasons to emulate human musicianship with computers. Readers may determine for themselves on the basis of extensive existing repertoire whether or not computer music programs have contributed to enduring compositions. Those who dismiss machine musicianship tend to argue that algorithmic composition programs (as one example) are more interesting technologically than they are musically. Another prominent source of dissatisfaction with the enterprise derives from a belief that the development of increasingly musical programs forms a real and growing threat to the livelihood of human musicians.

The need for better musicianship in music processing is relatively self-evident when contrasted with the aesthetic and ethical questions surrounding the use of automated composition and performance programs. Computers in music have made possible new kinds of creation at the same time that they have caused upheaval in the social and cultural practice of music making. Music programs are cheap, easy to use, and tireless. These attributes make it attractive to use them for many tasks that previously were performed by human

musicians. None of these properties, however, have anything to do with the nature of the music being performed. In other words, a large part of the motivation for making music with computers is that computers are less troublesome to employ than people. This situation has had a dramatic effect on the economic prospects for musicians almost as profound as the proliferation of television and sound recording equipment since the 1940s. One can condemn this trend in a hand-wringing Luddite reflex, but the situation is unlikely to change except in the direction of ever greater reliance on machines.

There are other reasons to use computers in music, however, that have everything to do with the nature of the music performed. My own interest in computer music generally, and interactive music systems in particular, stems from the new compositional domains they open up. Composers have used algorithms in the creation of music for centuries. The speed with which such algorithms can now be executed by digital computers, however, eases their use during the performance itself. Once they are part of a performance, they can change their behavior as a function of the musical context going on around them. For me, this versatility represents the essence of interaction and an intriguing expansion of the craft of composition.

An equally important motivation for me, however, is the fact that interactive systems require the participation of humans making music to work. If interactive music systems are sufficiently engaging as partners, they may encourage people to make music at whatever level they can. I believe that it is critical to the vitality and viability of music in our culture that significant numbers of people continue (or begin) to engage in active music making, rather than simply absorbing reproduced music bombarding them from loudspeakers on every side.

Tod Machover stresses a similar point:

Traditional instruments are hard to play. It takes a long time to [acquire] physical skills which aren't necessarily the essential qualities of making music. It takes years just to get good tone quality on a violin or to play in tune. If we could find a way to allow people to spend the same amount of concentration and effort on listening and

thinking and evaluating the difference between things and thinking about how to communicate musical ideas to somebody else, how to make music with somebody else, it would be a great advantage. Not only would the general level of musical creativity go up, but you'd have a much more aware, educated, sensitive, listening, and participatory public. (1999)

We are at an inflection point in the technology of our culture as the trajectories of television and computer usage cross. Already more computers than television sets are sold each year in the United States. Televisions themselves are due to become digital within a matter of years and households are already becoming wired to receive a much higher bandwidth of information than they currently get from a telephone connection. None of this comes as a revelation anymore and has been thoroughly discussed elsewhere. The interesting question for this discussion is whether people using the new computer/televisions will simply look at these devices or be moved to interact with them. I believe that if computers interact with people in a musically meaningful way, that experience will bolster and extend the musicianship already fostered by traditional forms of music education. Ultimately, the goal must be to enrich and expand human musical culture. Certainly, music will continue to be produced in any case, but without the ferment of an actively engaged audience it will lapse into yet another form of consumerism.

Philippe Manoury makes this assessment of the relationship between music and its society:

I am convinced that a certain culture is being lost. Music is increasingly playing the role of a diversion and that scares me. I don't have anything against music as a diversion, but I have the impression that our society, faced with numerous problems and no resolutions in sight, considers diversion as an antidote to those problems. . . . The more society stagnates, the more it distributes this antidote of diversion, in which music plays an important role. There is an overconsumption of the music of diversion and people don't see that music can also be the fruit of a reflection and an internal process, something they recognize more easily in literature. (Derrien 1995, 19–20 [my trans.])

Although it is tempting to believe that one's own approach to music-making will lead to a more engaged society and more fully developed art form, I make no claims of special aesthetic or social virtue inherent to interactive music. However, as computer music is so often accused of leading us to a day when machines will listen only to machines, I feel compelled to observe that many of us are motivated by a much different vision of the computer's potential connection to the community of human musicians.

1.2 Algorithmic Composition

The formalization of processes for generating music has a long and distinguished history in Western art music. From Guido d'Arezzo's chant generation method through the isorhythmic motet to serial techniques and Xenakis' ''formalized music,'' interest in processes that produce music has waxed and waned through several centuries of composition (Loy 1989). Such algorithms move the compositional act to a meta-level where the evolution of the music's character is controlled over time by the manipulation of a limited number of parameters. Computers can now execute these processes so quickly that they can be realized on stage as part of an ongoing performance (Chadabe 1989). Interactive systems change the values of compositional parameters using information from a variety of inputs, including live performance data from multiple members of an ensemble.

Because these systems derive control parameters from a real-time analysis of performance, they can generate material based on improvised input as easily as they can on interpretations of notated music. They become a kind of ligature connecting improvisation to notated composition, just as the same processes used to govern the response to notated music can be employed to generate new improvisations in performance. This possibility expands the domain of composition. By delegating some of the creative responsibility to the performers and some to a computer program, the composer pushes composition up (to a meta-level captured in the processes executed by the computer) and out (to the human performers improvising within the logic of the work).

An interesting effect of this delegation is that the composer must give very detailed instructions to the computer at the same time that she gives up such precise direction of the human improviser. The resulting music requires a new kind of performance skill as much as it enables a new kind of composition. The human player working with an interactive system must learn how to perform with it much as he would learn to play with another human. The very real differences between computer performers and human performers mean, however, that the human also acquires a new degree of freedom in invoking and directing real-time algorithms through different styles of performance. An interactive composition changes and matures as the human and computer performances increasingly intertwine.

Another possibility, of course, is that the composer will take to the stage to perform with the system herself (figure 1.1). One of the notable characteristics of the field is the resurgence of the composer/

Figure 1.1 Mari Kimura improvising

improviser, those musicians who design interactive systems and then improvise with them and/or other players in performance (e.g., Richard Teitelbaum, George Lewis, Chris Chafe, Mari Kimura, David Wessel, Ed Campion, Laetitia Sonami, and many others).

There is very seldom something new under the composition sun. Algorithmic thought is certainly not new, having been in evidence in Western music composition from the beginnings of its notation. Using processes in performance that change their behavior according to an analysis of other players's music, however, was never possible before the advent of computers and interactive music systems. Such systems therefore engender a realm of composition that was unknown only a few decades ago. I believe that this music, however, should not be described as being ''in its infancy'' or passing through an ''experimental'' phase. Doing so belittles the very real aesthetic credibility many of these works have achieved and gives composers an excuse to present works that still belong in the studio.

The musical values evinced in interactive compositions are ultimately the same as those underlying a string quartet. By transferring musical knowledge to a computer program and compositional responsibility to performers onstage, on the other hand, the composer of interactive works explores the creative potentials of the new technology at the same time that he establishes an engaging and fruitful context for the collaboration of humans and computers.

1.3 Algorithmic Analysis

There is a certain paradox at the heart of the transfer of musical knowledge to a machine. We must labor mightily to make a computer program perform the analysis required of a freshman music student. Once the work is done, however, the program can make analyses more reliably and certainly much more quickly than the freshman. The computer can deliver complete descriptions of each chord in a dictation within milliseconds of its performance, for example.

The purely quantitative difference of a very great acceleration produces a qualitative difference in the kinds of tasks a machine musi-

cian can reasonably be asked to perform. We would not set a novice musician in front of an ensemble with no idea of the piece of music to be played, its key, tempo, character, or form, and expect that apprentice player to follow what was going on very well, let alone contribute to the performance in more than a perfunctory way. Interactive systems whose knowledge of music theory does not go much beyond that of our hypothetical novice are often put into just such situations, however. Because these systems always do what they do correctly and very quickly, a little musical knowledge goes a long way.

The formalization of musical concepts is proceeding apace through research in several fields, including music theory, music cognition, and artificial intelligence. So much work has been done in recent years that it would be inconceivable to document it all in one volume. The work reviewed in this text, then, is delimited by the requirement that the algorithms discussed be able to work in real time as part of a musical performance involving human players. Even with that restriction, this text in no way forms a comprehensive overview of the field.

There is a particularly interesting convergence between the fields of music cognition and interactive composition: as music cognition research becomes increasingly concerned with processes that could account for musical competence in a real musical environment, it gives rise to algorithms that can be adapted and used by composers and improvisers in performance. Whether or not it was a concern of the great variety of developers whose algorithms are described in these pages, all of these programs also pass the minimum threshold of psychological plausibility: they are all capable of execution in real time using only the information that becomes available as it is presented in sequence.

Certainly, some aspects of musicianship do not require such demanding performance in time; analysis is usually carried out over a period of days, not milliseconds, with an open score that allows the analyst to consult the music in any desired sequence. Many excellent systems of algorithmic analysis model this situation, and a suspen-

sion of the real-time requirement often allows them to produce better results than their performance-oriented counterparts. To maintain a more manageable scope, however, some such systems will be considered only to the extent that they can be adapted to real-time use.

Figure 1.2 illustrates some of the main processes extant in the literature that can be applied to real-time analysis of musical input. Space from left to right in the figure corresponds roughly to movement from low- to high-level processes in the algorithms. The arrows approximate the flow of information between stages: pitch input is forwarded to root and key identifiers, for example, while segmentation

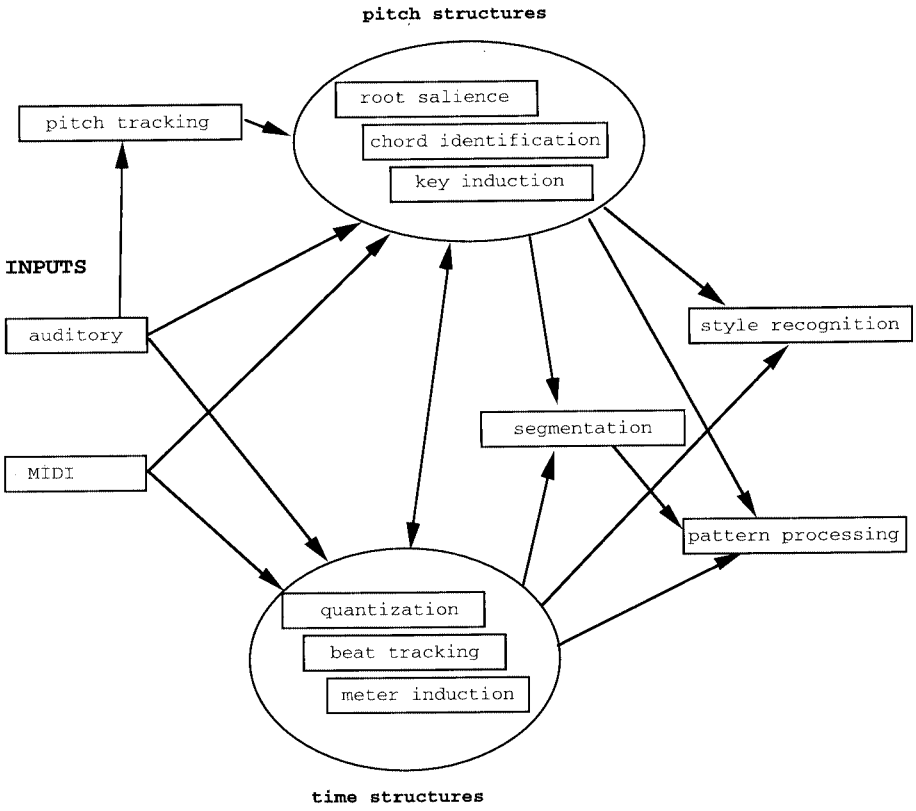

Figure 1.2 Machine musicianship processes

and style recognition rely in turn on the output of those lower-level analyses. This list is not exhaustive, but every element in it is manifested by one or more published algorithms.

My interest in this field is twofold: (1) to implement certain published processes so that they can be executed in performance; and (2) to design control structures within which these components can be combined to produce a more complete account of musical context and to make the cooperating components work better. This figure sketches only those processes related to analysis; there are similar collections that pertain to algorithmic composition and to the generation of expressive performance.

In their article ''On the Thresholds of Knowledge,'' Douglas Lenat and Edward Feigenbaum propose the Empirical Inquiry Hypothesis: ''The most profitable way to investigate AI [artificial intelligence] is to embody our hypotheses in programs, and gather data by running the programs. The surprises usually suggest revisions that start the cycle over again. Progress depends on these experiments being able to falsify our hypotheses. Falsification is the most common and yet most crucial of surprises. In particular, these programs must be capable of behavior not expected by the experimenter'' (Lenat and Feigenbaum 1992, 187).

The Empirical Inquiry Hypothesis—clearly related to Sir Karl Popper's observations on the nature of science (1992)—suggests that machine musicianship programs should be built to exhibit behaviors that observers can recognize as correct or incorrect. Many, if not most interactive music systems are written to function in an environment of new music performance and improvisation. Their output can certainly be evaluated by their makers and those familiar with the idiom. My previous book, *Interactive Music Systems* (Rowe 1993) and several sections of this one deal with just such examples. A still broader class of musicians can evaluate the performance of algorithms that process standard works, however, and in accordance with the Empirical Inquiry Hypothesis many examples in this text will treat the mainstream classical and jazz repertoires as well.

I should make clear that this is not a psychology text, though the techniques I describe could be used to implement music cognition models or experiments. Psychological theories must address the question of how the processes they propose are realized in humans. My measure of success, however, is not whether these programs match empirical data from research with human subjects, but whether they output structures that make musical sense. I will gauge their performance in those terms by comparing their output with the answers expected from students studying introductory texts in music theory. The software may produce an acceptable answer by using processes similar to those of humans, or by using others that are wildly different. All else being equal, I would prefer that the machine processes resemble the human ones. Whether or not they do is a side effect, however. Ultimately I am concerned with machine musicianship and not a strict emulation of human music cognition.

1.4 Structure of the Text

The programming examples in *Machine Musicianship* are written using two languages: C++ and Max. C++ is an object-oriented programming language that is widely available, well documented, and firmly established as one of the main vehicles for developing computer music applications. As examples are described in the text, I will develop a library of C++ objects that can be used as the basis for the reader's custom programs. This book is not an introduction to C++ programming. As examples are introduced I will summarize a few features of object orientation that are particularly valuable in developing a library for machine musicianship. Beyond that, any computer store will have a shelf full of introductory C++ books to which the reader is referred.

The fact that I will be illustrating concepts with C++ programs does not mean, however, that one must be or become a programmer to follow the text. C++ programs are a compact and complete way of notating algorithms. The algorithms themselves are the topic of interest here and will be explained in the text as they are imple-

mented in code. All of the applications described are included on the accompanying CD-ROM, but only a very small portion of the associated source code is printed in the text. Non-programmers approaching *Machine Musicianship* can then read the theory of the algorithms in question and run the associated applications to test their operation. Programmers can run the applications and modify the source, recorded in its entirety on the CD-ROM, to produce their own variations. C++ fragments in the text are concentrated at the end of each chapter so that non-programmers can skip over the code if they wish.

Max is a graphic programming language developed by Miller Puckette and David Zicarelli. There are a number of compelling reasons to include it as a development language here. First of all, Max has spawned a user community that is the most active and prolific group of interactive music designers working in the world today. There is no good reason to port Max patches to another language when there are probably more readers who know Max than know C++. In fact, I will demonstrate how C++ code is translated into a Max external to suggest how the Max community might make use of the C++ applications introduced here. Another good reason to use Max is the library of objects that has already been written for it and for MSP, a set of digital signal processing extensions. Programmers can quickly write powerful applications by building on the work of others.

Following this introduction, chapter 2 focuses on symbolic representations and algorithms directed toward the processing of pitch material. Issues such as root salience, chord identification, and key induction are addressed there. Chapter 3 continues with subsymbolic processes, notably neural networks, and expands the field of application to include rhythm. Chapter 4 moves to higher-level musical constructs including segments and patterns and discusses systems whose input consists of a digital audio stream. Chapter 5 begins to look at compositional techniques, including score following and algorithmic digital signal processing. In chapter 6 processes for the automatic application of expressive performance techniques are reviewed.

In the remaining chapters I look in detail at some distinctive interactive systems that have been used in performances or installations. The particular techniques of interactive improvisation are the nucleus of chapter 7, leading to a discussion of how such machine performers can collaborate with an ensemble of other players. Chapter 8 looks at extensions of interactive environments to include other media, most prominently graphics, in live performance situations. Chapter 9 presents several interactive installations, where the inherent ability of such systems to deal with unpredictable input contributes to responsive environments exhibiting a variety of behaviors. A presentation of research directions form the conclusion in chapter 10.

1.5 Machine Musicianship Library

The CD-ROM enclosed in this book contains a library of C++ objects that can be used to build interactive programs. The source code of the examples, also listed on the CD-ROM, provides a set of templates for users to follow in writing their own applications. The library includes the files listed in table 1.1.

Many more files are included with their associated projects: the library is a repository of base classes from which specializations are built for almost every program in the book. In object-oriented programming, a *base class* is a generalized encapsulation of data and processes concerned with a particular subset of some application area. Specializations refine those general classes into *derived classes* that address the details of some specific application. All of the analysis processes depicted in figure 1.2, for example, have examples detailed in the text and included on the CD-ROM that are made from a combination of base classes, derived classes, and custom code.

Max programs are referenced as stand-alone applications, but are not included in the *Machine Musicianship* library as the necessary objects already form the heart of Max itself. One fully developed example is ported from the C++ environment into Max as a Max external, and many of the other C++ examples could similarly produce useful

Table 1.1 Machine Musicianship Library

Clock.cp	timing routines
Event.cp	representation of a group of notes
EventBlock.cp	representation of a group of events
File.cp	file handling
Listener.cp	analysis of incoming MIDI events
ListenProps.cp	analysis processes
Mac.cp	macintosh I/O
MMerrors.cp	error reporting
Note.cp	representation of notes
OMSInPort.cp	OMS input routines
OMSOutPort.cp	OMS output routines
OMSSystem.cp	OMS communication
Scheduler.cp	scheduling facilities
Segment.cp	representation of segments
Utilities.cp	miscellaneous

Max objects. Taking programs in the other direction, from Max to C++, should be facilitated by the *Machine Musicianship* base classes.

The danger in producing a set of classes like this, particularly when it includes such entries as `Note`, `Event`, and `EventBlock`, is that it can be taken as a general representation of music. My intention is precisely the opposite—I do not believe that there is a simple and general way to represent all of the aspects of music we might want to process. The representations suggested by the *Machine Musicianship* library emerged from the particular collection of applications described in this book. Other tasks will demand other representations, or at the very least, modifications of these. Rather than a proposal for a generalized solution, the classes described here should be seen as an example of how programmers might design their own. After getting our feet wet with an initial application, in fact, I will discuss the issues of representation design more thoroughly in chapter 2.