ALGORITHMIC MUSIC: USING MATHEMATICAL MODELS IN MUSIC COMPOSITION

by GUSTAVO DIAZ-JEREZ

Submitted to

The Manhattan School of Music

in partial fulfillment of the requirements

for the degree of

Doctor of Musical Arts

and approved by

August 2000

ABSTRACT OF THE THESIS

Algorithmic Music: Using Mathematical Models in Music Composition

by GUSTAVO DIAZ-JEREZ

Thesis advisor: Nils Vigeland

The intimate connection between music and mathematics has been acknowledged throughout history. This paper is a survey of the most relevant methods employed today in algorithmic composition. It covers stochastic processes (randomness, probability functions, Markov chains); one-, two-, and three-dimensional chaotic systems; fractals; algorithms derived from noise spectra (1/f noise); number theory (Morse-Thue sequence, 3n+1/Hailstone numbers, prime numbers); cellular automata (one- and two-dimensional cellular automata); genetic algorithms; formal grammars (L-systems.) It includes a brief historical survey, an exhaustive list of available computer programs which implement most of the processes discussed, and C++ source code listings of routines valuable for algorithmic composition. Numerous musical examples are provided to demonstrate each process. These examples are included in an accompanying audio CD.

CONTENTS

ACKNOWLEDGMENTS

I wish to thank Dr. Erik Oña for his help in the preparation of this thesis; Jeremy Cavaterra for his invaluable comments; Mario Gosálvez Blanco, for his help in the preparation of the CD of the Musical Examples; and last but not least, Belinda Sánchez, for her patience and support.

PREFACE

Throughout its history, western music has been based largely on four paradigms. First, the diatonic model was dominant until the sixteenth century. Gradually, this model gave birth to a second paradigm, the tonal system, which evolved, increasing in chromaticism, until its "breakdown" at the beginning of the twentieth century. Free atonality followed, in which the rules of tonal harmony no longer hold and in which both pitch relationship and formal structure are created anew in each piece. The fourth paradigm, Shoenberg’s twelve tone system, a logical step (according to him) in the evolution of the tonal system, was an attempt to put all twelve pitches in the chromatic scale on an equal footing. Free atonality and the twelve-tone system have altered western music most dramatically on the twentieth century. It is worth noting that rhythmic evolution has evolved at a much slower pace than pitch and formal evolution. It has been only in the twentieth century that rhythmic complexity has increased greatly.

These paradigms provide *outside-of-time* structures, serving as scaffolds which the composer manipulates until the moment of their temporal inscription (that is, when a composition is created.) These outside-of-time structures are the tetrachords, Greek modes, scales, rhythmic values, dynamics, the row and magic square of twelve-tone music, harmonic rules, formal schemes such as sonata form. Composers have used them over and over again. They are, in a manner of speaking, the prime matter of music. These outside-of-time structures provide a means through which composers are able to create variation and development.

Aesthetic and value judgements, I believe, are not applicable to these outside-of-time structures, since they are mere abstract constructs from which actual works of art are engendered. They are comparable to the marble stone out of which Michelangelo carved *La pietà* or *David*. Marble (the sculptor's prime matter) is moot to any aesthetic or value judgements, while the artist's work upon it, or the product, is subject.

The development of computer technology in recent years has offered composers a way to incorporate new outside-of-time structures to the compositional process. These structures are drawn from a discipline as old as mankind: mathematics.

Mathematics and music has been intermingled since the times of Pythagoras (ca. 500 B.C.) Throughout the history of music, many composers have used mathematical models as a source for compositional creativeness. However, it has been only recently (thanks to the digital computer) that the composer can incorporate complex mathematical models in composition without having to make the tedious and monotonous calculations they require.

The goal of this thesis is to demonstrate that mathematical models provide for music outside-of-time structures which yield variation and development.

The connection between music and mathematical models is made through a process known as *mapping*. Mapping consists of establishing a correspondence between the mathematical model's data (usually numerical) and a set (or sets) of musical outside-of-time structures (such as scales, rhythmic values, dynamics, etc.) Mapping thus creates an intimate link between a new process (the mathematical model) and established musical structures. The choice of those outside-of-time musical structures for the mapping process is ultimately determined by the composer.

For the mapping process in the musical examples that demonstrate the various mathematical models discussed in this thesis, I have chosen to use only simple sets: scales, usually chromatic, and simple rhythmic values and dynamics. The main reason is clarity: simplicity allows the structures yielded by the mathematical models to be recognized much more easily. Instead of scales and simple rhythms, I could have used continuos (that is, not discrete) frequency, rhythmic and dynamic sets——thus allowing microtonality and limitless rhythmic complexity——without affecting the *intrinsic* structure yielded by the mathematical model. Furthermore, the musical examples included are not fully-fledge compositions by themselves. They serve only to demonstrate that mathematical models offer the composer a new path for variation and musical development.

All musical examples were generated by the author through the implementation of their corresponding algorithm in the C++ programming language. Because of the length of these programs, they could not be included in the body of the thesis. However, some valuable routines are provided in Chapter V.

It is presupposed that the reader has the necessary mathematical and programming proficiency to implement the algorithms by himself/herself.

CHAPTER I

INTRODUCTION

Music and mathematics have been intermingled since the dawn of history. In antiquity, music was considered a branch of mathematics. Flavius Cassiodorus (ca. 485-ca. 575,) a man who played a pivotal role in the transmission of ancient culture to the Latin Middle Ages, describes mathematics as the union of four disciplines: arithmetic, music, geometry and astronomy. "Mathematical science ... is that which considers abstract quantity ... It has these divisions: arithmetic, music, geometry, astronomy ... Music is the discipline which treats of numbers in their relation to those things which are found in sounds."[1]

As early as the sixth century B.C., the Greek philosopher and mathematician Pythagoras developed the

[1]F. Cassiodorus, "Fundamentals of Sacred and Secular Learning," in *Strunk's Source Readings in Music History*, ed. O. Strunk (New York: Norton, 1998), 144-145, footnote 7.

concept of "music of the spheres"[2] as part of his theory of the functional significance of numbers in the objective world and in music. The principles of perfect proportion that the ancients held to govern the natural universe were applied to their organization of musical pitches into fixed interval schemes known today as the *Greek Modes*.

To our knowledge, the ancients went no further in their provisions to organize the act of musical composition than to systematize pitch into fixed patterns; no evidence exists to suggest that they might have evolved procedures whereby these pitches were combined into principled forms. It was not until almost 1500 years after Pythagoras, in the Medieval period, that composers began to formulate rules by which pitch relations and combinations were governed.

In more recent times, music and mathematics——the one art, the other science, linked together by the common factor of logic——have found true conjugality in our age of the computer, by whose agency the two fields have been consciously and practically integrated. The recent advent of software programs developed for algorithmic composition

[2]A. Boethius, "Fundamentals of Music," in *Strunk's Source Readings in Music History*, ed. O. Strunk (New York: Norton, 1998), 140.

reflects man's ancient endeavor to furnish himself ever more proficient tools.

The custom of borrowing systems of organization from outside the musical realm has been firmly established in compositional practice.[3] However, as computer-assisted composers have begun to appropriate highly specific technical terms from disciplines once thought to be completely unrelated to music, the definitions of these terms have become clouded and convoluted. The term *algorithm* has been adopted from the fields of mathematics and computer science; however, in some cases its misappropriation has caused confusion in meaning. It is therefore necessary to clarify the underlying misconceptions that have accompanied the term "algorithmic composition" in order to dissolve the growing sense of ambiguity.

The word algorithm has been in use for over a millennium; still, in order to clarify its use in reference to musical composition, a cursory glance at its etymology is useful. The word comes to us from the ninth century,

---

[3]G. Loy, "Composing with computers: a Survey of some Compositional Formalisms and Music Programming Languages," in *Current Directions in Computer Music Research*, eds. M. Mathews and J. R. Pierce (Cambridge: MIT Press, 1989), 292–396.

from the work of the Arabic mathematician and writer Abu 'Abdullah Muhammad ibn Musa of Khwarizmi.[4] His most distinguished book, entitled *al-Kitab al-mukhtasar fi hisab al-jabr wa'l-muqabala* (Rules of Reintegration and Reduction,) was the basis for the standardization of Arabic numerals in European mathematics. Indeed the word algebra is a Medieval latinization of the Arabic word *al-jabr*, which means literally "the reduction." The Latin stem of the word algorithm was most likely inherited from this mathematician's homeland, *al-Khwarizm* (The Khwarizm) and termed, "algorismus." The Anglicized form, "algorism," was used to mean "the Arabic or decimal system of numerals." This archaic word appears in literature even before 1200, and little philological argument is needed to deduce that it can only be the forerunner of the modern term *algorithm*.

An algorithm in contemporary terms is characterized by the following properties:

- An algorithm consists of a finite sequence of actions.
- The sequence of actions has a unique initial action.
- Each action in the sequence has a unique successor.

[4]K. H. Parshall, "The Art of Algebra from Al-khwarizmi to Viète: a Study in the Natural Selection of Ideas," *History of Science* (June 1988): 129-164.

- The sequence terminates with either a solution to the problem, or a statement that the problem is unsolvable.

In simplified terms, an algorithm is a process that solves a problem in a step-by-step fashion through either redistribution, recursion, or branching. The solution must be found in a finite number of steps. In application to music, algorithms may be thought of as procedures that test potential compositional material for its appropriateness within a given context. Pitch, duration, intensity, and other sound and structural constituents may be chosen according to a set of questions and answers. Needless to say, the computer is the indispensable tool for the incorporation of algorithmic processes into musical composition. The use of algorithms, by reasons of both quantity and complexity, is a task perfectly suited to the computer. Best used in this capacity as a labor-saving device——to free the composer from carrying out such tedious calculations by hand——the computer is nothing more and nothing less than a utensil for the realization of abstract design constructs, which then may be applied to music-making according to the composer's own creativity and imagination. Used judiciously and according to the natural parameters of its faculties, the computer enables, for instance, the composer to experiment with the musical properties of many different algorithmic procedures and to judge their potential development into fully-fledged compositional systems. The computer in algorithmic music, like the "magic square" in dodecaphonic composition, is most usefully employed as a compositional implement rather than as a one-stop musical solution.

Within the young history of algorithmic composition with computers, three major approaches have emerged: (1) algorithms for sound synthesis; (2) algorithms for compositional structure; (3) algorithms for the correlation of sound synthesis with structure.

Sound synthesis algorithms have been used in a variety of ways, from the generation of complex waveforms[5] (building sounds,) to the evolution of timbre development over time.[6]

Some of the algorithmic programs and compositions specify score information only. Score information includes

---

[5]B. Truax, "Chaotic Non-Linear Systems and Digital Synthesis: An Exploratory Study," in *Proceedings of the 1990 International Computer Music Conference*, ed. Stephen Arnold (San Francisco: International Computer Music Association, 1990), 100-103.

[6]A. Kurepa and R. Waschka, "Using Fractals in Timbre Construction: an Exploratory Study," in *Proceedings of the 1989 International Computer Music Conference*, eds. David Butler and Thomas Wells (San Francisco: International Computer Music Association, 1989), 332-335.

pitch, duration, and dynamic material, written for acoustic and/or electronic instruments: e.g., there are instances in which a composer makes use of a computer program to generate the score while the instrumental selection has been predetermined as either an electronic orchestra or a realization for acoustic instruments.

Other algorithmic programs specify both score and electronic sound synthesis. In this instance, the program is used not only to generate the score, but also the electronic timbres to be used in performance.

One of the primary concepts involved in algorithmic composition is mapping. Mapping consists of creating direct relationships between an algorithm's output (generally numerical) and musical parameters. Mapping can be also understood as the translation of the algorithm's extra-musical material into music. How the mapping is performed is entirely the composer's choice. It must be emphasized that there are conceivably infinite ways to map an algorithm's numerical output to musical parameters; the same algorithm will yield totally different music given the application of different mappings. In addition to pitch, mapping can be applied to any musical parameter (rhythm, dynamics, etc.,) and to higher-order musical events, such as motives or even fully-fledge phrases.

There are four basic categories of algorithms that can be applied to music composition:

- Stochastic processes (probability functions, Markov chains.)
- Iterative (chaos, fractals, non-linear equations, number theory.)
- Rule-based (L-systems, formal grammars.)
- Genetic algorithms.

Each of these processes has its own unique characteristics, and the musical material they generate varies accordingly.

Stochastic processes——which involve the use of probability functions——were the first to be explored, by Iannis Xenakis in the 1950s[7], a time when avant-garde music was dominated by the heritage of Shoenberg's twelve-tone system.

The "aesthetic drive" to search for a new direction in music composition led Xenakis to formulate a new paradigm for music, which he called *stochastic music.*[8]

[7]I. Xenakis, *Formalized Music; Thought and Mathematics in Music* (New York: Pendragon Press, 1992).

[8]Ibid., 8.

The use of these new techniques required computers to be specially programmed. Decades ago, when access to a computer was relatively difficult and no relevant software existed, composers had to write the programs themselves or work in conjunction with a programmer. This presupposed a deep knowledge of the algorithm's mathematical structure as well as strong programming abilities (the more so in a time when most software of this nature was programmed in low-level languages such as Assembly or machine code.) It is not surprising that composers who used these techniques were also fairly well versed in mathematics and computers (e.g., Iannis Xenakis, an architect and mathematician; Lejaren Hiller, a chemist; et al.)

Today, fortunately, thanks to the development of high-level, object-oriented programming languages specifically designed for music composition (such as CSound[9] and MAX[10], among many others[11]) the composer can implement algorithmic procedures in a much more intuitive way. The learning curves of these programming environments, although steep,

[9]R. Boulanger, ed. *The Csound Book: Perspectives in Software Synthesis, Sound Design, Signal Processing, and Programming*, (Cambridge: The MIT press, 2000).

[10]Opcode MAX, Opcode Systems Inc.; available from http://www.opcode.com/products/max; Internet.

[11]Loy, "Composing with Computers," 318–376.

pale when compared to those of standard low- and mid-level programming languages such as Assembly or C. Furthermore, because these programming environments are specifically designed for music, the composer saves an enormous amount of time in the coding process. The code length of an algorithmic process implemented in CSound of MAX is typically ten to fifty times shorter than if it were to be programmed in C, let alone Assembly language.

In addition to mastering programming techniques and in order to implement and fully exploit the possibilities of algorithmic processes, the composer must have a deep understanding of their mathematical formulation. Today, there is a panoply of ready-made software programs that implement many algorithmic processes. Many of these programs claim that "little or no mathematical knowledge" is needed to operate them. The user just has to plug in a few numbers here and there (most of the time without even knowing what those numbers apply to) and the computer will "do the rest." This is the kind of one-stop musical solution previously denounced. Although these programs may be useful to have a rough estimate of an algorithmic process, the only way to get the most out of the process is through an understanding of its mathematical formulation and through direct experimentation with it. An algorithm can be mapped in virtually unlimited ways. Ready-made software programs offer at most a few mapping possibilities. In order to fully explore and investigate different mappings types as well as mathematical variations of the same algorithm, the composer must implement it (program it) using relevant software.

Algorithmic music uncovers a new direction in musical composition. This direction lies along a path that has lain dormant throughout the history of music. The aesthetics of mathematics as applied to music was discovered and studied by the Pythagorean more than 2,500 years ago. Today, with contemporary computer technology, we can continue investigating along that path.

CHAPTER II

A BRIEF HISTORY OF ALGORITHMIC MUSIC.

The Pythagorean

The main assertion of the Pythagorean doctrine was trifold: that all things are numbers, or that all things are furnished with numbers, or that all things behave like numbers.[1] To say that this mathematical thesis is pertinent to music is to say that the roof of a house pertains to its foundation; the one arises from the other. Indeed, the Pythagorean concept of the ubiquity of numbers evolved from the study of musical intervals which sought to obtain the "orphic catharsis,"[2] in accordance with the belief that music "cleansed the soul" as medicine cleanses the body. Pythagorism, Greek in origin, was passed on to the Byzantine, whence it was transmitted to the Arabs and to Western Europe, eventually to permeate all occidental thought.

[1]Xenakis, *Formalized Music,* 202.

[2]Ibid., 201.

Music theorists from Aristoxenos to Hucbald, Zarlino to Rameau, have returned again and again to the same assertion, expressed in more or less varied styles. The consensus seems to strike a clear unison of thought: that the arts, and conceivably all intellectual activity, are immersed in the world of numbers.[3]

No first-hand details have survived to tell us how Pythagoras discovered the numerical ratios of musical intervals. One legend tells that Pythagoras, a student of Thales——the father of Greek mathematics, astronomy and philosophy——discovered musical ratios by striking blacksmith's hammers[4] of various sizes. Their weights are generally translated in contemporary measures as 12, 9, 8, and 6 pounds. When struck in pairs, they produced the octave (12:6,) the fifth (12:8 and 9:6,) the fourth (12:9 and 8:6,) and the whole tone (9:8.) On cursory hearing the story sounds plausible; these numbers correctly represent the ratios of frequencies of those intervals. However, as soon as the acoustics of the situation are probed, the myth is revealed for what it is. On the other hand, the notion

[3]Ibid., 202.

[4]*New Harvard Dictionary of Music*, ed. Don Randel, s.v. "Pythagorean hammers," (Cambridge, MA: Harvard Univ. Press, 1986).

that Pythagoras discovered these ratios by measuring the lengths corresponding to them on the monochord (a single-stringed instrument with a moveable bridge,) is quite a creditable one and incurs no error of acoustics.

The process of continuos subtraction developed by the Greeks (*antanairesis*) was probably the principal method used for establishing several intervals, based on the Pythagorean scale.[5] This method takes the numerical proportions of two intervals and subtracts the smaller from the larger, leaving a smaller proportion, which is then subtracted from the previous proportion, and so forth. For instance, an octave minus a fifth leaves a fourth (2:1 – 3:2 = 4:3); a fifth minus a fourth leaves a whole tone (3:2 – 4:3 = 9:8); this process is repeated, creating smaller and smaller intervals: the impure minor third (32:27,) the *diesis* or *leimma* (256:243,) the *apotome* (cut-off) (2187:2048,) and the Pythagorean comma (531441:524288,) which has a frequency difference of only 23.5 cents (a tempered semitone is 100 cents.) These ratios can also be represented by arithmetic and harmonic means. The arithmetic mean between two numbers is the sum of the numbers divided by two. If *a* is the arithmetic mean

[5]Ibid., s.v. "Pythagorean scale."

between x and y then, assuming $x > y$, $x - a = a - y$. The harmonic mean between to numbers, h, has the following proportion:

$$x > y,\ (h - y)/y = (x - h)/x$$

or, solving for h:

$$h = (2xy)/(x + y)$$

The Pythagorean understood the musical application of these means, noting that the arithmetic mean of the octave (12:6,) 9, produced the fourth (12:9 = 4:3,) and the fifth (9:6 = 3:2.) Similarly, the harmonic mean of the octave, 8, produced the fifth (12:8,) and the fourth (8:6.)

Another important proportion used in antiquity was the geometrical mean. The geometrical mean, g, between two numbers x and y is:

$$x > y,\ (x / g) = (g / y)$$

or, solving for g:

$$g = \sqrt{(x*y)}$$

Aristides Quintilianus, one of the first music theorists, discovered that whenever arithmetic means (a and b) are inserted between the members of a geometric proportion ($x:g:y$,) where a is the arithmetic mean between x and g, and b is the arithmetic mean between g and y, then g is the harmonic mean between a and b.[6]

The discovery of musical ratios by Pythagoras flourished into the complex Greek musical theory, of which Aristoxenus (born ca. 365 B.C.) was the first and most important theorist.[7] Later music theorists were largely influenced by this mathematical description of music by the Greeks. These include the musical treatises of Quintilianus (*De musica*,) Ptolemy (*Harmonics*,) Boethius (*De Institutione musica*,) Gaudentius (*Harmonic Introduction*,) Hucbald (*De harmonica institutione*,) Zarlino (*Institutioni harmoniche*,) Fux (*Gradus ad Parnassum*,) Rameau (*Traité de l'harmonie*,) to name but a few.

[6]Ibid., s.v. "Arithmetic and harmonic mean."

[7]Xenakis, *Formalized Music*, 183-189.

The Middle Ages

The earliest example known of an algorithmic method applied to music composition——and the first example of mapping——dates from the 11th century by Italian composer and music theorist Guido D'Arezzo.[8] His method——developed ca. 1026——consists of creating a correspondence between the vowels of a text and a set of pitches, as follows.

The pitches of the standard two-octave vocal tesitura of the time were first laid out:

Γ A B C D E F G a b c d e f g a

where 'Γ' corresponds to the G below middle C. Next, three iterations of the vowels "a e i o u" were placed below the pitches, yielding

Γ	A	B	C	D	E	F	G	a	b	c	d	e	f	g	a
a	e	i	o	u	a	e	i	o	u	a	e	i	o	u	a

Next, the vowels from the text to be set were extracted. Lastly, using the pitch look-up table above, the composer generated the melody. Because every vowel can be mapped to

[8]Loy, "Composing with Computers," 303.

three different pitches (except for the vowel 'a', which has four correspondences,) a set of *n* vowels can generate at least 3^n different melodies (this number may be even larger since the vowel *a* has four possible mappings.) The choice among those three possible mappings per vowel (four in case of *a*) was decided by the composer. This allowed the composer to make choices so that the melodic outline would conform to the stylistic rules of the time.

Although this method is clearly algorithmic in its essence, its definiteness is somehow obscured by the exponential grow of possible melodies as the number of vowels in the mapped text increases (for a 15 vowel text, just a few words, there are at least 3^{15}——over 14 million——possible melodies!)

Isorhythmic Motets of the 14th and 15th centuries are another example were a mathematical process is intermingled with music composition. Isorhythm——the term was coined by Fridrich Ludwig in 1904[9]——means the repetition of rhythmic and melodic patterns throughout a voice part (mostly the tenor part, although in the late 14th – early 15th centuries many compositions were isorhythmic in all voices, that is,

[9]F. Ludwig, "Die 50 Beispiele Coussemakers aus der Handschrift von Montpellier," *SIMG* 5 (1903-4): 177-224.

panisorhythmic.) The rhythmic pattern was called *talea* and the melodic pattern *color*. The two patterns can be of different length, thus the successive repetition of the *talea* may occur with different pitches. If the *talea* consisted of (say) 10 durations and the *color* of 6 pitches, we would have 30 (5 X 3 X 2, since both 10 and 6 share the common factor 2) possible permutations of *talea-color* until they repeat again. In panisorhythmic motets, the number of possible permutations can grow astronomically. Naturally, of all the possible combinations, only a subset of them were allowed by the harmonic rules of the time.

Isorhythm was discussed in musical treatises of Jehan des Murs (*Libellus cantus mensurabilis*,) and Prosdocimus de Bedelmantis (*Tractatus pratice de musica mensurabili*.)[10] Composers who employed isorhythm include Vitry——who was the first to use this technique——, Ciconia, Machaut, Dunstable, Dufay, among many others. After the 15th century, isorhythm slowly died away, and we have to wait until the 20th century to find similar practices.

Another procedure developed during the Middle Ages which involves the mapping of non-musical material to music is *soggetto cavato*. Soggetto cavato——the term was coined

[10]*New Harvard Dictionary of Music*, s.v. "Isorhythm."

by Zarlino[11]——consists of making a one-to-one correspondence between the individual letters of words in text to music. For instance, Josquin des Prez' mass dedicated to *Hercules Dux Ferrarie* derives its theme from this dedication by employing the six solmization syllables, or the names of the pitches of the scale (*ut*, *re*, *mi*, *fa*, *sol*, *la*,) as follows: *re*, *ut*, *re*, *ut*, *re*, *fa*, *mi*, *re*, corresponding to the vowels e-u-e-u-e-a-i-e, yielding the pitches d-c-d-c-d-f-e-d. Soggetto cavato has been used by composers throughout the history of music. Well known examples include Bach's use of his own name (B A C H: B-flat - A - C - B natural) in the final three-subject fugue in *The Art of the Fugue*, Schumman's *ABEGG Variations* for piano (A - B-flat - E - G - G,) and many others——the letter B in German corresponds to B-flat, the H to B natural. This procedure could be applied to all letters of the alphabet, not only to the ones that suggest pitches. For example, Ravel in his *Menuet sur le nome de Haydn* maps "Haydn" as B – A – D – D – G, mapping the 'y' to D and the 'n' to G, as follows:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
       ABCDEFGABCDEFGABCDE
```

[11]G. Zarlino, *Institutioni harmoniche*, trans. G. Tomlinson, (Ridgewood, NJ: Gregg Press, 1966).

The Classical Period

Another example of a mathematical process applied to music before the computer age is found in Mozart's *Musikalisches Würfelspiel*, K 516f (a musical dice game.) The idea was to cut and paste pre-written measures of music together to create a minuet and trio. This technique appears to have been pioneered by Kirnberger in 1757.[12] In the late Baroque through Classicism, it was common practice for composers——even those of great prestige——to use compilations of progressions, cadences, motives, etc., in their works, sometimes as a source of inspiration, sometimes as the actual music. This technique became known as *ars inveniendi.*[13]

Mozart used a table of 176 possible measures for a 16-measure Minuet (11 2-dice distinct combinations times 16 measures) and 96 for the trio section (6 X 16.) The structure of the composition was determined by chance: two dice were rolled to determine the measures for the minuet, one for those of the trio. The result of a dice roll was checked against the table of measures, in order to determine which one to play.

[12]Loy, "Composing with Computers," 303.

[13]Ibid.

The instructions for the *Musikalisches Würfelspiel* appear on a sheet of sketches in the Bibliothèque Nationale de Paris, dating from 1787.[14] The compositional procedure is as follows:

1. Throw both dice for measure 1 of the minuet;
2. Match the number shown by the dice (which will necessarily be a number between 2 and 12) to the corresponding number on the table of the 176 pre-composed measures;
3. Repeat until all 16 measures have been determined.

The trio is composed analogously, except using only one die. In theory, there are $11^{16} * 6^{16} \approx 1.3 * 10^{29}$ possible compositions. Many of them will be related, since there is a predetermined amount of musical material, but none of them will be exactly the same. This number is so large (13 followed by 28 zeroes) that if so many grains of sand were to be arranged in a single file (assuming each to be one millimeter in length,) they would span a distance of

[14]E. Smith, liner notes to W. A. Mozart, *A musical dice game* (Phillips Complete Mozart Edition—Rarities and Surprises 422 545-2, 1991).

roughly 13 billion light years, which is approximately 130,000 times the diameter of the Milky Way galaxy!

This process involving chance is an important antecessor of the aleatoric procedures used by John Cage two centuries later[15], with a key difference: in Mozart's game, the sequence of measures in the "generated" minuet and trio had to conform to the stylistic rules of the time. This was possible because each measure in the look-up table was selected from a set of possible successor measures, arranged in such a way so that all sequences would agree to established harmonic and stylistic canons.

The Golden Mean

Another mathematical model that has been used in the arts throughout history is the *Golden Mean*.[16] The golden mean——whose mathematical symbol is the Greek letter phi (ϕ)——is defined as the point that divides any segment in two sections so that the proportion between the larger section and the smaller is equivalent to the proportion between the whole segment and the larger section. Mathematically, if

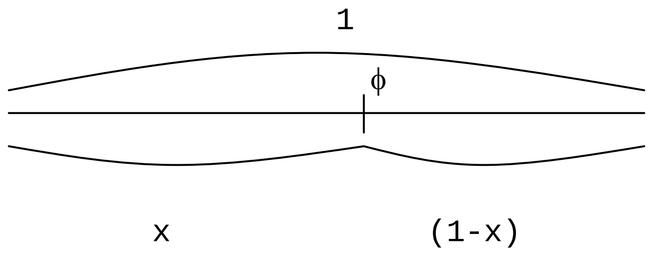
[15]Loy, "Composing with Computers," 304.

[16]J. T. Anderson and C. S. Ogilvy, *Excursions in Number Theory* (New York: Dover, 1988), 139-140.

we take a segment of length 1 (for convenience,) the Golden Mean is a point x on the segment that satisfies the following equation:

$$1/x = x/(1-x) \quad \text{or} \quad x^2 + x - 1 = 0$$

Put graphically,



The Golden Mean is the positive solution of the quadratic equation $x^2 + x - 1 = 0$ (the other solution is negative and has no physical relevance here.) Phi is exactly equal to $(\sqrt{5} - 1)/2$, or approximately 0.6180339887499. This number appear everywhere in nature: in the branching of trees, sunflowers, seashells, etc.[17] This proportion is known since the Greeks, who used it as a structural model in their architecture. The Parthenon in Athens, for instance, is full of golden mean proportions.

[17]C. C. Clawson, *Mathematical Mysteries: the Beauty and Magic of Numbers* (New York: Plenum Press, 1996), 126.

The Golden Mean is intimately related to the Fibonacci series, discovered by Leonardo of Pisa in the 13th century[18]:

1 1 2 3 5 8 13 21 34 55 89 144 ...

Starting with two ones, each new term of the sequence is generated by adding together the last two. Put mathematically,

$$F_n = F_{n-1} + F_{n-2}$$

The Fibonacci sequence and the Golden Mean are related as follows: if we divide the nth term of the sequence by the (n+1)th term, we get:

```
F(1)/F(2) = 1/1   = 1
F(2)/F(3) = 1/2   = 0.5
F(3)/F(4) = 2/3   = 0.666666666
F(4)/F(5) = 3/5   = 0.6
F(5)/F(6) = 5/8   = 0.625
F(6)/F(7) = 8/13  = 0.6153846153846
F(7)/F(8) = 13/21 = 0.6190476190476
F(8)/F(9) = 21/34 = 0.6176470588235
                 .
                 .
```

As the terms become larger and larger these ratios converge to a finite value: ϕ, the Golden Mean:

[18]Ibid., 123.

$$\lim_{n\to\infty} F(n)/F(n+1) = \phi$$

This proportion has fascinated many composers throughout history, who have used it mainly as a structural device. If we consider the whole length of a piece, its Golden Mean is approximately at 61.8% of its total duration. Composers usually reserve the moment at which the Golden Mean happens for something special, such as the climax of the piece or a dramatic moment. One of the earliest musical examples of the use of this is Thomas Tallis' 40-voice motet *Spem in alium*: at the Golden Mean there is a bar of complete silence, followed by the entrance of all 40 voices together. In the twentieth century, Claude Debussy used this proportion in many of his pieces[19], such as *Reflets dans l'eau*, where the climax occurs at measure 58 of a total of 94 ($58/94 \approx \phi$.) Béla Bartók also made extensive use of the Golden Mean in his music. Examples include the first movement of his *Sonata for two Pianos and Percussion*, the first movement of *Music for Strings, Percussion and Celesta*——the climax occurs at

[19]R. Howat, *Debussy in Proportion: a Musical Analysis* (New York: Cambridge Univ. Press, 1983).

the Golden Mean, at bar 55 of a total of 89——, among others.

The Twentieth Century

The advances in mathematics and science in general since the mid-nineteenth century have allowed composers to incorporate ideas and procedures into their music that were inconceivable in previous centuries. In the present day, the idea of music as an art-science sparked by the ancient Greeks has been rekindled.

The Schillinger System of Musical Composition

Through the 1920s and 1930s Joseph Schillinger, an Ukrainian composer and music theorist, developed a *System of Musical Composition*[20] allegedly based on scientific principles. In *The Mathematical Basis of the Arts*[21], his magnum opus, he advocates for a formalized theory of aesthetic creation. His efforts, however, were futile. Human creativity has eluded so far any type of formalization. This seems to be an intractable problem and

---

[20]J. Schillinger, *The Schillinger System of Musical Composition*, (New York: Da Capo Press, 1978).

[21]J. Schillinger, *The Mathematical Basis of the Arts*, (New York: The Philosophical Library, 1948).

some authors have argued against a computable model of creativity on the basis that the mental processes that govern creativity and intuition are simply not computable.[22]

Schillinger's *System* covers all fundamental aspects of musical composition, such as counterpoint, harmony, rhythm, etc. It is essentially geometrical in its basis, especially the concept of "phase relationships." This concept encompasses virtually every component of the *System*. These phase relationships are in essence simple periodic motions. His methodology projects these phase relationships into the areas of rhythm and structural proportion as well as into the much less obvious ones of pitch structures (scales and chords,) counterpoint, harmonic progression, etc. The groundwork underlying much of Schillinger's *System* is a process by which interference with simple regular rhythmic patterns produces more complex and irregular rhythmic patterns. For instance, the following rhythmic pattern:

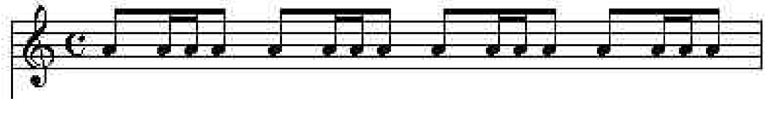


Figure 1

[22]R. Penrose, *The* Emperor's *New Mind: Concerning Computers, Minds, and the Laws of Physics* (New York: Oxford Univ. Press, 1989).

can be represented as the union of the following two simpler patterns:

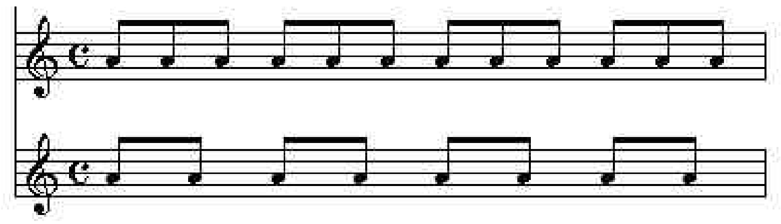


Figure 2

This process of obtaining complex rhythms from logical operations worked upon simpler ones resembles very much the way complex sounds are synthesized: by the addition of many simpler, pure sine waves.

The work of Schillinger has been greatly criticized by some authors as having absolutely no scientific or mathematical foundation whatsoever.[23] However, some his ideas have shed light on new directions in the compositional process. He presaged many developments of algorithmic composition that were not taken up in full until many years later. Schillinger's *System* contains frequent references to music as "natural dynamic." This

[23]L. Fichet, *Les theories scientifiques de la musique* (Paris: Librairie philosophique J. Vrin, 1996).

attitude is, of course, nothing new: it dates back to Zarlino in the early Renaissance, which defended the idea of music as an imitation of Nature.[24] Schillinger's use of the Fibonacci series, with reference to its use in describing growth patterns of plants, seashells, etc., is one of many examples scattered throughout his *System* of composition. This is clearly related to the recent developments in fractal geometry, particularly the studies of chaotic attractors and non-linear dynamics as models of natural phenomena.

Edgard Varèse

Another composer who played an important role in the consolidation of the view of music as an art-science was Edgard Varèse. He championed the ancient idea that music's place is in the company of mathematics and geometry. But he was fully aware that music is not just science. It participates in both science and art. He insisted that the basis of creative work was experimentation, and was conscious of the immense creative possibilities that new emerging technologies offered to music: " ... liberation

[24]Loy, "Composing with Computers," 308.

from the arbitrary, paralyzing tempered system; the possibility of obtaining any number of cycles or, if still desired, subdivisions of the octave, and consequently the formation of any desired scale; unsuspected range in low and high registers; new harmonic splendors obtainable from the use of sub-harmonic combinations now impossible..."[25]

Varèse, the true father of electronic composition, had a tremendous influence on generations of composers after him, opening a new path for music composition. Stockhausen, Ligeti, Boulez, Xenakis, and many others could not have flowered as fully without Varèse as their predecessor. His background in mathematics and science——he prepared for a career as an engineer——undoubtedly helped him in his search for a new aesthetic of sound.

Although Varèse was more interested in the new world of sound opened by electronic music than with the application of specific mathematical procedures to composition——which may be realized by electronic means or not——his influence and ideas fertilized the ground for composers wishing to experiment with these new techniques. As he puts it, "the world is changing, and we change with it. The more we

[25]E. Varèse, "The Liberation of Sound," in *Source Readings in Music History*, ed. O. Strunk (New York: Norton, 1998), 1343.

allow our minds the romantic luxury of treasuring the past in memory, the less able we become to face the future and to determine the new values which can be created in it."[26]

Iannis Xenakis

The first composer who adopted a pure mathematical approach to music composition——not just as a tool, but as a philosophy of composition——was Iannis Xenakis. Xenakis criticized the serial approach. Himself one of the pioneers of modern computer-assisted composition, he thought post-serial music posed a fundamental aesthetic problem. He argued that "the completely deterministic complexity of the operations of composition and of the works themselves produced an auditory and ideological nonsense."[27] Furthermore, "linear polyphony destroys itself by its very complexity; what one hears is in reality nothing but a mass of notes in various registers. The enormous complexity prevents the audience from following the intertwining of the lines and has as its macroscopic

[26]Ibid., 1340.

[27]Xenakis, *Formalized Music*, 8.

effect an irrational and fortuitous dispersion of sounds over the whole extent of the sonic spectrum."[28]

Xenakis recognized a fundamental contradiction between linear polyphony and what one actually hears. He advocated that this contradiction would vanish if the independence of sounds were absolute, i.e., when linear combinations of sounds and their polyphonic superposition no longer function as such. "What counts will be the statistical mean of isolated states and of transformations of sonic components at a given moment. The macroscopic effect can then be controlled by the mean of the movements of elements which we select. The result is the introduction of the notion of probability, which implies, in this particular case, combinatory calculus."[29] Xenakis called this new paradigm of music "stochastic music." Stochastic music is concerned with masses of sounds, rather than with the linear succession of pitches that conform those masses. These "sonic events" may involve thousands of individual sounds. The linear successions of these sounds is "non-deterministic," or rather, they do not follow any discernible patterns and are distributed randomly.

[28]Ibid.

[29]Ibid.

However, the overall shape they create is well-defined and directed. The way gases behave illustrates this point very well: a gas is composed of a large number of gas molecules; the position and speed of individual molecules within the gas is unpredictable and random, yet the overall shape of the gas is not: it has a definite pressure and temperature. The individual pitches in these sound masses resemble individual molecules in a gas: taken individually, they appear randomly distributed, but taken as a whole they create a well-defined entity. Composers can control how these sound-masses behave, by manipulating their density, rate of change, etc. By reducing the densities of these sound-masses, composers can achieve results that resemble linear polyphony. This new conception of music is therefore a superset, or extension, of traditional polyphonic models. Probability theory is indeed needed to control the evolution of these sound-masses. This includes probability distribution functions such as the continuous probability function, Poison's law, Markovian analysis, etc. With these mathematical tools, the composer can manipulate and control how these sound masses evolve and are transformed. This is accomplished macroscopically, upon the whole sound-event. The individual pitches that form those sound-events are arranged in a non-deterministic (random) manner; in fact, random number generators are used to generate the note-level of the sound-event.

At the time Xenakis developed stochastic music (the 1950s,) a composer wishing to incorporate these techniques in his music required no small degree of mathematical literacy, since the bulk of calculations were done by hand. Computers were a luxury, and even if access to one was possible, it had to be specially programmed for the task. Such a composer had to be a "hybrid"[30] between musician and mathematician, with almost equal proficiency in both fields. Today, however, thanks to the development of computers, composers can acquire readily made programs that perform all the necessary calculations, saving them from these tedious tasks. Still, a profound knowledge of the abstract mathematical model is necessary for to completely exploit its possibilities, even while the composer is spared the "dirty work."

Apart from stochastic techniques, Xenakis employed other mathematical models for composition, such as set theory[31] (*Herma*)——not to be confused with Allen Forte's theory——, which deals with performing logical operations on

[30]Ibid., vii.

[31]Ibid., 155-177.

sets of pitches; game theory[32] (*Duel*, *Stratégie*); group theory[33] (*Nomos Alpha*); etc.

Stochastic music has been criticized by some authors for the way it may limit the stylistic objectives of the composer.[34] The composer has control over the way probabilistic distributions are applied in order to shape the work, but not over the "fine detail" of the sound-masses. In the 1960s, Max Mathews together with other researchers, investigated ways to overcome this restriction by using deterministic algorithms and pitch quantization.[35] In one of their experiments, they created counterpointal textures in which intervals between voices were quantized to the nearest 3rd, 4th, 5th, or 6th. The number of voices and the method for pitch generation could be changed at will. Different strategies were explored, including different levels of pitch quantization, from no quantization at all (absolute frequency) to discrete quantization into tempered and diatonic scales. Their methods allowed the composer to have control over the note

[32]Ibid., 110-130.

[33]Ibid., 219-241.

[34]Loy, "Composing with Computers," 310.

[35]Ibid., 311.

level of the work, while still retaining the philosophy of stochastic procedures.

Lejaren Hiller

One of the first composers who adopted a systematic, algorithmic approach for composing——and allegedly the first person to compose a piece with the help of a computer——was Lejaren Hiller. His strong scientific and mathematical background (he earned a Ph.D. in chemistry) provided him with the necessary knowledge to realize his ideas. Hiller's musical thinking was greatly influenced by information theory.[36] Information theory is concerned with assimilating, quantifying, and optimizing the efficiency of information transfer. It is central not only to modern communication technology, but also to the understanding of language, and the transmission of information by other processes. In applying information theory to music, Hiller conjectured "fluxes" in the sum of information conveyed by a piece of music to the listener to be the essential dramatic nature of music. This premise greatly influenced

[36]L. Hiller and L. Isaacson, eds., *Experimental Music; Composing with an Electronic Computer* (New York: McGraw-Hill, 1959).

the way in which Hiller composed. In 1957, he wrote *The Illiac Suite*, also known as *The Illiac String Quartet* or, simply, as *String Quartet No. 4*. This piece, generated on a Illiac computer at the University of Illinois with the help of Leonard Isaacson and Robert Baker is allegedly the first example of a composition created with a computer. In 1963, together with Robert A. Baker, Hiller created the computer composition language program MUSICOMP, which was used to create his *Computer Cantata*. Several of the commands available in this program are designed with ideas of these "information fluxes" in mind. This program was also used to create other pieces such as *Algorithms I*, *Algorithms II*, and *Algorithms III*.

Hiller advantaged himself of a great deal of other tools besides computers. The total output of his work shows a highly eclectic composer, partaking of all the musical currents of his day, from those as popular as jazz to those as erudite and abstruse as serialism or indeterminacy.

Contemporary Techniques

With the emergence chaos theory and the development of other scientific disciplines such as cellular automata, fractal geometry, etc.——once again, thanks to the tremendous increase in computational power——new links between mathematics and music have been established.

Chaos Theory

Chaos Theory[37], pioneered by French mathematician Henri Poincaré[38] at the beginning of the 20th century and flourishing in the 1980s, describes the unpredictable behavior of systems when influenced by a common condition or set of conditions.

Until recently, scientists believed that random influences made systems behave unpredictably. They believed that if they eliminated random influences, they could predict behavior. They now know that many systems can exhibit long-term unpredictability without random influences. Such systems are chaotic. Chaotic systems are unpredictable because they are sensitive to their initial

[37]J. Gleick, *Chaos* (New York: Penguin Books, 1987); E. Ott, *Chaos in Dynamical Systems* (New York: Cambridge University Press, 1993).

[38]H. Poincaré, *Science and Hypothesis* (New York: Dover, 1952).

conditions, such as position and velocity. Two identical chaotic systems, set in motion with slightly different starting conditions, can quickly produce very different motions. What is so thought-provoking about chaos theory is that unstable aperiodic behavior can be found in mathematically simple systems. These systems display behaviors so complex and unpredictable so as to merit the description "random." Because of the complexity and number of calculations involved in studying chaotic systems, it has been only recently, thanks to the increase in computational power in the past twenty years, when chaos theory has truly flourished.

The mathematical equations that model chaotic behavior are, paradoxically, exceedingly simple. They are called "non-linear" equations.[39] Linear equations——those whose solution lays on a straight line, having the form $f(x) = ax + b$, where a and b are constants——exhibit simple, predictable behavior: the input is proportional to the output, and they have unique solutions. Non-linear equations, on the other hand, have infinite solutions. The key ingredient of non-linear equations is *iteration*: the

[39]J. Briggs and F. D. Peat, *Turbulent Mirror* (New York: Harper and Row, 1989), 23-24.

solutions are fed back into the equation's variables recursively. Iteration allows these relatively simple mathematical systems to model the chaotic behavior of many natural processes. The fashion in which mathematical systems are utilized in musical composition is through the mapping of their equations' numerical output to musical parameters.

Chaos theory has fascinated composers, and many——such as Charles Wuorinen, Gary Lee Nelson, David Clark Little[40], to name but a few——have adopted it as a tool for composition.

Cellular Automata

Another discipline that has recently been proven to give very fruitful results in music composition is *Cellular Automata*.[41] Cellular automata were originally introduced in the mid-1960s by John von Neumann as a model for computer–simulated biological self-reproduction.[42] This model

[40]D. C. Little, "Composing With Chaos; Applications of a New Science for Music," *Interface* 22(1) (1993): 23-51.

[41]S. Wolfram, ed., *Theory and Applications of Cellular Automata* (Singapore: World Scientific, 1986).

[42]J. von Neumann, *Theory of Self-Reproducing Automata* (Champain, IL: University of Illinois Press, 1966).

consisted of a two-dimensional grid of cells, in which each cell was allowed a number of different states. Cells would change their states on the grid according to a predefined set of rules. This set of rules takes into account the current state of the cells' immediate neighbors. Starting with an initial configuration of cells——generally called generation 0——in which each cell may have any of the allowed states, the set of rules is applied to all cells on the grid, thus producing a new configuration of cells (generation 1) to which the set of rules is applied again, and so forth. Depending on the number of allowed cell states and the rule set, different cellular automata can be generated. Some of them exhibit great richness, while others die out quickly. Many cellular automata manifest pattern propagation, in which a determined configuration of cells can move undisturbed throughout the matrix. Certain types of cellular automata also show "self-reproducing" capabilities: under certain conditions, some cell configurations are able to regenerate themselves generation after generation.[43] These characteristics, particularly pattern propagation, are very interesting from a compositional point of view. Traditionally, composers have

[43]C. Langton, "Self-Reproduction in Cellular Automata," *Physica D* 10 (1984): 135-144.

employed pattern propagation intuitively: sequences are a good example of pattern propagation. However, cellular automata algorithmic procedures applied to music allow pattern propagation to be formalized at a higher level.

Cellular automata have already been proven to be a very fruitful for algorithmic composition. Composer Eduardo R. Miranda, of SONY CLS, Paris, has developed a computer program for the PC, CAMUS, which maps two-dimensional cellular automata onto musical parameters.[44] His piece *Entre l'Absurde et le Mystère* was composed with this program.

L-Systems

Another field of mathematics who has recently been introduced in algorithmic composition is *L-Systems*. L-systems were originally proposed by Aristid Lindenmayer in 1968 as the basis for an axiomatic theory of development.[45]

[44]E. R. Miranda, "Cellular Automata Music: An Interdisciplinary Project," *Interface* 22 (1993): 3-21; E. R. Miranda, "Music Composition Using Cellular Automata," *Languages of Design* 2 (1994): 105-117.

[45]A. Lindenmayer, "Mathematical Models for Cellular Interactions in Development, Parts I-II," *Journal of Theoretical Biology* 18 (1968): 280-315.

They were subsequently used to model living organisms.[46] Basically, L-Systems consist of a set of "substitution rules" recursively applied to an initial string of symbols, and interpreting the resulting string of symbols (usually more complex) as structural elements of the organism. The substitution rules determine how each symbol in the current generation should be replaced. Because of the very rich structures L-Systems generate, they can be a interesting source for algorithmic composition. Some composers have already successfully applied L-System to composition, such as Gary Lee Nelson in his *Summer Song* for solo flute.

[46]A. Lindenmayer and P. Prusinkiewicz, *The Algorithmic Beauty of Plants* (New York: Springer, 1990).

Genetic Algorithms

Genetic algorithms[47] are one of the latest mathematical techniques applied to music composition[48] as well as to sound synthesis.[49] A genetic algorithm is a procedure which, in effect, searches a potentially vast solution space for an optimal solution to a given problem.

Solutions are encoded as strings over a finite alphabet. A *fitness function* (or *objective function*) is used to evaluate each string (solution.) Bits and pieces of the fittest strings (solutions) are used to generate new strings (solutions.) Each time step (or generation) of the algorithm produces a new population of possible solutions based on the population from the previous generation.

---

[47]J.R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (Cambridge, MA: The MIT Press, 1992).

[48]D. Goldberg and A. Horner, "Genetic Algorithms and Computer-Assisted Music Composition," in *Proceedings of the 1991 International Computer Music Conference* (San Francisco: International Computer Music Association, 1991), 479-482; D. Horowitz, "Generating Rhythms with Genetic Algorithms," in *Proceedings of the 1994 International Computer Music Conference* (San Francisco: International Computer Music Association, 1994), 142-143; J. Biles, "GenJam: A Genetic Algorithm for Generating Jazz Solos," in *Proceedings of the 1994 International Computer Music Conference* (San Francisco: International Computer Music Association, 1994), 131-137.

[49]J. Beauchamp, A. Horner, and L. Haken, "Genetic Algorithms and Their Application to FM Matching Synthesis," *Computer Music Journal* 17(4) (1993): 17-29.

Genetic algorithms, when applied to music, follow these general procedures. The strings (or solutions) correspond to musical entities, such as motives or even whole phrases. The initial population of solutions (generation 0) can be supplied by the composer or generated by the computer itself through stochastic means. The fitness tests have to be necessarily supplied by the composer. These determine which solutions (musical entities) "survive" and which "die." When an optimal set of solutions is encountered——a process whose length depends on the initial population, the solution space and the fitness test——the program stops. Genetic algorithms incorporate techniques directly from natural genetics:

- Natural selection: strings with a good fitness value survive from one generation to the next with high probability; strings with a poor fitness value fail with high probability.
- Reproduction: two strings chosen via natural selection mate (via crossover, which picks a position within the strings at random and exchanges the information of the two strings) to produce new strings. Musically, this is analogous to constructing a new phrase or motive from several others.

- Mutation: strings can undergo spontaneous changes (with small probability) to produce new strings (new musical entities) in a different part——that is, for a small subset——of the solution space.

These operations require the use of probability functions and are, therefore, stochastic in their nature.

Other Techniques

Other mathematical models that have been adopted for algorithmic composition in recent years include noise (specially the so-called *1/f noise* or *pink* noise,) number theory (number series, the Morse-Thue sequence, etc.,) and fractals. These will be explored in subsequent chapters. The preceding panoply of processes, issuing from the latter-day boom in computer science, constitute the most important methods used in algorithmic composition today. Undoubtedly, as computational power increases and new scientific disciplines flourish, existing algorithmic procedures will be further explored and new ones will be incorporated. However, these techniques, this writer believes, should be always considered a mere tool at the composer's disposal. The material they generate can be regarded as musical "prime matter" which can be later modified, transformed and incorporated according to the composer's own aesthetic judgement. Algorithmic methods in music should be used primarily as a source of inspiration, and not as a one-step musical solution.

CHAPTER III

TECHNIQUES AND APPLICATION OF MATHEMATICAL MODELS IN MUSIC

Mapping Techniques

Transcribing the numerical output of an algorithm to musical events is done through a mathematical operation known as *mapping*. Mapping consists of creating a one-to-one correspondence between the algorithm's numerical output and a set of ordered musical events. Musical events can be single pitches, motives, or even fully-fledge phrases. For simplicity, in most examples we will consider musical events associated to pitch only.

Two types of mapping techniques will be used throughout this text: *modulo-based* mapping and *normalized* mapping.

Although most processes can be applied any of these two types of mapping techniques, some are most appropriately transcribed using one specific mapping type.

Modulo-based Mapping

A modulo operation is a simple mathematical procedure performed between two whole numbers. It consists of dividing the first number by the second and taking the reminder. For instance

$$10 \bmod 3 = 1,$$

since the reminder of 10 divided by 3 is 1. Put more generally,

$$x \bmod y = n$$

When applying this mapping method to a process' numerical output set, x represents the process's output and y the total number of elements in our ordered musical event set. The result, n, is always a number between 0 and y-1, which corresponds to the index of the element in our musical event set. Modulo-based mapping is best suited to processes in which the numerical output set grows without bound, such as in number series. For processes in which the numerical output is bounded in very small intervals (such as the Logistic equation,) the modulo mapping is completely unpractical, since it takes into account only the integral part of the process' output. For these cases the best type of mapping is the normalized mapping.

Normalized Mapping

This mapping method uses the following formula to make the correspondence between the algorithm's data and the ordered elements in the musical event set:

$$\text{Event No.} = \left[\frac{(\text{value} - \text{minval}) * \text{numevents}}{\text{maxval} - \text{minval}} \right]$$

Figure 1. Formula used in normalized mapping

where "value" is the current value of the process being mapped, "minval" is the minimum value in the process' data set, "maxval" is the maximum value, and "numevents" is the number of elements of our event set. The brackets ([]) indicate to take only the integral part of the result, disregarding any decimals. The normalized mapping method effectively maps the data set onto the interval [0, 1]. The minimum and maximum values in the process' data set must be computed beforehand. This mapping method is more appropriate for processes whose output is bound in small intervals. In addition, this method maps much more faithfully the contour (the way the numerical data behaves) onto the musical event set.

To show the differences among these two mapping schemes an example follows. First, we will map the Logistic equation,——which will be discussed in more depth later in this text——using both mapping techniques. The mathematical formulation of this equation is as follows:

$$x_{n+1} = x_n*\mu*(1-x_n)$$

The output of the Logistic equation is bound in the interval [0, 1]. Our musical event space will be the pitches of a chromatic scale from C_4 to B_5 (24 events):

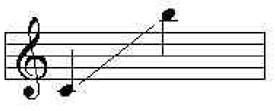


Figure 2. Pitch event space

A sequence of ten events is then generated using a value of $\mu = 3.57$.

The equation's numerical output is as follows:

0.892500	0.342519	0.803963	0.562655	0.878486
0.381093	0.842024	0.474880	0.890247	0.348814

Applying the modulo mapping to the above values yields the following results:

```
0    0    0    0    0    0    0    0    0    0
```

which transcribes as follows:

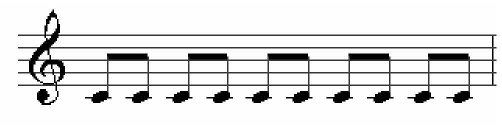


Figure 3. Modulo-based mapping

Using the normalized mapping method yields the following sequence (we know beforehand that the maximum and minimum values of the algorithm are 1 and 0, respectively):

```
21    8    19    13    21    9    20    11    21    8
```

which transcribes as follows:

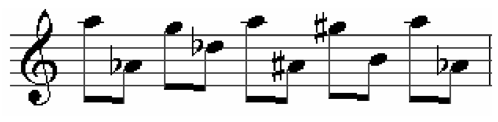


Figure 4. Normalized mapping

As we can readily see, the normalized mapping is more appropriate for this process in particular.

Because each process behaves differently, one must first analyze the numerical output to decide which mapping method is more suitable. Some processes must be mapped using only one of the two methods, some others give satisfactory results using any of the two. In the latter case, the decision is entirely up to the composer.

Stochastic Algorithms

Randomness and Probability Distributions

Mathematically, a stochastic process is defined as a set of quantities randomly distributed. Stochastic formulas are used by statisticians in data in order to detect patterns so that a more consistent description for that data can be achieved.[1] In music, stochastic processes are used inversely——that is, to provide a structural

[1]K. Jones, "Compositional Applications of Stochastic Processes," *Computer Music Journal* 5(2)(1981): 45.

framework for the synthesis of collections of musical events.

Stochastic music processes require random strings of numbers. Randomness is something very difficult both to define mathematically and to recreate in a computer. In a computer nothing can be truly random; a computer simulates randomness through complex mathematical procedures and hence the generated strings of random numbers are really "pseudo-random." The process of random number generation on a computer is, in fact, totally deterministic. What seems to be strings of random numbers are really small portions of a very vast cycle of numbers that repeat after a finite number of iterations. Each new number of the sequence is created by applying a mathematical transformation to the previous number. Random seeds are used to specify where to start in the sequence. Identical seed values generate identical strings of pseudo-random numbers.

In all forthcoming examples we will use one of the best random number generator functions available.[2] This function generates strings of random numbers with a period of repetition larger than 10^{18}. When called, the function——

[2]W. T. Vetterling, ed., *Numerical Recipes in C* (New York: Cambridge Univ. Press, 1997).

which we will call simply R——returns a random number in the interval [0,1.)

Mapping strings of random numbers to musical events is a straightforward procedure. It begins with the consideration of an event-set to which the mapping will be applied. The elements of this collection could be any musical structure, from individual pitches, to short motives, to fully-fledged musical phrases. For the sake of simplicity, the following example will concern only pitch. The composer would first decide upon a pitch-space, to be the set of pitches to be mapped to the random generator function. This could be any collection of pitches——for instance, a two-octave chromatic scale from C_4 to B_5 (see Fig. 2.)

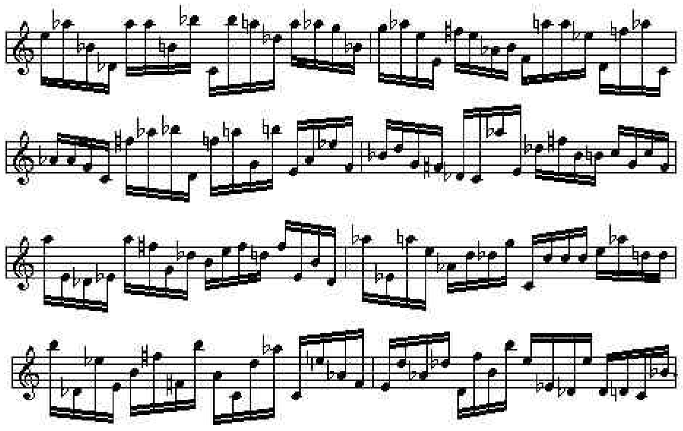
This set contains 24 pitches, numbered 0 (C_4) to 23 (B_5.) The random number generator function——R——produces pseudo-random numbers between [0, 1.) Since the output of the function is bounded in a small interval, the most appropriate mapping method is the normalized mapping.

Because the minimum and maximum values of the function are 0 and 1 respectively, the mapping formula is equivalent to:

$$\text{Event No.} = [R*24]$$

The result will always be a number between 0 and 23, which is precisely within the range of our pitch set. Zero corresponds to C_4, 1 to $C^{\#}_4$, 2 to D_4, and so forth.

Applying this procedure 128 times to our example pitch-set we get the following sequence of notes (arbitrarily assigning sixteenths as note durations):



Musical Example 1

Any set of musical parameters would be mapped similarly. In this particular example, all pitches in the set are equiprobable, that is to say, they all have the same probability of occurring. This is in effect equivalent to using a 24-sided fair die to choose the pitches. If all elements in our set have the same probability, we have an *aleatoric process*, in which all elements are equally probable to take place. In aleatoric processes, sufficiently long sequences of generated elements will show no discernible pattern (see Musical Example 1.)

A step further consists in introducing probability for each element in our set. Each element is assigned a "probability of being chosen." This way we obtain a "weighed" set, where elements with a higher probability will have a greater chance to occur. An example illustrates this point.

Two parameters, pitch and duration, will be mapped. For simplicity, the pitch set will be a one-octave chromatic scale from C_4 to B_4. Let our duration set be as follows:

{whole, half, quarter, eighth, sixteenth}

Next, we assign the following probabilities to the ordered members of our sets:

Element	Probability	Index
C	.17	0
C#	.02	1
D	.07	2
D#	.03	3
E	.15	4
F	.12	5
F#	.01	6
G	.16	7
G#	.08	8
A	.10	9
A#	.02	10
B	.07	11

and

Element	Probability	Index
Whole	.03	0
Half	.06	1
Quarter	.11	2
Eighth	.34	3
Sixteenth	.46	4

The probabilities can take any value between 0 and 1. The sum of the probabilities of all members should always equal 1. An element *e* with a probability of, say, .20, means that 20% of the generated sequence of events is likely to be the element *e*. In other words, for sufficiently large sequences, element *e* will occur 20% of the time. Adding probability introduces a new step in the mapping process, which is as follows:

1. Draw a random number *r* between [0, 1)
2. Multiply *r* by the number of elements of our pitch set and take the integral part of the result: [*r**12]. This gives a value *n* between 0 and 11, which corresponds to one of the elements of the set.
3. Look up the probability value (P_n) for element *n* in the set.

   3.1 Draw a new random number r_2.

3.2 Is $r_2 \leq P_n$? If the answer is affirmative then continue, otherwise go to step 1.

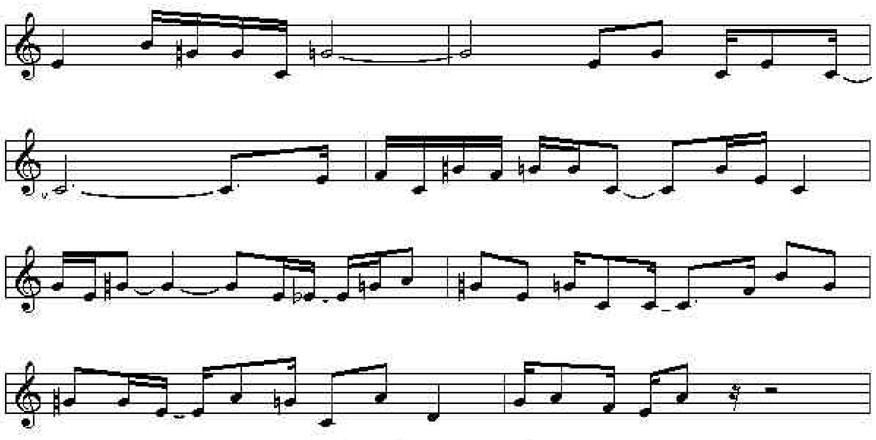
4. Accept the element.
5. Follow steps 1 through 4 again for the duration set (5 elements in our example) and assign duration to previously selected pitch.
6. Repeat until the desired number of elements have been chosen.

Step 3.2 is the key here. Because all possible values generated by the random number generator are equiprobable, the probability of any random number being equal or smaller to the probability of an element being chosen is the equivalent to the probability of that element being chosen, put symbolically:

$$r_2 \leq P_n \rightarrow n$$

This process, known as the *Monte Carlo* method, can be used to determine if an element is selected or not.

Applying it to our choice sets yields the following sequence:


Musical Example 2

It is clear that the elements with high probabilities (such as pitch-classes C, E, and G, and sixteen note values) appear more often than elements with a lower probability. For instance, in this sequence of 50 elements, pitch-class C appears 9 times, or 18%, in accordance with its probability. Likewise, there are 24 sixteenths (48% of the note values,) which is also in accordance with its probability value.

Probability functions can be used to distribute probability values among the elements of our sets. Gaussian distribution, for instance, provides an appropriate framework. Gaussian distribution[3] is the most commonly observed in nature and the starting point for modeling many natural processes. For instance, the

[3]Xenakis, *Formalized Music*, 14-15.

distribution of heights among a large population follows Gaussian distribution. Gaussian distribution is defined mathematically as follows:

$$f(x)= \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(x - \mu)^2}{2\sigma^2}}$$

Figure 5. Formula for Gaussian distribution.

Where μ is the mean of the distribution, σ is the variance, and e is the base of natural logarithms (the transcendental constant 2.718281828459...) Gaussian probability distributions graphs exhibit a characteristic "bell" shape:

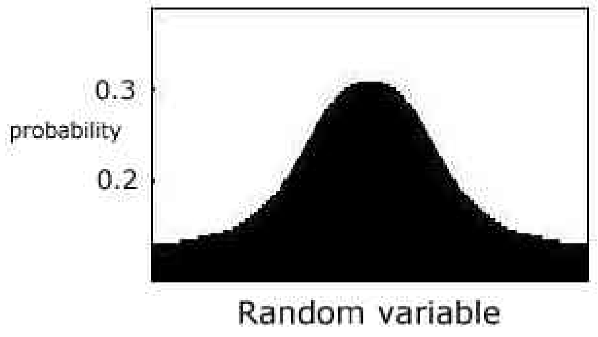


Figure 6. Gaussian distribution bell-shaped curve

In musical Example 2 (above,) the probabilities were calculated——with the help of a computer program——according to Gaussian distribution:

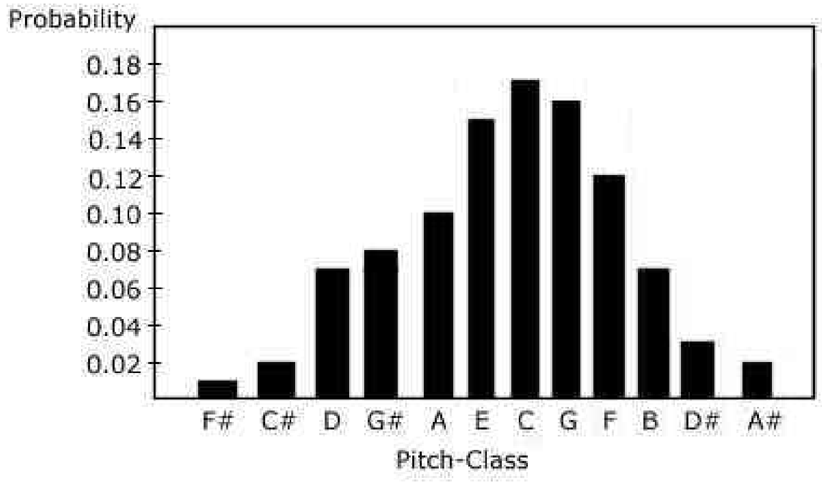


Figure 7. Pitch-class distribution in Musical Example 2

Note how the distribution of probabilities resembles the bell shape characteristic of Gaussian distributions.

In Musical Example 2, the highest probabilities were deliberately assigned to specific pitches——C, G, F, and E—— to give the feeling of a tonal center.

Although Gaussian probability distribution is by far the most important, there are other types of probability distribution that can be incorporated in stochastic music.

One of such is Poisson's probability distribution.[4] Many natural processes follow Poisson's law of probability distribution, such as radioactive decay, the number of failures in an electronic device——such as a hard drive——

[4]Ibid., 12-13.

over a given time period. In stochastic music, Poisson's law is useful to control the density (the number of sounds/events per a given unit of time.) Given a mean density, λ, the probability of a particular density *n* is derived from Poisson's law, defined as follows:

$$P(n) = \frac{\lambda^n e^{-\lambda}}{n!}$$

Figure 8. Poisson distribution formula

where λ is the mean density, n is any particular density and n! is n factorial (the product of all integers from 1 to n.)

Poisson's probability distribution is specially useful when dealing with large masses of sounds. A typically applicable problem would be thus: given a sounds mass with a mean density of 3 sounds per second, what is the probability of a particular density of, say, 5 sounds per second? Applying Poisson's law affords the following solution:

$$P(5) = \frac{3^5 e^{-3}}{5!}$$

or approximately 0.1.

Applying probability distribution to sound events can provide them a with coherent structure, a structure which is, quite literally, borrowed directly from the laws of nature.

Markov Chains

Markov analysis studies a sequence of events and examines the propensity of one event to be followed by another. Using this analysis, sequences of random but interrelated events can be generated.[5]

In the processes studied so far, the probabilities for the occurrence of an event remained unchanged. Each event of our set had a fixed, stationary probability. Markov chains provide a more refined mechanism to control probabilities.

Using Markov chains allows the composer to control sequences of events by making the probability of any particular event depend on the previous. A Markov chain requires a matrix of probabilities, where the row of probabilities for one event is used to generate the next,

[5]Ibid., 43-109.

the row of probabilities of this new event is used to generate the next, and so forth.

The relationship between events in a Markov chain exhibit different properties according to how their probabilities are arranged in the matrix. What follows is a brief description of these properties[6]:

- *Accessibility* ($e_x \rightarrow e_y$): An event e_y is accessible to another event e_x, if e_x can be followed by e_y.
- *Communicability* ($e_x \leftrightarrow e_y$): Two events *communicate* if they are *both* mutually accessible. Communication between events can be *reflexive* (if an event is accessible to itself,) *symmetric* (reciprocal accessibility between any two events,) or *transitive* (if event e_x communicates with event e_y, and e_y communicates with event e_z, then e_x communicates with e_z.)

Events can be further structured into *equivalence classes* of communicating events. Events in one equivalence class can not *communicate* with events in another equivalence classes, but events in one equivalence class may be *accessible* from other events in a different equivalence class. Events in equivalence classes may be

[6]Jones, "Compositional Applications of Stochastic Processes," 47-48.

divided into to groups: *recurrent* events, which are events that are *certain* to occur at some point after they have occurred once; and *transient* events: events that have a probability of not recurring. If an event is recurrent, all events with which it communicates within its class are also recurrent. Analogously, all events that communicate with a transient event, are also transient. Two types of equivalence classes can therefore be defined: *recurrent classes*, which contain recurrent events, and *transient classes*, which contain transient events. Events in a recurrent class cannot access events in other classes. Because events can not communicate between classes, once the sequence of events has left a transient class, it can never return to it. For the same reason, if a sequence enters a recurring class, it can no longer leave it. Recurrent classes form confined sets. Markov chains must contain at least one recurrent class, they can not consist of only transient classes. Several types of Markov chains can be defined according to the types of classes it contains:

- *Irreducible*: Markov chains containing only one recurrent class, and no transient classes.

- *Ergodic*: Markov chains containing only one recurrent class and several transient classes.

To further clarify all this theory and to show how Markov analysis is applied in music composition, a practical example follows:

A Markov chain with 10 musical events creates an order 10 matrix. The event-relation diagram is as follows:

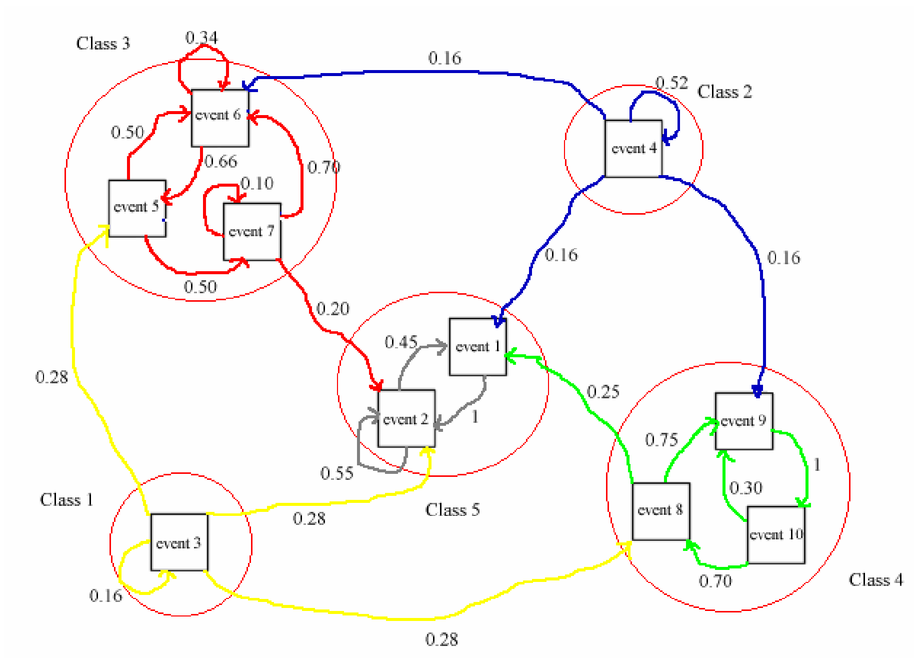
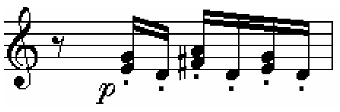


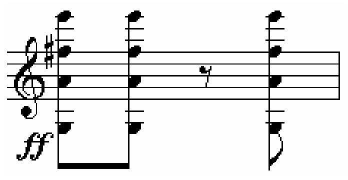
Figure 9. Diagram of classes

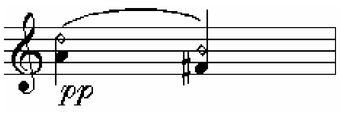
Events are grouped in 5 classes: four transient (classes 1, 2, 3, and 4,) and one recurrent (class 5.) The example, therefore, is an ergodic Markov chain. Note also how events 2, 3, 4, 6, and 7 show reflective communication. The correspondent probability matrix is as follows:

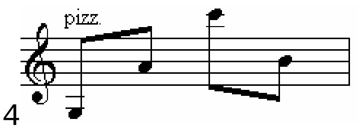
	Next Events									
Current Events	1	2	3	4	5	6	7	8	9	10
1	0	1	0	0	0	0	0	0	0	0
2	.45	.55	0	0	0	0	0	0	0	0
3	0	.28	.16	0	.28	0	0	.28	0	0
4	.16	0	0	.52	0	.16	0	0	.16	0
5	0	0	0	0	0	.50	.50	0	0	0
6	0	0	0	0	.66	.34	0	0	0	0
7	0	.20	0	0	0	.70	.10	0	0	0
8	.25	0	0	0	0	0	0	0	.75	0
9	0	0	0	0	0	0	0	0	0	1
10	0	0	0	0	0	0	0	.70	.30	0

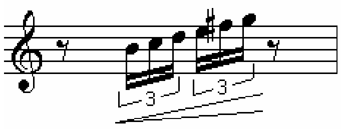
The event space remains to be defined. For this example, we will use events to construct a solo violin composition:

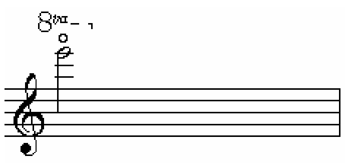
Event 1 

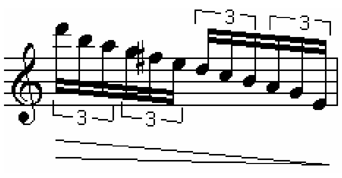
Event 2 

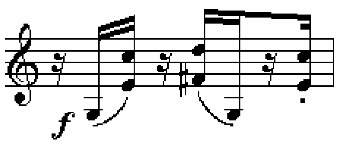
Event 3 

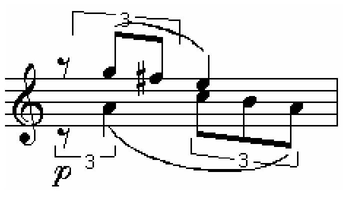
Event 4 

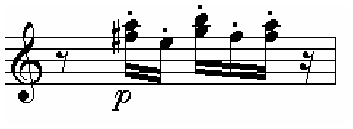
Event 5 

Event 6 

Event 7 

Event 8 

Event 9 

Event 10 

The procedure to obtain a sequence of events according to the probability matrix is as follows:

1. Randomly select a first event from the event list by drawing a number between 1 and the number of events.
2. Draw a number randomly between 1 and the maximum number of events, r.
3. In the matrix, check the probability setting of current event at column r, P_r.
4. If P_r is nonzero, then:
   - 4.1 Draw a random number n between 0 and 1.
   - 4.2 if $n \le P_r$ select event in column r as the next event (Monte Carlo method.)
   - 4.3 if $P_r = 0$ or $n > P_r$ then to back to step 2 (no next event selected yet.)
5. Repeat from step 2 until the desired number of events in the sequence has been reached.

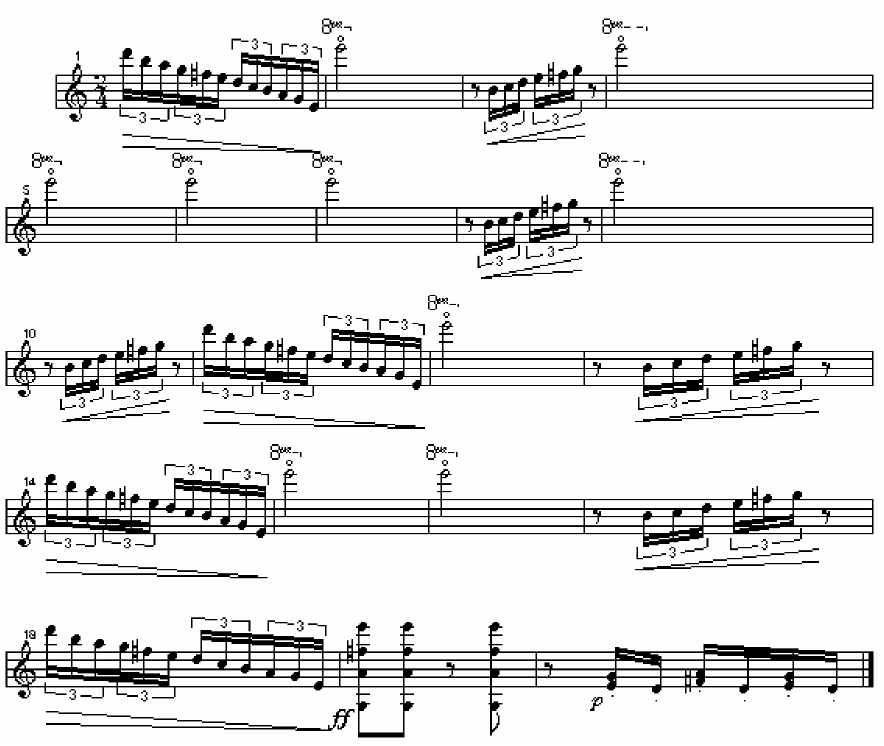
Applying this process to our event space, three possible sequences of 20 events each were generated:

s1 = e7,e6,e5,e6,e6,e6,e6,e5,e6,e5,e7,e6,e5,e7,e6,e6,e5,e7,e2,e1

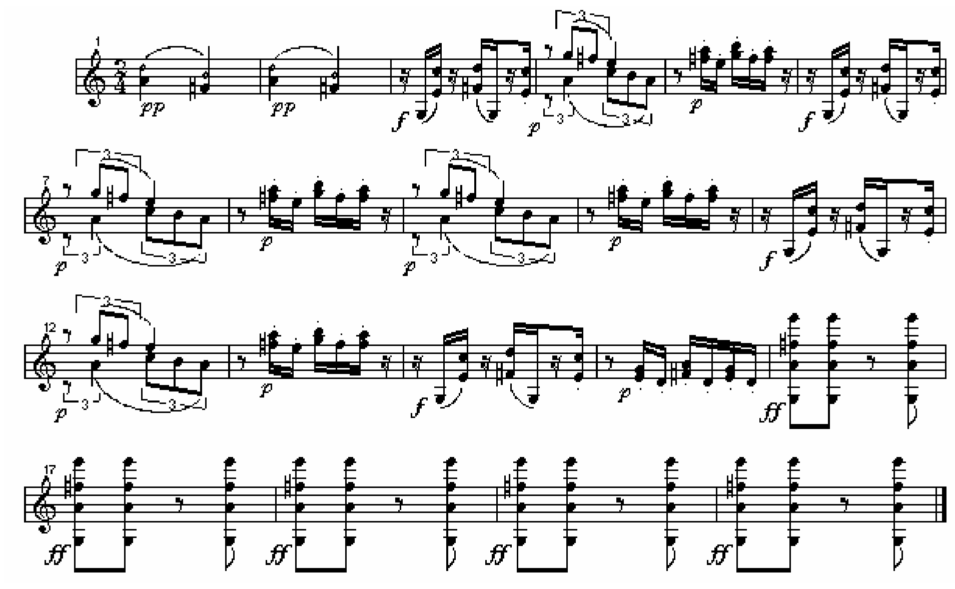
s2 = e3,e3,e8,e9,e10,e8,e9,e10,e9,e10,e8,e9,e10,e8,e1,e2,e2,e2,e2,e1

s3 = e4,e4,e4,e4,e4,e6,e6,e5,e7,e2,e2,e2,e2,e1,e2,e2,e1,e2,e2,e1

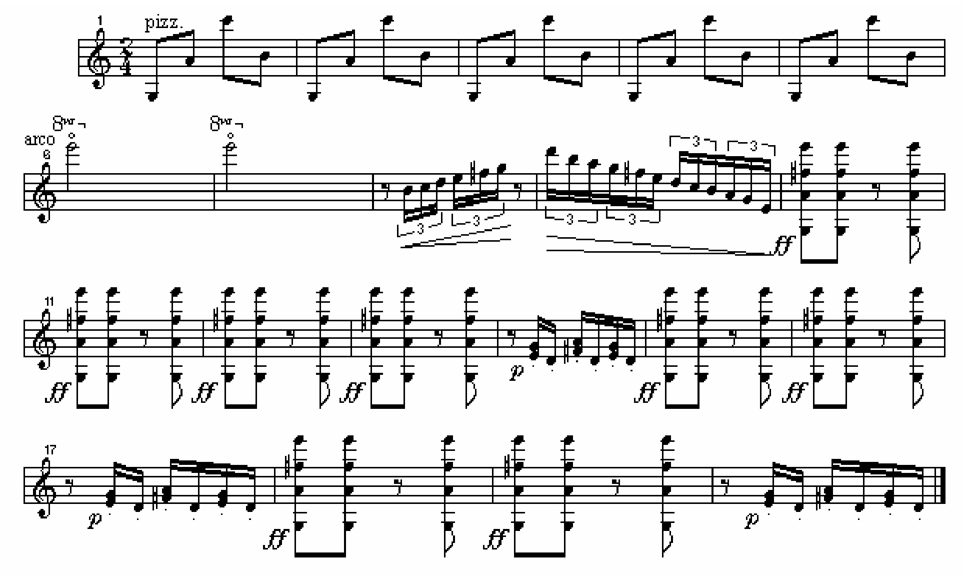
Transcribed to music, they become:



Musical Example 3. Music from sequence s1



Musical Example 4. Music from sequence s2



Musical Example 5. Music from sequence s3

Comparing these generated sequences with the event diagram of the Markov chain clearly shows how the different classes affect the musical output. For instance, once a process enters a recurrent class (in this case class 5, with contains events 1 and 2,) it never leaves it. Musically, this yields a repetition of events 1 and 2.

A very interesting kind of Markov chain is the so-called *random-walk* procedure. An event e_x in a *random-walk* procedure can only by followed by one of its neighboring events, e_{x-1} or e_{x+1}. The matrix for this kind of procedure has nonzero probability entries on either side of its main diagonal and zeroes everywhere else. Using our example, the matrix would be as follows:

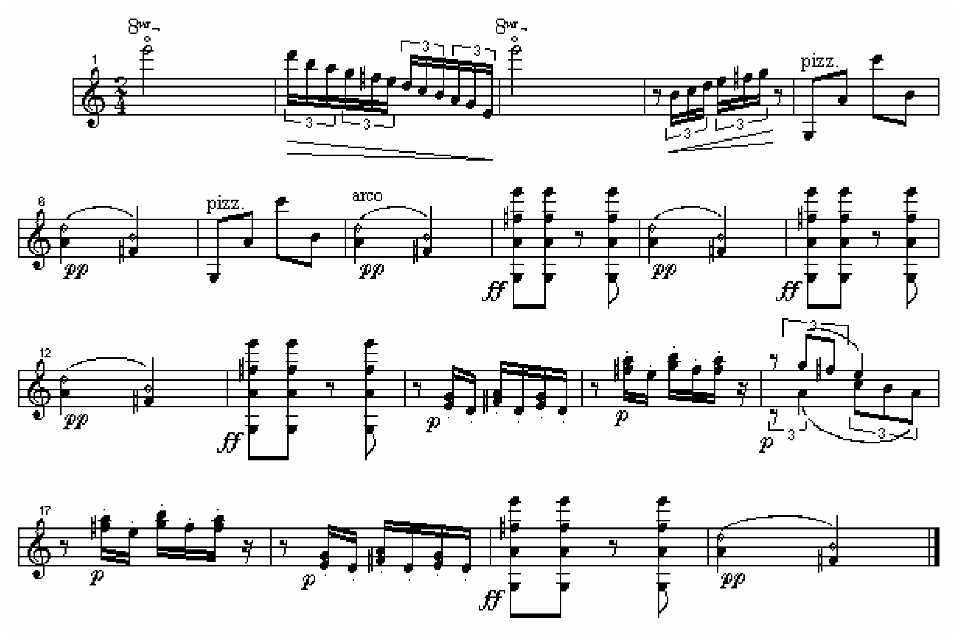
	Next Events									
Current Events	1	2	3	4	5	6	7	8	9	10
1	0	.50	0	0	0	0	0	0	0	.50
2	.50	0	.50	0	0	0	0	0	0	0
3	0	.50	0	.50	0	0	0	0	0	0
4	0	0	.50	0	.50	0	0	0	0	0
5	0	0	0	.50	0	.50	0	0	0	0
6	0	0	0	0	.50	0	.50	0	0	0
7	0	0	0	0	0	.50	0	.50	0	0
8	0	0	0	0	0	0	.50	0	.50	0
9	0	0	0	0	0	0	0	.50	0	.50
10	.50	0	0	0	0	0	0	0	.50	0

In this case, a 50% probability has been assigned to each event. Note also how event 1 can be preceded by event 10, and event 10 can be followed by event 1, thus creating a circular random-walk. If we apply this new process to our event space, we get the following sequences:

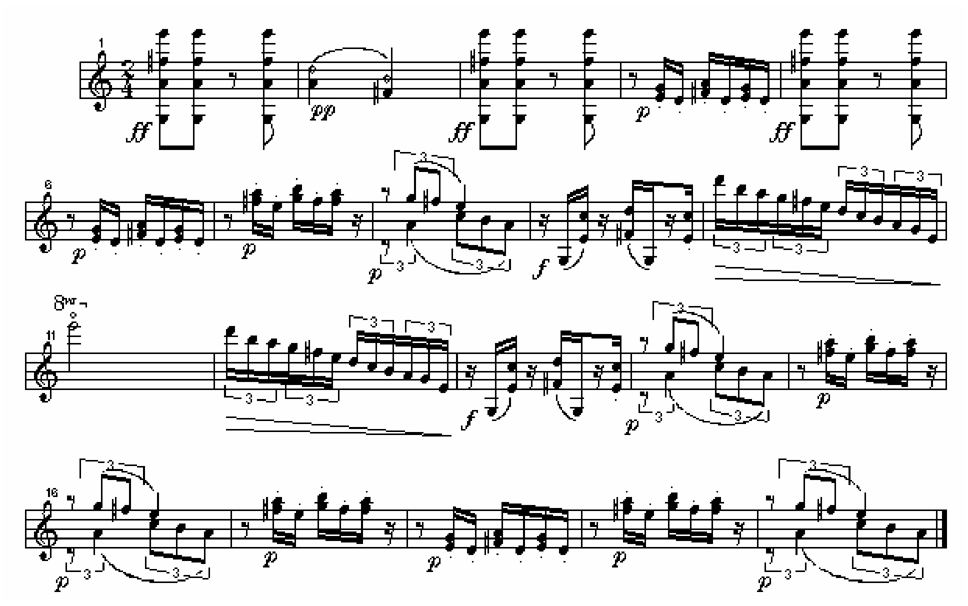
s4 = e6,e7,e6,e5,e4,e3,e4,e3,e2,e3,e2,e3,e2,e1,e10,e9,e10,e1,e2,e3

s5 = e2,e3,e2,e1,e2,e1,e10,e9,e8,e7,e6,e7,e8,e9,e10,e9,e10,e1,e10,e9

which transcribe musically as follows:



Musical Example 6. Music from sequence s4


Musical Example 7. Music from sequence s5

Markov chain stochastic processes can be extended in a number of interesting ways. One is to increase the order of the Markov chain. For instance, three-dimensional Markov chains can be used to control the probability of occurrence of an event depending on the two preceding events. This increase of dependence between events allows the composer to construct more sophisticated structure patterns. Furthermore, we could use a four-dimensional Markov chain to make events depend on the three preceding ones. In general, events in a nth-dimensional Markov chain depend on their n-1 preceding events. Needless to say, very high-order Markov chains can become extremely complex and will require more computational power.

Another way in which Markov chains could be extended is to consider events not as single, predetermined entities, but to control them by a set of parameters which can themselves be structured stochastically with Markov chains.[7] For instance, events could be sound clouds where their density, pitch range, dynamic, etc. are controlled by a set of Markov chains. This allows the composer to create super-structures of great coherence.

Music from Chaos

Chaotic systems can be broadly categorized in two classes: *conservative*, in which energy is preserved, and *dissipative* in, in which energy radiated to the environment.[8] The great majority of natural phenomena represented by these systems are dissipative. Because dissipative systems loose energy continuously, their *phase space*——that is, the spatial location where the system evolves——is transformed towards a definite state (usually going first through an initial transitory stage) known as

[7]Ibid., 49-50.

[8]R. Bidlack, "Chaotic Systems as Simple (but Complex) Compositional Algorithms," *Computer Music Journal* 16(3) (1992): 33.

the *attractor* of the system. The attractor of a system can be a single point (limit-point attractor,) such as in pendulum motion; a set of points (cycle attractor,) among which the orbits of the system oscillate; or a complex shape, usually of fractal dimension (chaotic or strange attractor.) The phase space of conservative systems, however, is always constant.

The mathematical systems which model chaos are either *iterated systems*, formulated with non-linear difference equations, in which the orbits on the attractors consist of discrete points, or *continuous flows*, formulated with non-linear *differential* equations, in which the orbits are continuous, unbroken curves.[9]

In subsequent sections we will study the mapping onto musical event sets of a one-dimensional iterated map: the *Logistic equation*; three two-dimensional, dissipative iterated maps: the Hénon map, Martin's attractor, and Gingerbread Man attractor; and two tree-dimensional chaotic systems: an iterated map, Pickover attractor, and a continuous flow, Rössler attractor.

[9]Ibid., 34.

One-dimensional Chaotic Systems

Transcribing chaotic systems to music implies constructing a mapping process between the system's mathematical description and musical parameters. Since chaotic systems are modeled with non-linear equations, the process consists in mapping the numerical output of these equations onto a musical event space.

The Logistic Equation

The logistic Equation was first derived by Belgian sociologist and mathematician Pierre-François Verhulst in 1845 as he was trying to model population growth mathematically.[10] More than a century later, in 1976, R. May discussed its mathematical implications.[11] The Logistic Equation is formulated as follows:

$$x_{n+1} = x_n * \mu * (1 - x_n)$$

The key parameter is μ. This parameter, which can take any value in the interval [0, 4] determines the behavior of the equation in the long run, and thus, the type of musical

[10]Briggs, *Turbulent Mirror*, 56-57.

[11]R. May, "Simple Mathematical Models with very Complicated Dynamics," *Nature* 261 (1976): 459-467.

material that can be obtained from it. The numerical output of this equation is always a value in the interval [0, 1].

To map the logistic equation to musical parameters we must first define an event space onto which the numerical output of the equation will be mapped. This musical event-space is an ordered set of musical entities, which could be a collection pitches, durations, motives, etc. Let L be the numerical output of the equation and n_0 the number of elements in our event space. Since L is always a number in the interval [0, 1], then the mapping is performed using the normalized mapping method, as follows:

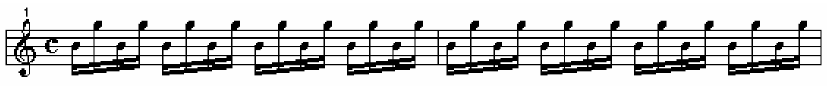
$$E = \mathrm{INT}(L * n_0)$$

The result (E) will be the index of the element in out ordered event set.

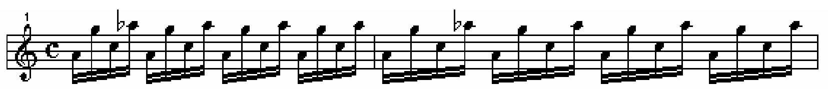
The output of the Logistic Equation depends on the value chosen for μ. For values of $\mu < 3$, the output quickly stabilizes to one value. As we increase the value of μ, the output stabilizes in 2 values, then 4, 8, 16...until we reach the critical point at $\mu = 3.569946$, where the output first becomes unpredictable. For values of μ between 3.57 and 4 (the maximum allowed value) the output is most of the time chaotic, but within this chaos, "islands of order" emerge, where the output oscillates between a specific number of values. Such a value of μ is 3.83, where the output oscillates between 3 values.[12]

To illustrate this, we will map several μ values of the Logistic Equation. For simplicity, our event space will be the pitches in a two-octave chromatic scale from C_4 to B_5 (see Fig. 2) ordered from 0 (C_4) to 23 (B_5.)

For μ = 3.3, the equation output settles oscillating between two values: 0.479427 and 0.823603, which transcribe musically to

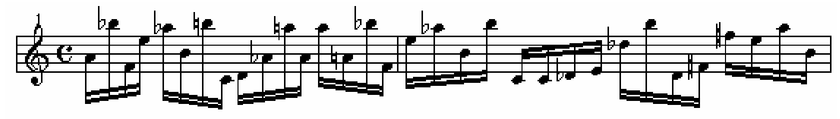

Musical Example 8

For μ = 3.5 it oscillates between 4 values: 0.382820, 0.500884, 0.826941 and 0.847977:


Musical Example 9

[12]Briggs, *Turbulent Mirror*, 56-57.

Finally, for μ = 4, it wanders chaotically between all possible values between 0 and 1. The musical transcription shows no discernible pattern:



Musical Example 10

In fact, the output of the logistic equation for a value of μ = 4 is indistinguishable from a randomly generated sequence of numbers, although it is completely deterministic. Note the resemblance of the musical output of Musical Example 10 and that of Musical Example 1, generated from a sequence of random numbers.

So far we have been working with specific values of μ. We can have a more general sense of the chaotic and orderly behavior of the logistic equation by making a "map" of continuos values of μ. When plotted, it looks as follows:

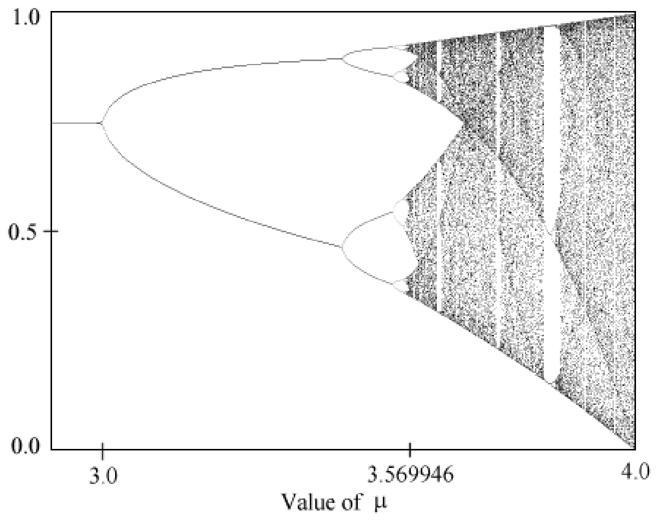
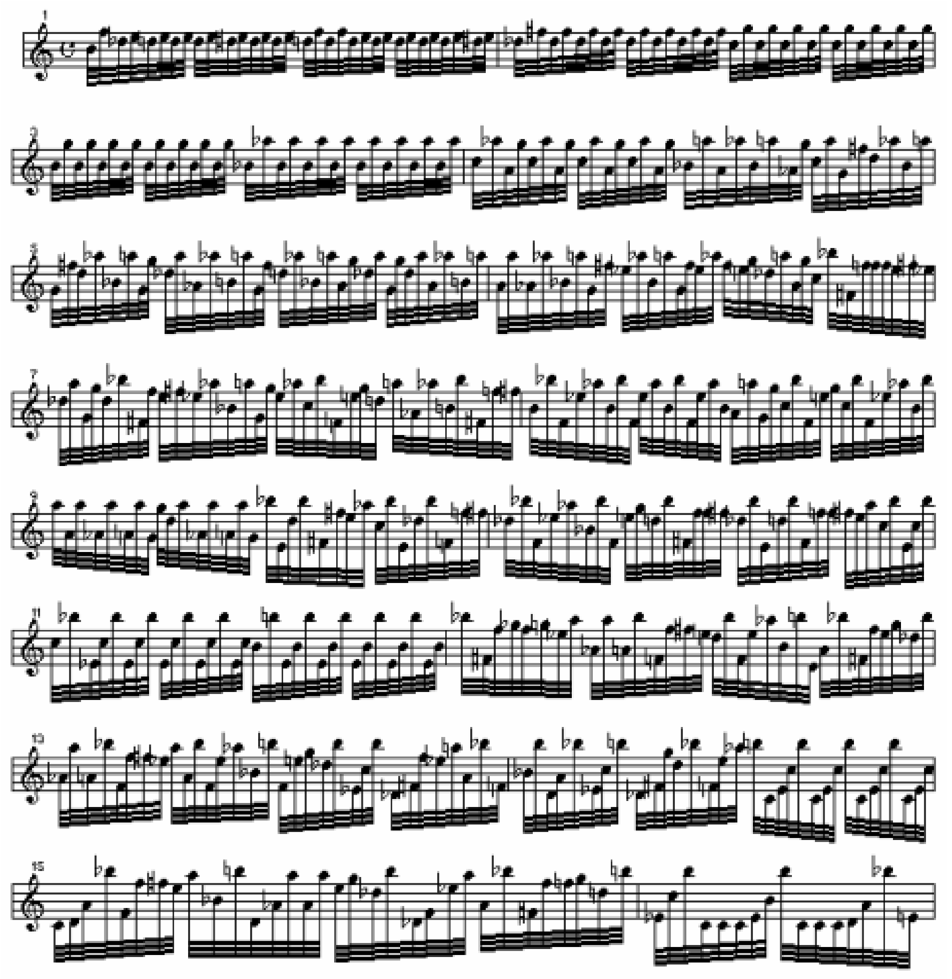


Figure 10. Bifurcation diagram of the Logistic equation for continuous values of μ.

This map shows how the output of the equation behaves for increasing values of μ. The dark regions in the picture correspond to the chaotic regions where the equation's output fluctuate wildly. Note how among these regions there are small windows of order (white bands.) The map also shows something common to many chaotic attractors and fractals: self-similarity at all scales.

A musical rendition of the logistic equation bifurcation diagram can be achieved by mapping sequences of increasing values of μ and adding then sequentially. The following musical example was constructed by mapping 30 sequences of 16 values each for increasing values of μ (2.9 to 4.0):



Musical Example 11

Observe how the features of the map are reflected musically: it starts orderly with several 2-note cycles, little by little it departs from order until it reaches total chaos (at measure 7.) It wanders chaotically for a while, but suddenly there is an outburst of order at measure 11: a 3-note cycle. This corresponds to a 3-cycle value for $\mu = 3.83$ (it is clearly seen in the map as a white vertical band towards to right, see Fig. 10.) Another chaotic region follows this 3-note cycle, followed by another window of order at the end of measure 14 (a 4-note cycle,) followed by chaos once again ($\mu = 4$.)

Aside from the logistic map, there are many other nonlinear equations that can be mapped to musical events in a similar way. Another of such equations which produces interesting results——with a behavior similar to that of the Logistic equation——is

$$X_n = \mu(3X_{n-1}-(4X_{n-1})^3)$$

This equation was derived by E. Lorenz as part of his research in climatology.[13] The parameter μ can take any value between [0, 1] and determines the behavior of the equation's output.

Each of these has its own characteristic "fingerprint," which transcribes musically when mapped.

[13]E. N. Lorenz, "Deterministic Nonperiodic Flow," *Journal of the Atmospheric Sciences* 20 (1963): 130-141.

Attractors from Two-dimensional Chaotic Maps

In this section we will study three two-dimensional iterated maps and how they are mapped to musical event sets. These attractors are Hénon Map, the Gingerbread Man attractor and Barry Martin attractor.

The Hénon map is named after its discoverer, Michel Hénon, an astronomer at the Nice Observatory in France.[14] It is a chaotic orbit in two dimensions. Although it is made entirely of lines, orbits on this attractor do not flow continuously, but jump from one location to another in the attractor. The Hénon attractor has an infinite amount of structure. Successive magnifications prove an ever increasing degree of detail. It is defined mathematically by the following iterated equations:

$$X_{n+1} = 1 + Y_n - a{*}X_n^2$$

$$Y_{n+1} = b{*}X_n,$$

$$X_0 = Y_0 = 0,\ a = 1.4,\ b = 0.3$$

When plotted, this chaotic attractor looks as follows:

[14]M. Hénon, "A two-dimensional Mapping with a Strange Attractor," *Communications in Mathematical Physics* 50 (1976): 69-77.

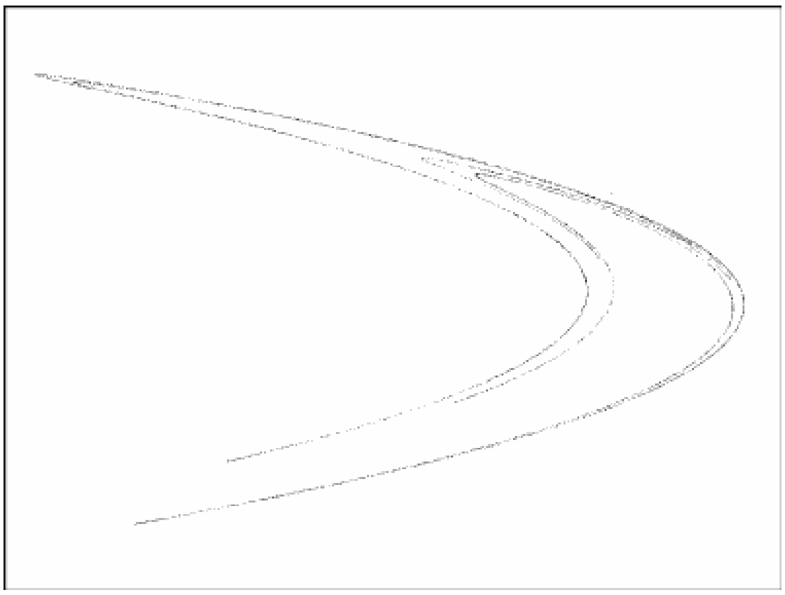


Figure 11. Hénon attractor

The Gingerbread Man attractor is another two-dimensional iterated map. It was proposed by R. Devaney in 1988.[15] It is defined by the following set of equations:

$$X_{n+1} = 1 - Y_n + |X_n|$$
$$Y_{n+1} = X_n$$

where $|X_n|$ is the absolute value of X_n and $X_0 = -0.1$, $Y_0 = 0$. When plotted, this attractors looks as follows:

[15]H. O. Peitgen and D. Saupe, eds., *The Science of Fractal Images* (Berlin: Springer, 1988), 149.

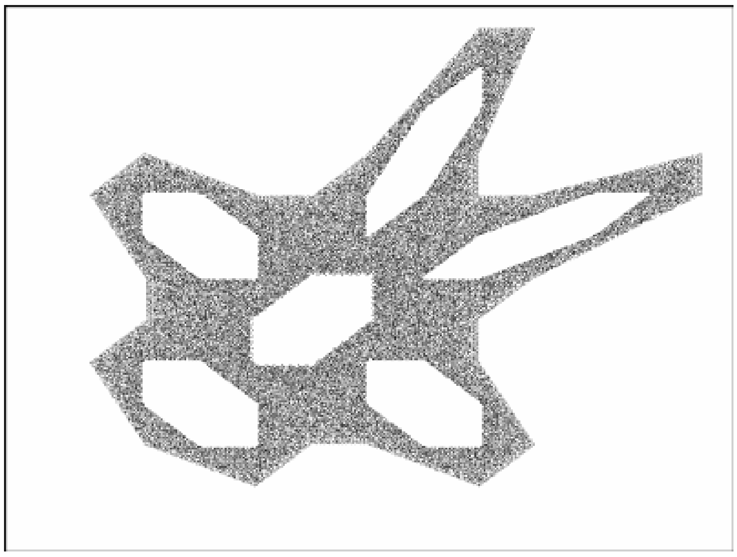


Figure 12. Gingerbread Man attractor

Lastly, the Martin attractor, proposed by Barry Martin of Aston University in Birmingham, England, and first discussed by A. K. Dewdney in 1986[16], is a two-dimensional orbit defined by the following iterated equations:

$$X_{n+1} = Y_n - \sin(X_n)$$
$$Y_{n+1} = a - X_n$$
$$Y_0 = X_0 = 0$$

where a is the controlling parameter. We will use a value of $a = \pi$ (3.1415926...) Barry Martin attractor looks as follows:

[16]A. K. Dewdney, "Computer Recreations," *Scientific American* (September 1986): 78-80.

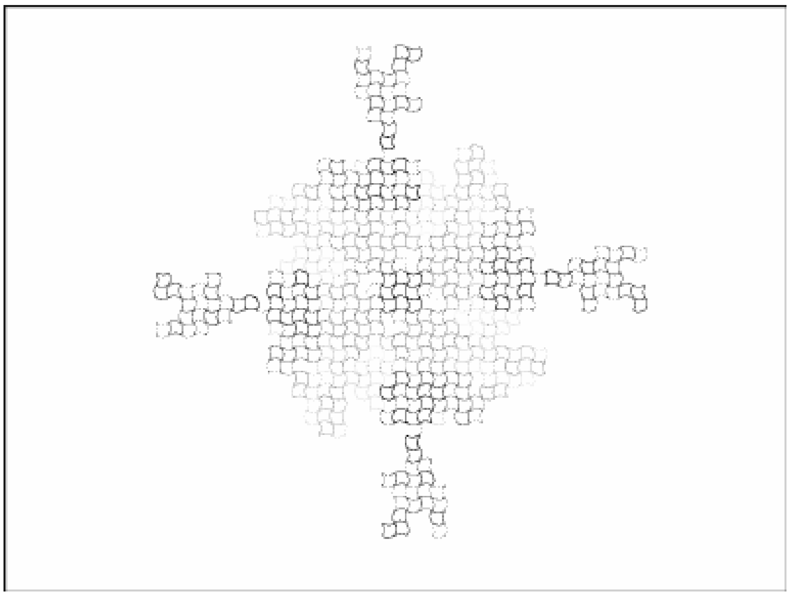


Figure 13. Martin attractor

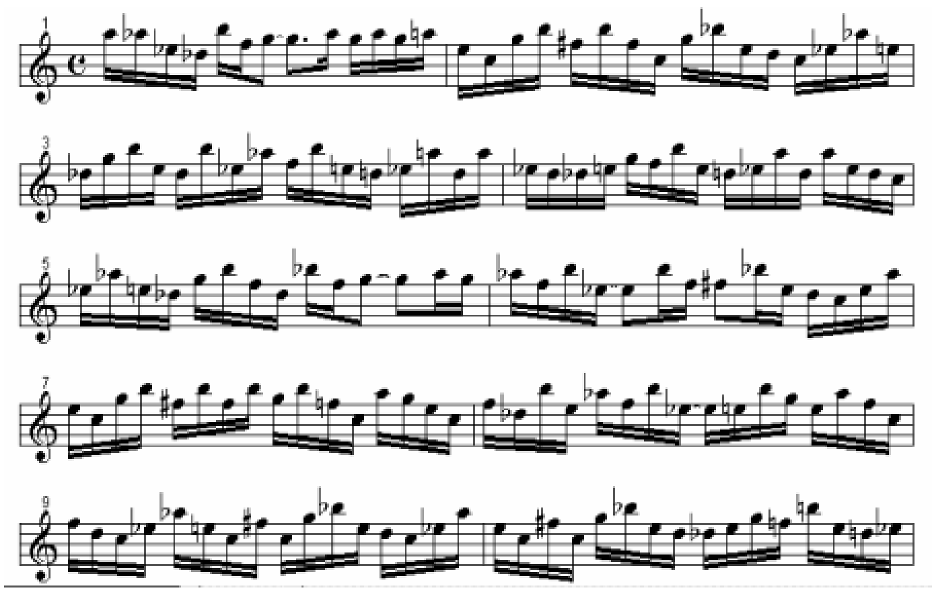
For consistency, all mappings will use the same event space: a one-octave chromatic scale from C_5 to B_5.

Because these are two-dimensional iterated maps, from each iteration of the equations, we get a new value for X_n and Y_n. This allows for different mapping alternatives. One possibility is to combine both coordinates in one value: the distance from the origin of coordinates to the point (X_n, Y_n) on the attractor. This value is given by the following formula:

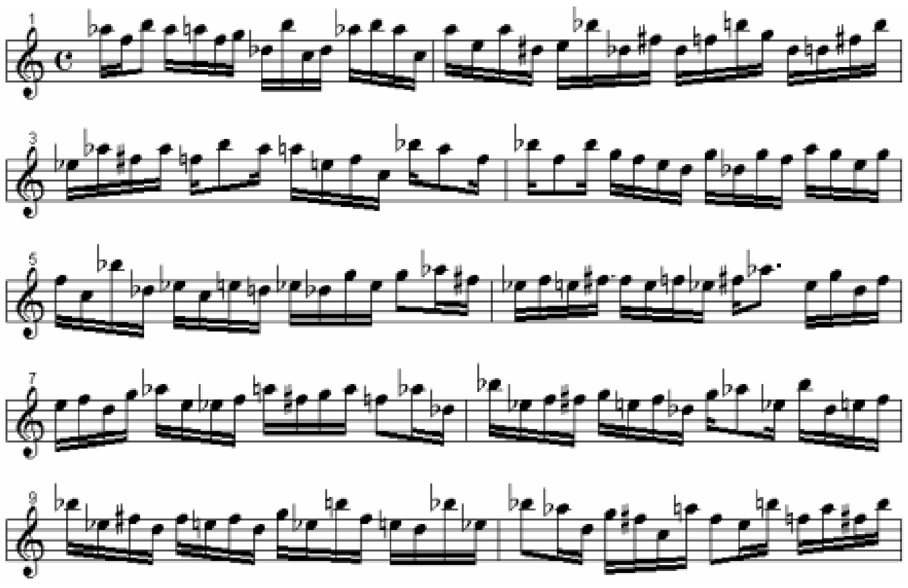
$$\sqrt{(X_n^2 + Y_n^2)}$$

In the final transcription to music, the repeating pitches are tied. Modulo-based mapping was used in all subsequent examples.

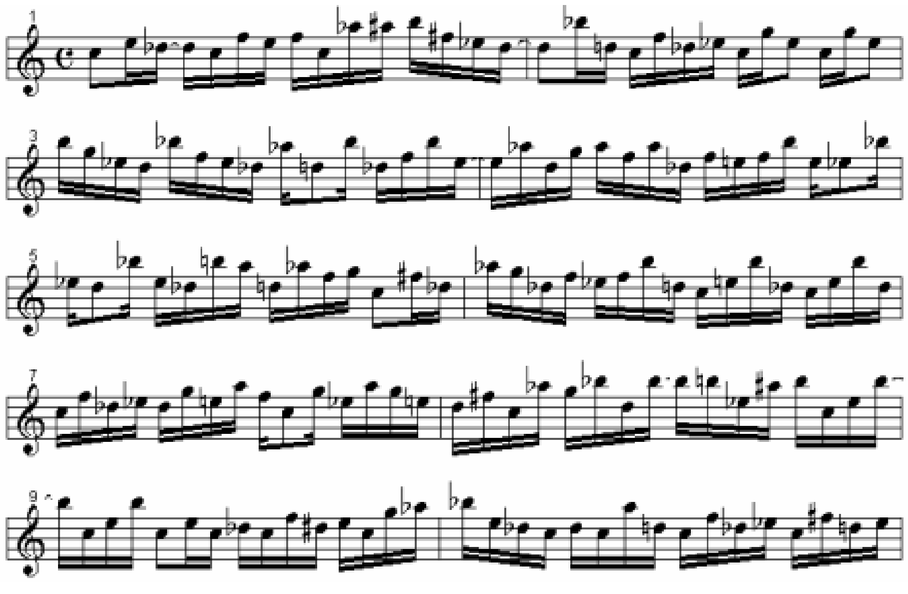
A sequence of 160 events (pitches) were generated from each attractor (sixteenths were chosen arbitrarily for the duration of the pitches in this case):



Musical Example 12. Music from Hénon attractor



Musical Example 13. Music from Gingerbread Man attractor



Musical Example 14. Music from Martin attractor

The following study of these melodies will reveal they are not as random as they may seem at first. First, we calculate the frequency of each pitch:

Music from Hénon attractor:

Pitch Class	Number of occurrences	Deviation from mean (13)
C	15	+2
C#	8	-5
D	12	-1
D#	18	+5
E	20	+7
F	13	0
F#	8	-5
G	23	+10
G#	13	0
A	7	-6
A#	5	-8
B	18	+5

Music from Gingerbread man attractor:

Pitch Class	Number of occurrences	Deviation from mean (13)
C	6	-7
C#	11	-2
D	9	-4
D#	13	0
E	16	+3
F	19	+6
F#	20	+7
G	17	+4
G#	16	+3
A	11	-2
A#	11	-2
B	11	-2

Music from Martin attractor:

Pitch Class	Number of occurrences	Deviation from mean (13)
C	26	+13
C$^{\#}$	20	+7
D	13	0
D$^{\#}$	22	+9
E	12	-1
F	15	+3
F$^{\#}$	4	-9
G	11	-2
G$^{\#}$	9	-4
A	4	-9
A$^{\#}$	12	-1
B	12	-1

It is evident from these tables that some pitches predominate over others. In a truly random collection of pitches, all pitches would have the same frequency. Does this frequency of occurrence tell us anything about their probability distribution? Let us rearrange them another way:

Hénon sequence:

Frequency	5	8	12	13	15	18	23	20	18	13	8	7
Pitch Class	A$^{\#}$	C$^{\#}$	D	G$^{\#}$	C	B	G	E	D$^{\#}$	F	F$^{\#}$	A

Gingerbread Man sequence:

Frequency	6	11	11	13	16	19	20	17	16	11	11	9
Pitch Class	C	C$^{\#}$	A	D$^{\#}$	E	F	F$^{\#}$	G	G$^{\#}$	A$^{\#}$	B	D

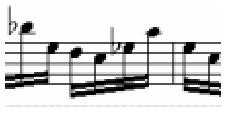
Martin sequence:

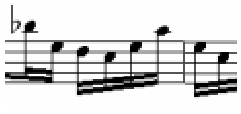
Frequency	4	11	12	15	20	26	22	13	12	12	9	4
Pitch Class	F#	G	B	F	C#	C	D#	D	A#	E	G#	A

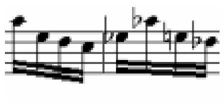
We have seen this type of probability distribution before: it is the Gaussian distribution. It is remarkable that all three sequences exhibit this type of probability distribution.

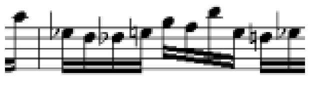
Observe also that the collection of predominating pitches (the ones with a high probability of occurrence) is different in each of the sequences (and will be, in fact, in any sequence derived from a different attractor.) This gives unique characteristics to each sequence.

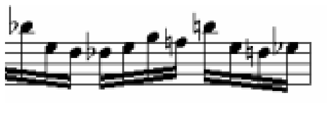
Pitch content is not the only remarkable quality of these sequences. After all, we could easily construct random sequences with Gaussian distribution (see Musical Example 2) in which the pitch probability is weighed. The attractors from which these sequences are derived reveal a high degree structure, which must, in some way, be transferred to their musical transcription. Looking at the sequence from Hénon attractor, we discover that certain patterns of pitches recur:

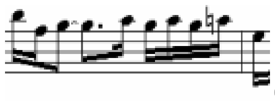

(End of measure 9, beginning of 10)

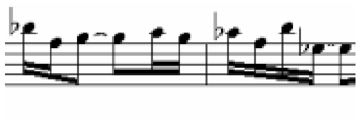

(End of m. 6, beginning of 7)


(End of m. 4, beginning of 5)


(End of m. 4, beginning of 5)

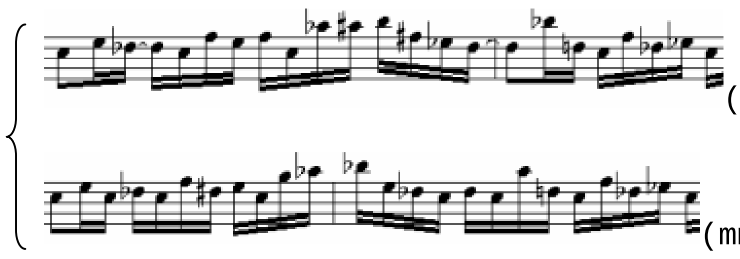

(m. 10)


(m. 1, beginning of m. 2)

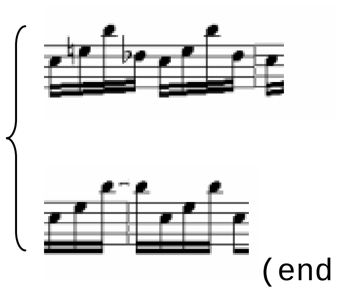

(End of m. 5, beginning of 6)

Observe that patterns are not repeated exactly the same. A few pitches may change, but their contour similar.

The sequence derived from Martin attractor also shows patterns of repeating pitches:

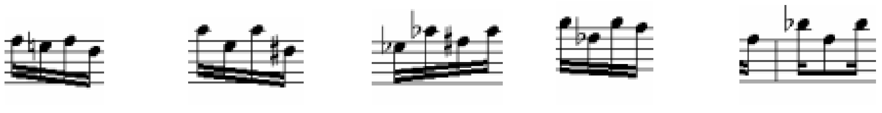

(mm. 1 and 2)

(mm. 9 and 10)


(end of m. 6)

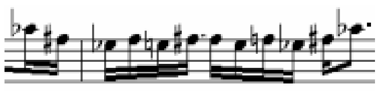
(end of m. 8 and beginning of 9)

And so does the Gingerbread man attractor sequence:



(mm. 9, 2, 3, 4, end of 3 and beginning of m. 4)

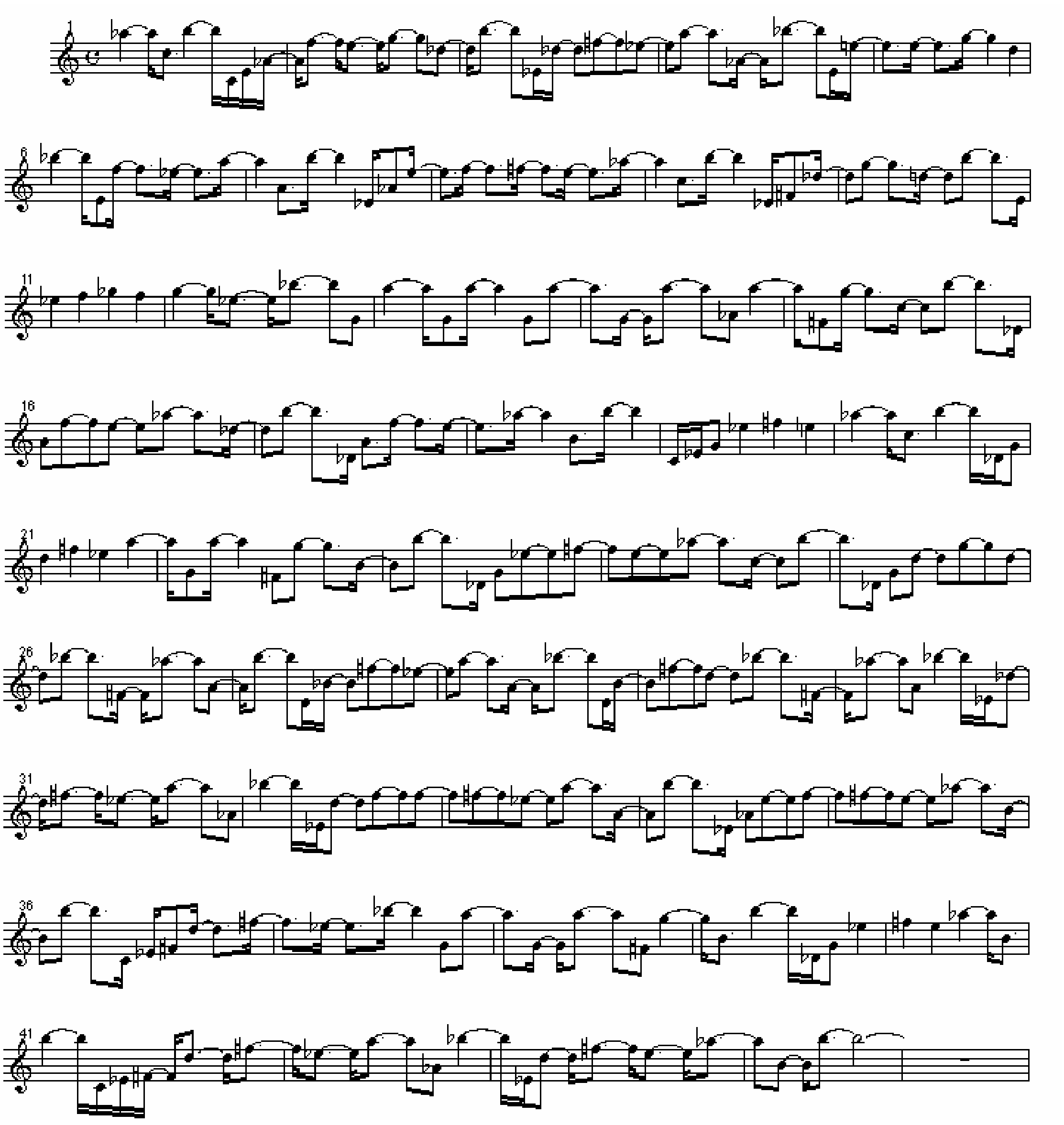
Note how these small cells appear in inversion, retrograde, augmentation, etc. There is even an instance of a non-retrogradable sequence of pitches!


(end of m. 5 and m. 6)

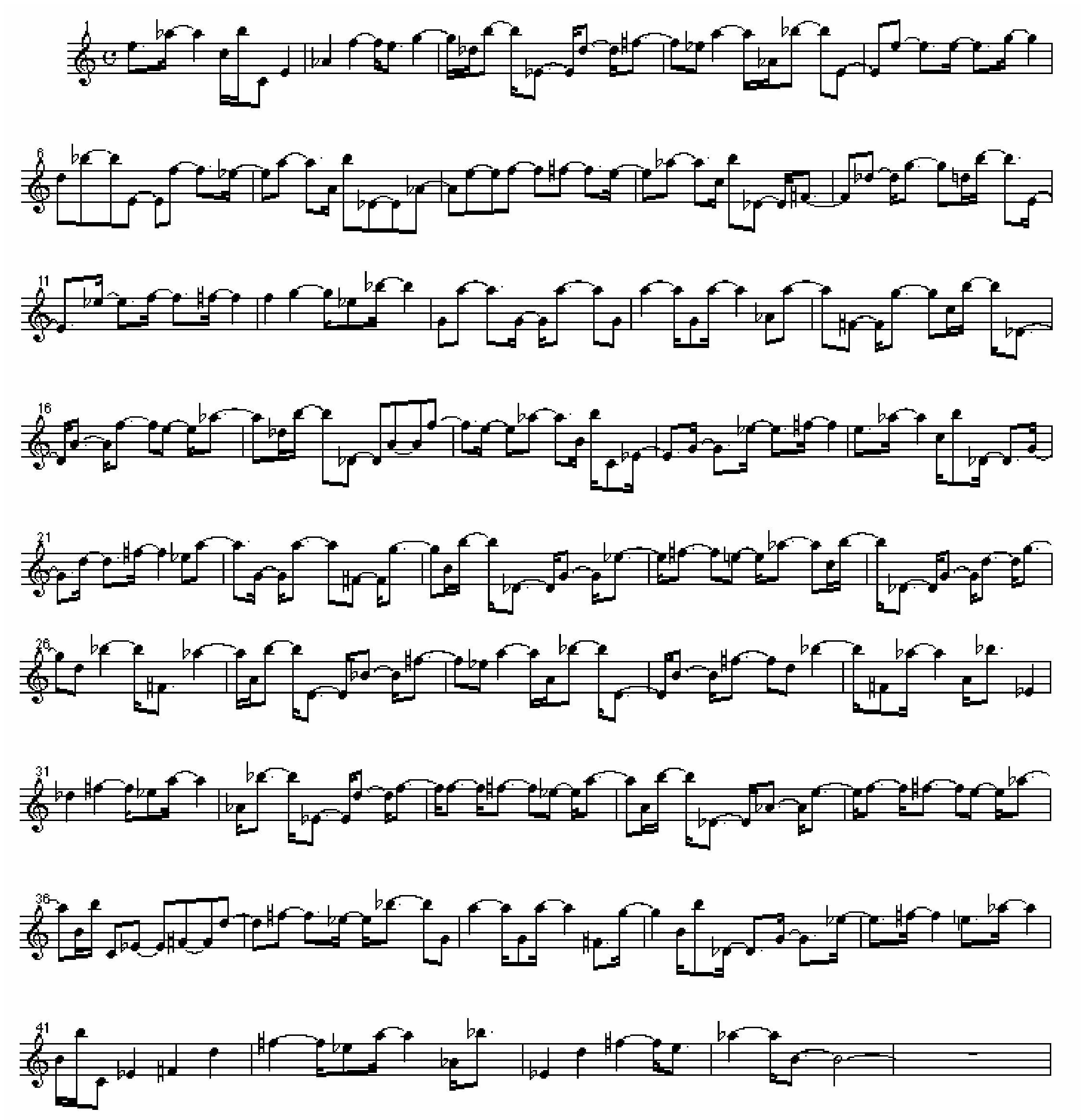
The intervalic relationship between pitches is also different in all three examples. In the sequence derived from Hénon attractor there seems to be a intertwining between small and large intervals (seconds and thirds and fourths and up.) This, in fact, resembles some of the visual representation of the attractor, which consists of lines separated at different distances. In Martin attractor there seems to be a predominance of larger intervals (fourths and sixths specially) creating thus a more disjunct melodic outline. In Gingerbread Man sequence the opposite holds true: smaller intervals predominate (minor and major seconds and minor thirds) and the melodic contour is more conjunct. In a purely random sequence all pitches, intervalic relationships would have the same probability of occurrence and no patterns will be evident. On the contrary, sequences derived from these chaotic attractors show structure and recognizable patterns.

Another way of mapping two-dimensional maps is to map each coordinate to a different musical parameter (such as pitch and rhythm.) We can map, for instance, the X coordinate to a pitch event space and the Y coordinate to a rhythmic set, or use the same coordinate to map *both* event sets.

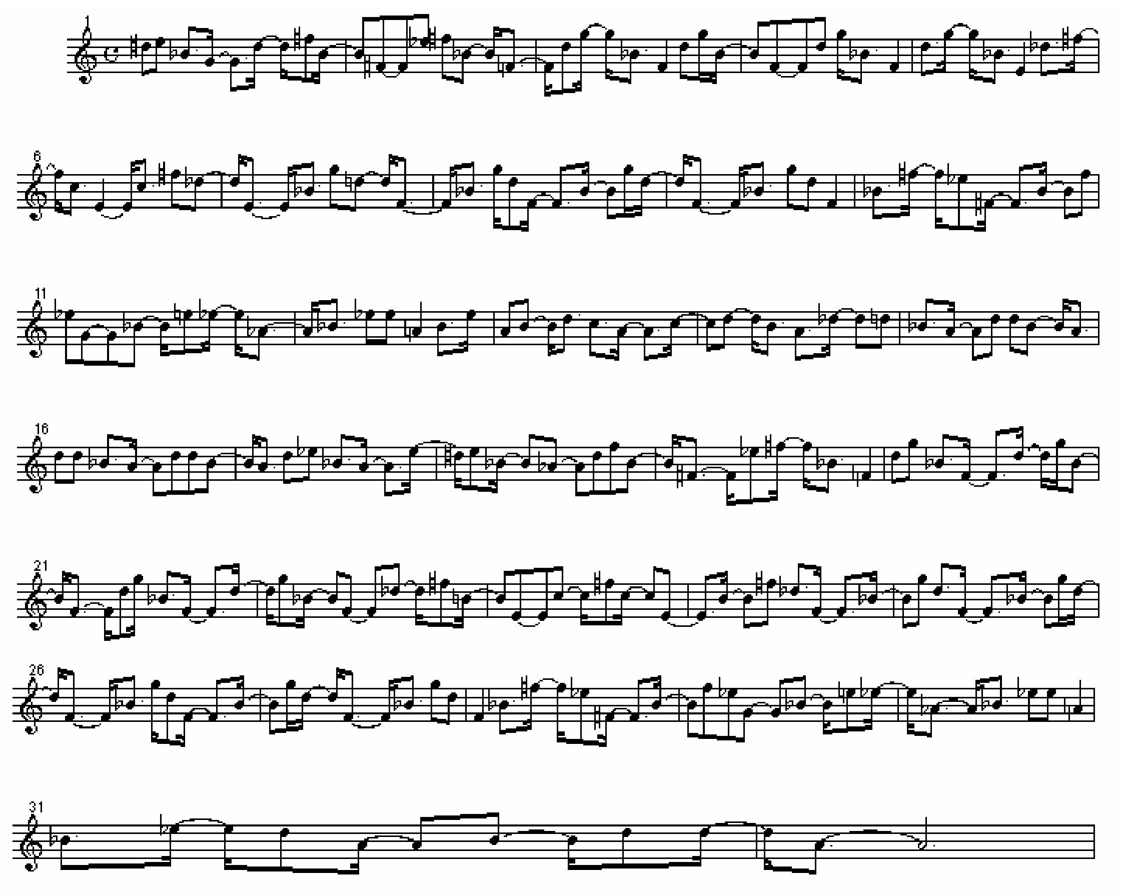
To demonstrate these alternative mappings, we will map the same three iterated maps onto two distinct event spaces (pitch and rhythm) using the values of the X and Y coordinates. Our pitch space will be a two-octave chromatic scale from C_4 to B_5 (24 elements) The rhythmic event space will consist of 5 values: {16th, 8th, dotted 8th, quarter, and quarter tied to 16th}. Normalized mapping was used this time. In order to apply this mapping type, the maximum and minimum values of X and Y were computed beforehand within a set of ten million iterations of the equations. The generated sequences follow:



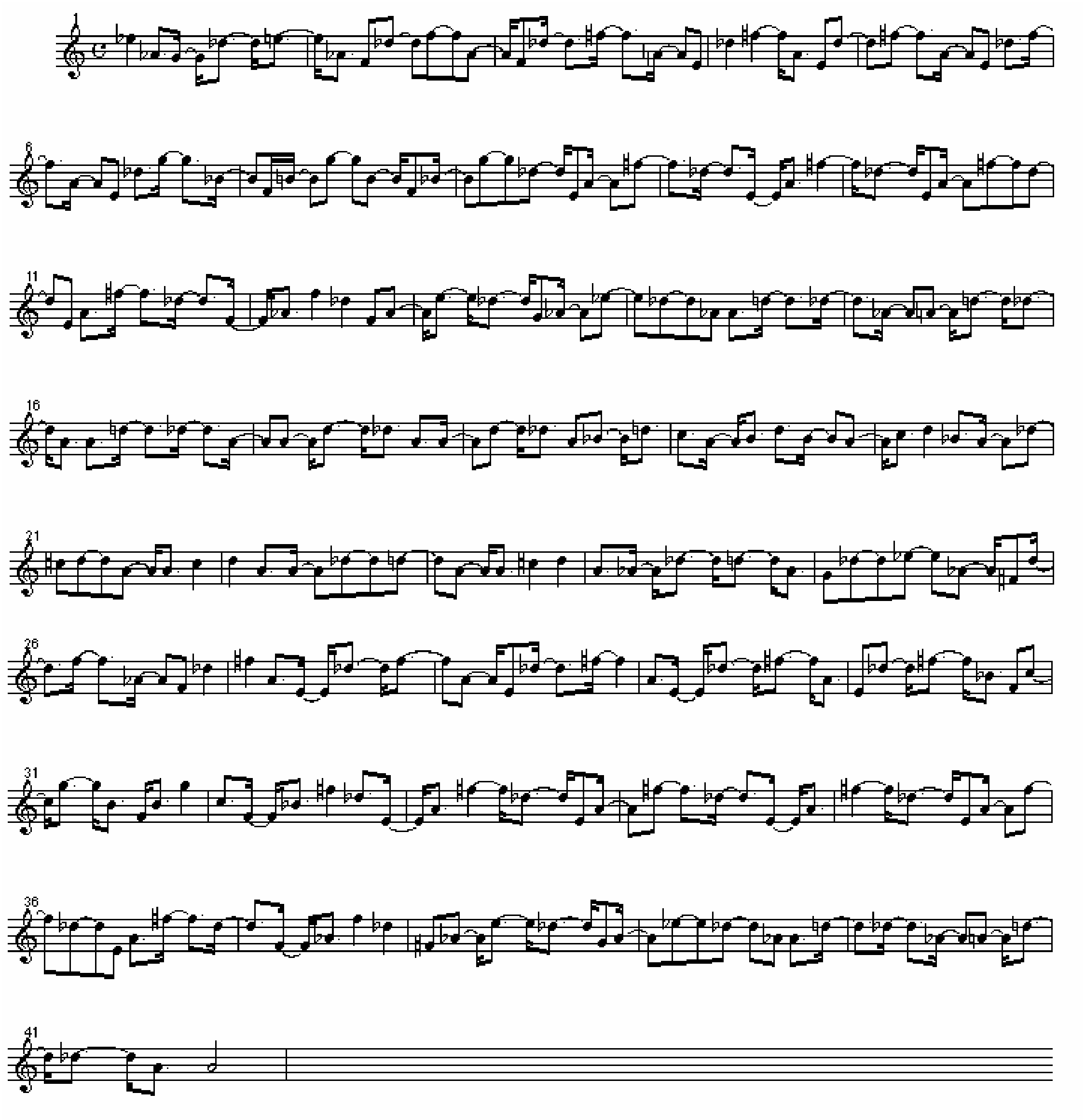
Musical Example 15. Music from the Hénon attractor. Pitch mapped to X coordinate. Duration to Y coordinate.



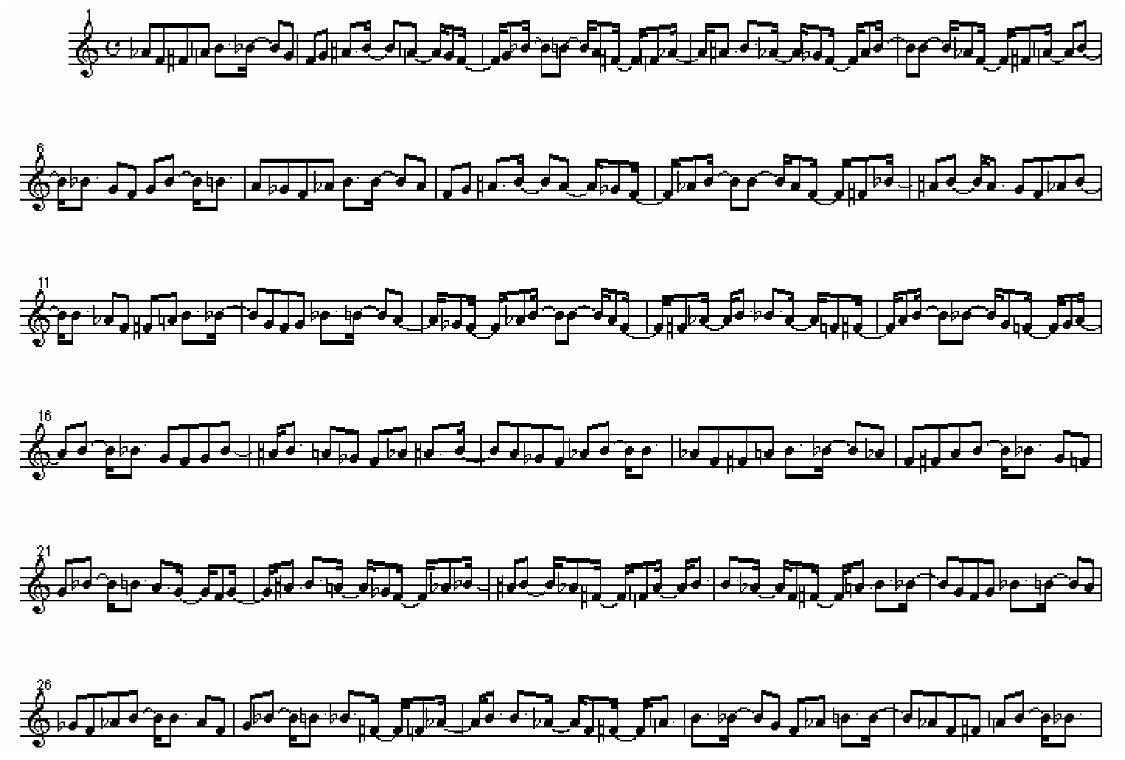
Musical Example 16. Music from Hénon attractor. Pitch mapped to Y coordinate. Duration to X coordinate.



Musical Example 17. Music from Martin attractor. Pitch mapped to X coordinate. Duration to Y coordinate.

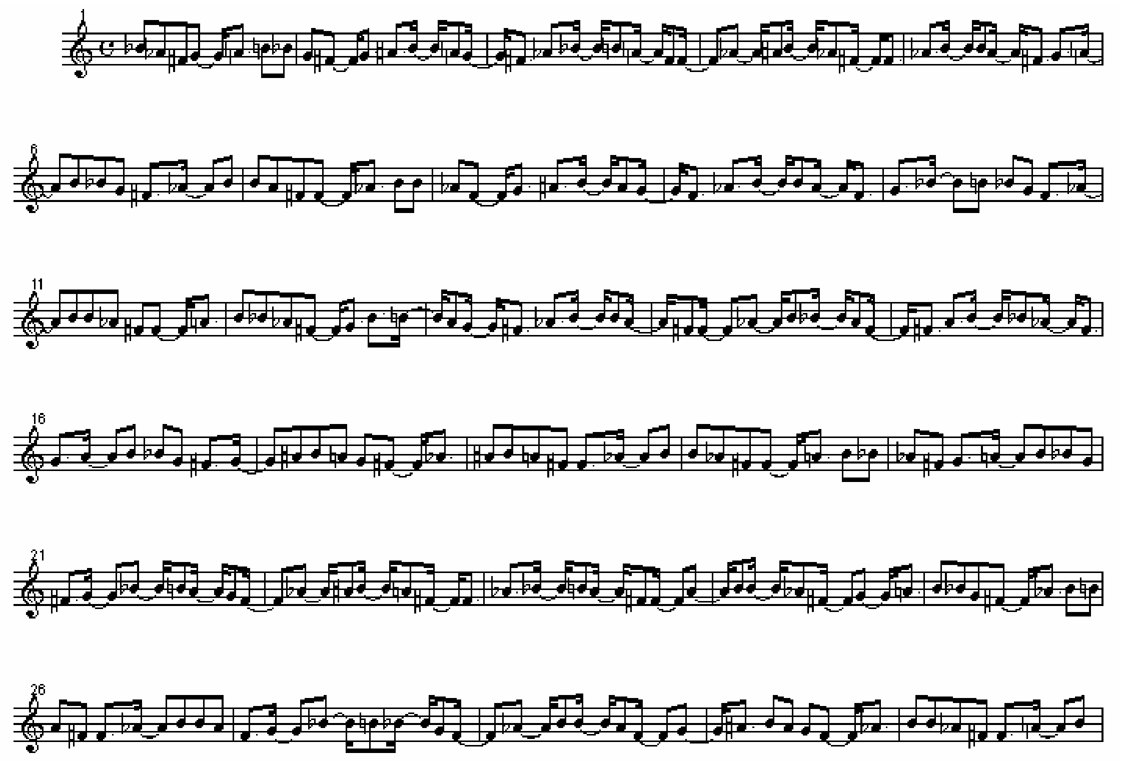


Musical Example 18. Music from Martin attractor. Pitch mapped to Y coordinate. Duration to X coordinate.



Musical Example 19. Music from Gingerbread attractor.

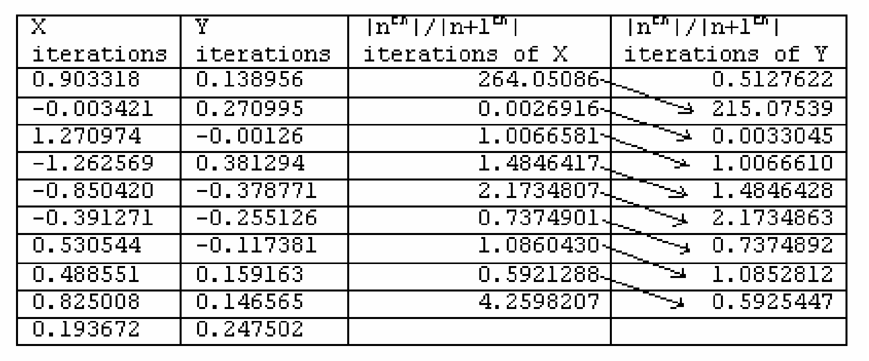
Pitch mapped to X coordinate. Duration to Y coordinate.


Musical Example 20. Music from Gingerbread attractor. Pitch mapped to X coordinate. Duration to Y coordinate.

Each attractor generates different sequences even though they are all mapped to the same event spaces. It is interesting to note that, within the same attractor, the two different mapping schemes (pitch mapped to X, duration to Y, and viceversa) generate very similar sequences. This is particularly evident in the sequences generated from Hénon attractor (Musical examples 15 and 16.) This attractor generates almost the same pitches and durations from both coordinates. Starting from the second pitch, Musical Example 16 has the same sequence of pitches than

Musical Example 15. Similarly, the durations are identical starting from the third value in Musical Example 15 and the second value in Musical Example 16. This is a consequence of the formulation of the equations, in which coordinate Y is just the previous value of X scaled by parameter *b.*

The following table of values from the Hénon attractor clarifies this point:



X iterations	Y iterations	$\|n^{th}\|/\|n+1^{th}\|$ iterations of X	$\|n^{th}\|/\|n+1^{th}\|$ iterations of Y
0.903318	0.138956	264.05086	0.5127622
-0.003421	0.270995	0.0026916	215.07539
1.270974	-0.00126	1.0066581	0.0033045
-1.262569	0.381294	1.4846417	1.0066610
-0.850420	-0.378771	2.1734807	1.4846428
-0.391271	-0.255126	0.7374901	2.1734863
0.530544	-0.117381	1.0860430	0.7374892
0.488551	0.159163	0.5921288	1.0852812
0.825008	0.146565	4.2598207	0.5925447
0.193672	0.247502		

Figure 14. Table of iteration values for Hénon attractor

The first two columns show the values of X and Y from ten successive iterations (from 100 to 109.) The next two columns show the result of dividing the absolute values of two successive iterations from each coordinate (iterations 100/101, 101/102, and so forth.) Note that the values connected by the arrows are almost identical. They differ by only one part in a thousand in average. This difference is a consequence of the rounding-off errors that inevitably accumulate from iteration to iteration in the computer calculations.

Mathematically speaking, the values of both X and Y have the same amount of *scaling*. A simpler example elucidates this concept further:

1	2	4	8	16	32	64	128	256	512	...
1.5	3	6	12	24	48	96	192	384	768	...

The above two series, although different at first sight, have the same amount of scaling. If we divide the nth term by the n+1th term in both series, we get the same values:

1/2 1/2 1/2 1/2 1/2 1/2 1/2 1/2 1/2 ...

The same is true for Martin and Gingerbread Man attractors, although the amount of scaling between X and Y coordinates differs from that of Hénon's.

A study of the number of occurrences of the elements in our event spaces (pitch and duration) reveals more about the structure of these sequences. The following tables were computed from a set of ten million iterations of the equations.

	Hénon Attractor			
	Pitch mapped to X coordinate. Duration to Y coordinate		Pitch mapped to Y coordinate. Duration to X coordinate	
Pitch class	No. of occurrences	Percentage (rounded to 2 decimals)	No. of occurrences	Percentage (rounded to 2 decimals)
C4	290370	2.90	290370	2.90
C#4	225702	2.26	225702	2.26
D4	224906	2.25	224906	2.25
D#4	306453	3.06	306453	3.06
E4	245163	2.45	245163	2.45
F4	201591	2.02	201591	2.02
F#4	289693	2.90	289693	2.90
G4	395089	3.95	395089	3.95
G#4	359682	3.60	359682	3.60
A4	278250	2.78	278250	2.78
A#4	290170	2.90	290170	2.90
B4	254934	2.55	254934	2.55
C5	235108	2.35	235108	2.35
C#5	341939	3.42	341939	3.42
D5	444934	4.45	444934	4.45
D#5	670898	6.71	670898	6.71
E5	570592	5.71	570592	5.71
F5	629198	6.29	629198	6.29
F#5	817871	8.18	817871	8.18
G5	448353	4.48	448353	4.48
G#5	464978	4.65	464978	4.65
A5	579705	5.80	579705	5.80
A#5	621368	6.21	621368	6.21
B5	813051	8.13	813051	8.13
Duration	No. of occurrences	Percentage (rounded to 2 decimals)	No. of occurrences	Percentage (rounded to 2 decimals)
𝅘𝅥𝅯	1250714	12.51	1250714	12.51
♪	1465888	14.66	1465888	14.66
♪.	1351970	13.52	1351970	13.52
♩	3100786	31.01	3100786	31.01
♩+𝅘𝅥𝅯	2830639	28.31	2830639	28.31

	Martin Attractor			
	Pitch mapped to X coordinate. Duration to Y coordinate		Pitch mapped to Y coordinate. Duration to X coordinate	
Pitch class	No. of occurrences	Percentage (rounded to 2 decimals)	No. of occurrences	Percentage (rounded to 2 decimals)
C4	30738	0.31	30725	0.31
C#4	79252	0.79	79230	0.79
D4	71072	0.71	71065	0.71
D#4	65800	0.66	65853	0.66
E4	273136	2.73	273143	2.73
F4	376221	3.76	376152	3.76
F#4	380298	3.80	380502	3.81
G4	815178	8.15	814963	8.15
G#4	964800	9.65	965123	9.65
A4	829554	8.30	830851	8.31
A#4	593017	5.93	591235	5.91
B4	520917	5.21	521173	5.21
C5	521173	5.21	520918	5.21
C#5	591235	5.91	593017	5.93
D5	830851	8.31	829554	8.30
D#5	965123	9.65	964800	9.65
E5	814963	8.15	815178	8.15
F5	380502	3.81	380298	3.80
F#5	376152	3.76	376221	3.76
G5	273143	2.73	273136	2.73
G#5	65853	0.66	65800	0.66
A5	71065	0.71	71072	0.71
A#5	79231	0.79	79252	0.79
B5	30724	0.31	30737	0.31
Duration	No. of occurrences	Percentage (rounded to 2 decimals)	No. of occurrences	Percentage (rounded to 2 decimals)
𝅘𝅥𝅯	455818	4.56	455761	4.56
𝅘𝅥𝅮	3152943	31.53	3152779	31.53
𝅘𝅥𝅮.	2782698	27.83	2782697	27.83
𝅘𝅥	3152779	31.53	3152943	31.53
𝅘𝅥+𝅘𝅥𝅯	455760	4.56	455818	4.56

	Gingerbread Man Attractor			
	Pitch mapped to X coordinate. Duration to Y coordinate		Pitch mapped to Y coordinate. Duration to X coordinate	
Pitch class	No. of occurrences	Percentage (rounded to 2 decimals)	No. of occurrences	Percentage (rounded to 2 decimals)
C4	160810	1.61	160810	1.61
C#4	474314	4.74	474314	4.74
D4	555051	5.55	555051	5.55
D#4	433238	4.33	433238	4.33
E4	372033	3.72	372033	3.72
F4	463095	4.63	463095	4.63
F#4	598465	5.98	598465	5.98
G4	583230	5.83	583230	5.83
G#4	489283	4.89	489283	4.89
A4	538638	5.39	538638	5.39
A#4	631876	6.32	631876	6.32
B4	570111	5.70	570111	5.70
C5	571833	5.72	571833	5.72
C#5	637424	6.37	637424	6.37
D5	799149	7.99	799149	7.99
D#5	812399	8.12	812399	8.12
E5	467700	4.68	467700	4.68
F5	149945	1.50	149945	1.50
F#5	96399	0.96	96399	0.96
G5	82186	0.82	82186	0.82
G#5	106832	1.07	106832	1.07
A5	140480	1.40	140480	1.40
A#5	140680	1.41	140680	1.41
B5	124663	1.25	124663	1.25
Duration	No. of occurrences	Percentage (rounded to 2 decimals)	No. of occurrences	Percentage (rounded to 2 decimals)
𝅘𝅥𝅯	1915379	19.15	1915379	19.15
♪	2525412	25.25	2525412	25.25
♪.	2937546	29.38	2937546	29.38
♩	2043623	20.44	2043623	20.44
♩+𝅘𝅥𝅯	577874	5.78	577874	5.78

It is obvious from these tables that some events (pitches and durations) predominate over others. In Hénon attractor, for instance, pitch-classes F#$_5$ and B$_5$, followed by D#$_5$, F$_5$, and A#$_5$ have a much higher rate of occurrence than, say, C_4 or F_4. Likewise, quarters and quarters tied to sixteenths occur more often than the rest of durations. Note that both mapping schemes (pitch mapped to X, duration to Y, and viceversa) generate the same occurrence values. This is a consequence of the scaling between the values of X and Y, as explained earlier.

The differences in event occurrence is even more dramatic in Martin attractor. Here, pitch-classes G_4, $G\#_4$, A_4, D_5, $D\#_5$, and E_5 account for 52.21% of all values. Furthermore, the distribution of values is structured with striking symmetry, as is evident in the following graph (percentage values were rounded to the nearest integer):

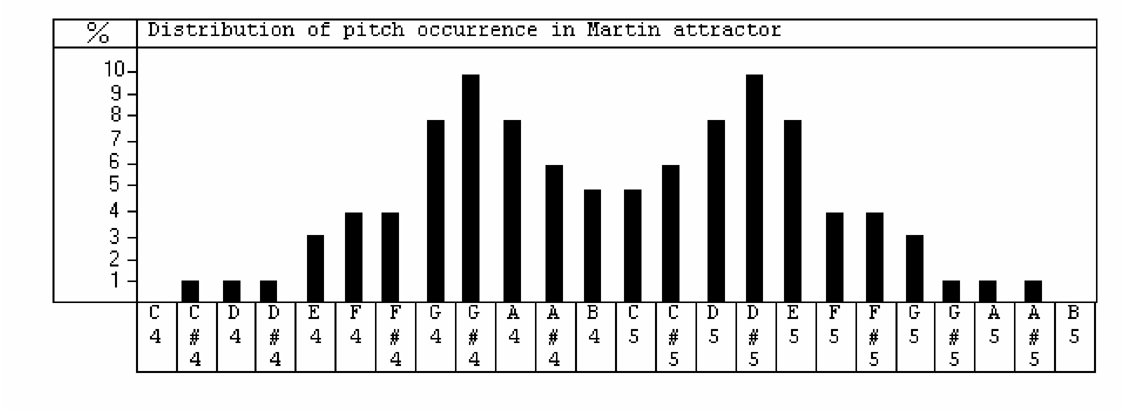


Figure 15. Pitch distribution in Martin attractor

This is a reflection of the attractor's perfectly symmetrical shape (see Fig. 13.) The same is true for durations. As with Hénon attractor, both mapping schemes generate the same occurrence values.

Lastly, Gingerbread Man attractor's occurrence values also differ from those of Martin and Hénon attractors. The bulk of pitch occurrence is distributed between pitch classes $C\#_4$-$D\#_5$, with a much lower frequency of occurrence for pitch-classes C_4 and F_5-B_5. Duration occurrence also shows the same pattern: eighths, dotted eighths and quarters have a higher frequency of occurrence than sixteenths and quarters tied to sixteenths. The Gingerbread Man attractor also produces the same occurrence values from both mappings, again a consequence of the scaling of the values of successive iterations of the X and Y coordinates.

A rearrangement of the occurrence values reveals they are arranged according to the Gaussian probability distribution, as in Musical Examples 12-14:

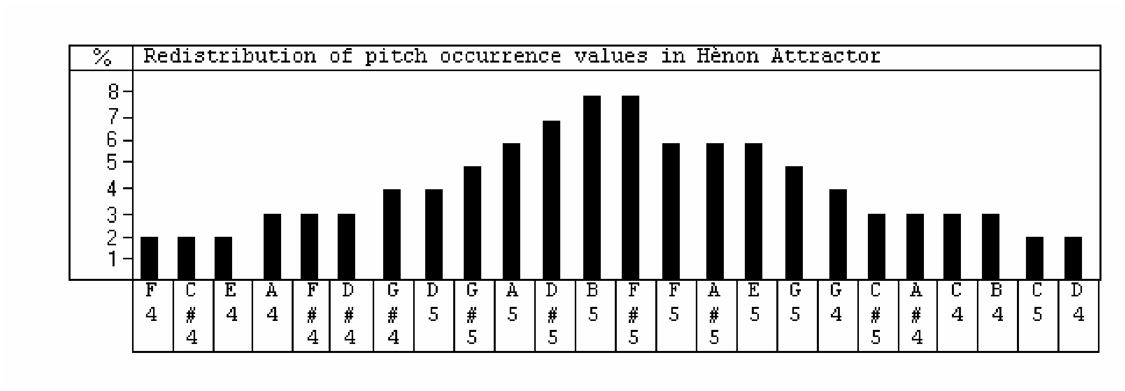


Figure 16. Pitch distribution in Hénon attractor

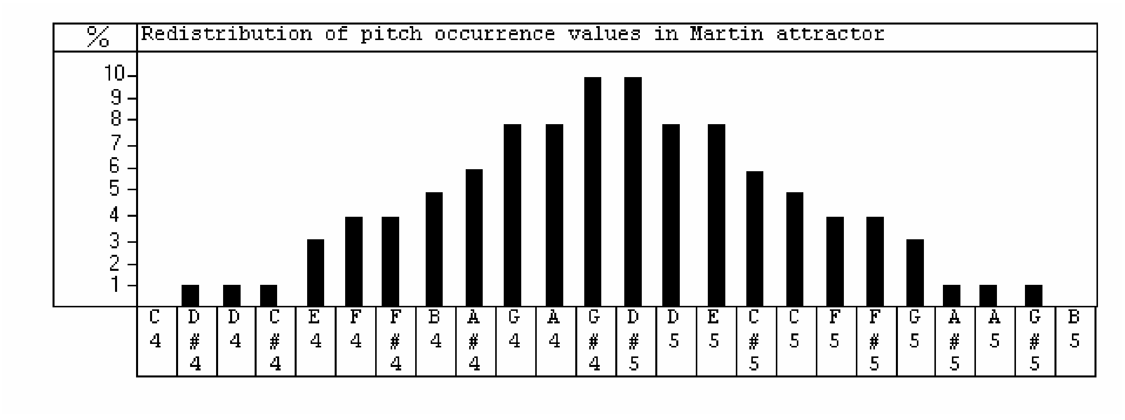


Figure 17. Pitch distribution in Martin attractor

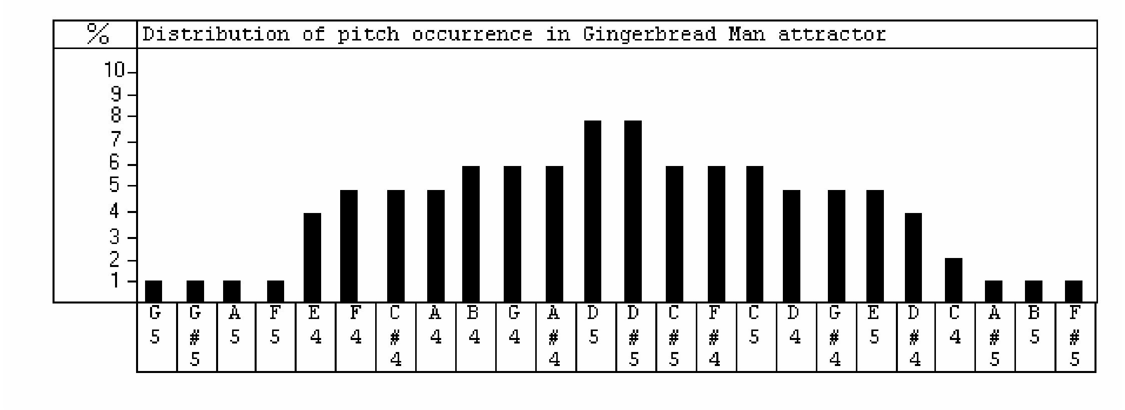


Figure 18. Pitch distribution in Gingerbread Man attractor

The percentage values in the above graphs were rounded to the nearest integer. This makes the contour of graphs seem more bumpy than they really are. Although only pitch occurrence values are shown, duration values also follow the same distribution.

It is important to realize that these event occurrences values may not be immediately obvious in the generated sequences (Musical Examples 15-20.) The reason is that only 200 iterations of the equations were mapped. This sample space is to small to show the overall behavior of the attractors. However, they are sufficient to demonstrate the structures these attractors yield, as well as the differences among them.

Attractors from Three-dimensional Chaotic Maps

Three-dimensional attractors exist in three-dimensional space. Their equations have, consequently, three variables. Each point in the attractor is expressed by three coordinates (X, Y, Z.) Each of these coordinates can be mapped to a different musical event space (such as pitch, rhythm and dynamic) in the same manner as with two-dimensional attractors. As an example, we will study two three-dimensional attractors: Rössler attractor, and Pickover attractor.

Pickover attractor is an iterated map in three-dimensions proposed by Clifford A. Pickover.[17] It is formulated by the following equations:

$$x_{n+1} = \sin(a*y_n) - z_n*\cos(b*x_n)$$
$$y_{n+1} = z_n*\sin(c*x_n) - \cos(d*y_n)$$
$$z_{n+1} = \sin(x_n)$$

where *a*, *b*, *c* and *d* are the controlling parameters. Default values for these parameters are

$$a = 2.24;\quad b = 0.43;\quad c = -0.65;\quad d = -2.43$$

When plotted, this function generates the following attractor:

[17]C. A. Pickover, "Million-Point Sculptures," *Computer Graphics Forum* 10(4) (1991): 333-336.

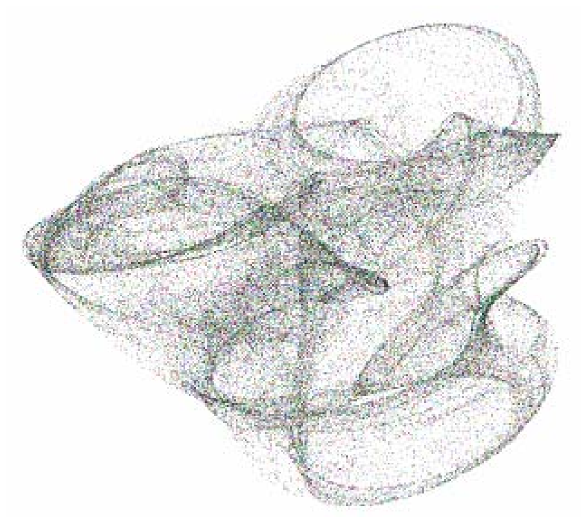
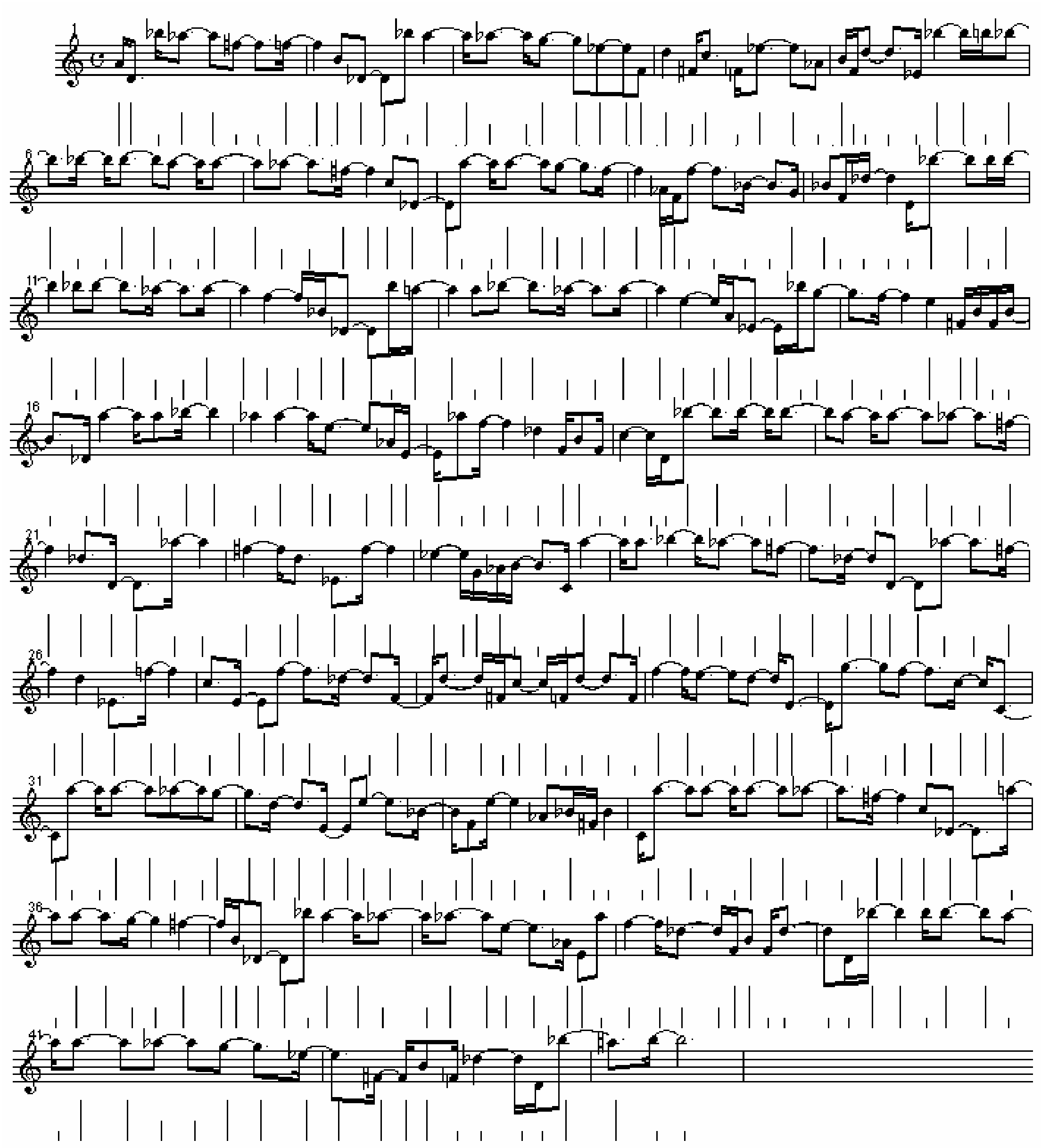


Figure 19. Pickover attractor

Unfortunately, one dimension is lost as the image is printed in the two-dimensional surface of paper.

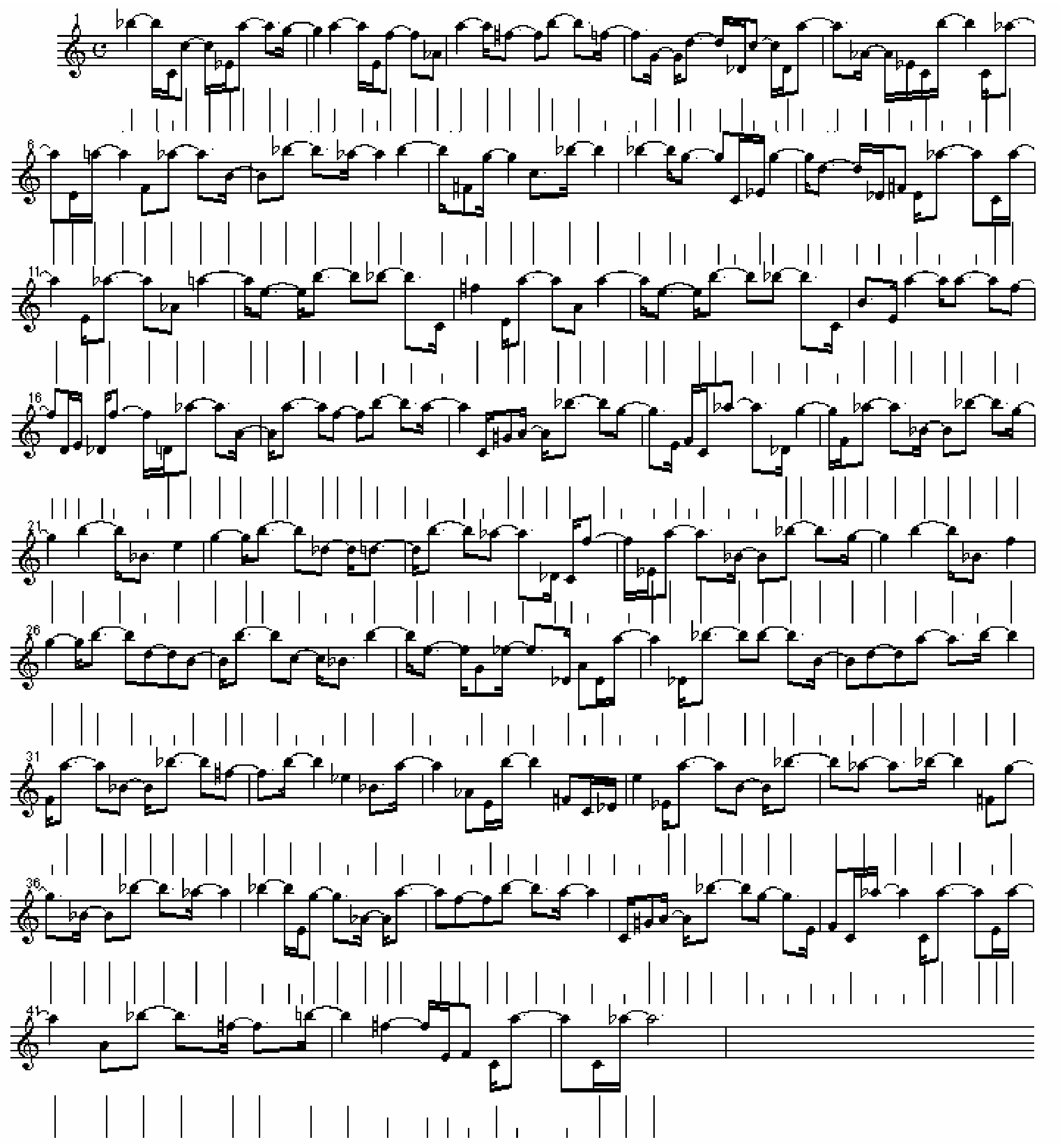
We will map each coordinate X, Y, Z to three different musical event sets: a pitch set (a two-octave chromatic scale from C_4 to B_5,) a rhythmic set, {16th, 8th, dotted 8th, quarter, and quarter dotted to 16th}, and a dynamic set, {*ff*, *mf*, *mp*, *pp*}. In the following mapping scheme, pitch was mapped to the Y coordinate, rhythm to the X coordinate and dynamic to the Z coordinate. Normalized mapping was employed. As with two-dimensional attractors, the maximum and minimum values of X, Y, and Z were precomputed from a set of ten million iterations of the equations. The resulting sequence is as follow:



Musical Example 21. Music from Pickover attractor. Pitch mapped to Y coordinate, duration to Y coordinate, dynamic to Z coordinate.

Dynamics are represented by four lines of varying length, shown immediately below the notes. The longest corresponds to *ff*, the shortest to *pp*.

By assigning different coordinates to the event spaces, different mappings can be applied to the same attractor. In the following mapping of the same attractor, pitch was assigned to the X coordinate, rhythm to Z, and dynamic to Y:



Musical Example 22. Music from Pickover attractor. Pitch mapped to X coordinate, duration to Z coordinate, dynamic to Y coordinate.

What follows is a table of event occurrence values of both mapping schemes.

	Pickover Attractor			
	Pitch mapped to Y coordinate. Duration to X coordinate. Dynamic to Z coordinate		Pitch mapped to X coordinate. Duration to Z coordinate. Dynamic to Y coordinate	
Pitch class	No. of occurrences	Percentage (rounded to 2 decimals)	No. of occurrences	Percentage (rounded to 2 decimals)
C4	241536	2.42	911628	9.12
C#4	297092	2.97	449760	4.50
D4	246241	2.46	392659	3.93
D#4	212484	2.12	295560	2.96
E4	267236	2.67	310222	3.10
F4	442227	4.42	256252	2.56
F#4	367935	3.68	239589	2.40
G4	325578	3.26	230758	2.31
G#4	312117	3.12	256644	2.57
A4	289991	2.90	249634	2.50
A#4	286410	2.86	222981	2.23
B4	293875	2.94	216693	2.17
C5	310055	3.10	230890	2.31
C#5	324552	3.25	230738	2.31
D5	372221	3.72	228202	2.28
D#5	339170	3.39	247993	2.48
E5	392788	3.93	257538	2.58
F5	428800	4.29	273819	2.74
F#5	469714	4.70	313475	3.13
G5	633861	6.34	397912	3.98
G#5	905144	9.05	574074	5.74
A5	1067178	10.67	989102	9.89
A#5	893675	8.94	750991	7.51
B5	280118	2.80	1472884	14.73
Duration	No. of occurrences	Percentage (rounded to 2 decimals)	No. of occurrences	Percentage (rounded to 2 decimals)

𝅘𝅥𝅯	2303250	23.03	2530043	25.30
♪	1188484	11.88	1055368	10.55
♪.	1094888	10.95	915281	9.15
♩	1313511	13.14	1098845	10.99
♩+𝅘𝅥𝅯	4099865	41.00	4400461	44.00
Dynamic	No. of occurrences	Percentage (rounded to 2 decimals)	No. of occurrences	Percentage (rounded to 2 decimals)
ff	2806789	28.07	1706816	17.07
mf	1225591	12.26	1875906	18.76
p	1253313	12.53	2167586	21.68
pp	4714305	47.14	4249690	42.50

Note that, as opposed to the mappings from Hénon, Martin, and Gingerbread attractors discussed earlier, in Pickover attractor different coordinates do not produce the same occurrence values. Pickover attractor also lacks the symmetry that these attractors have (see Fig. 19.) This does not mean that the musical mappings from Pickover attractor are devoid of structure. It is clear from Musical Examples 21 and 22 that Pickover attractor yields highly organized sequences. Pitch in Musical Example 21, for instance, appears to be structured by repetitions of the following descending pattern:

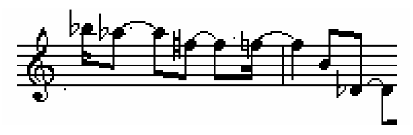


Figure 20. Detail from measure 1 in Musical Example 21

These patterns, however, never repeat exactly the same.

The following two graphs reveal that the distribution of event occurrence values in Pickover attractor——like in Hénon, Martin and Gingerbread Man attractors——is also Gaussian (only pitch occurrence distribution is shown):

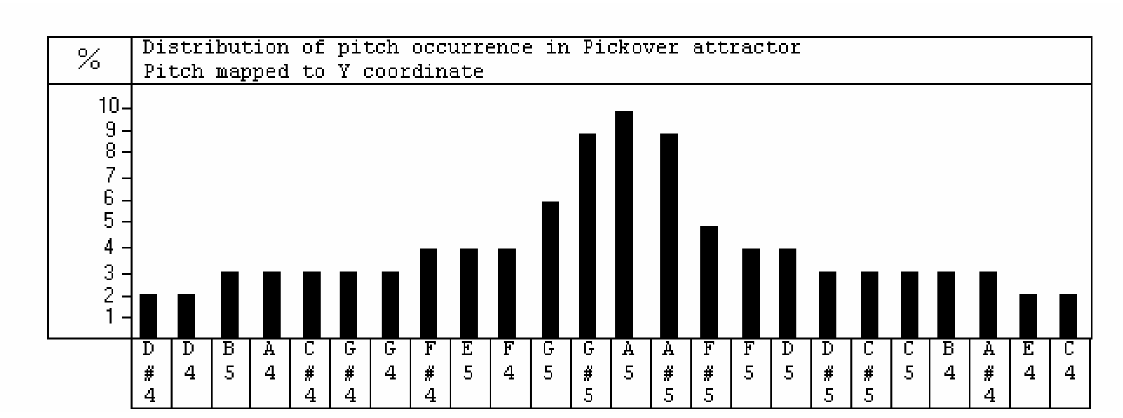


Figure 21. Pitch distribution in Pickover attractor (Y coordinate)

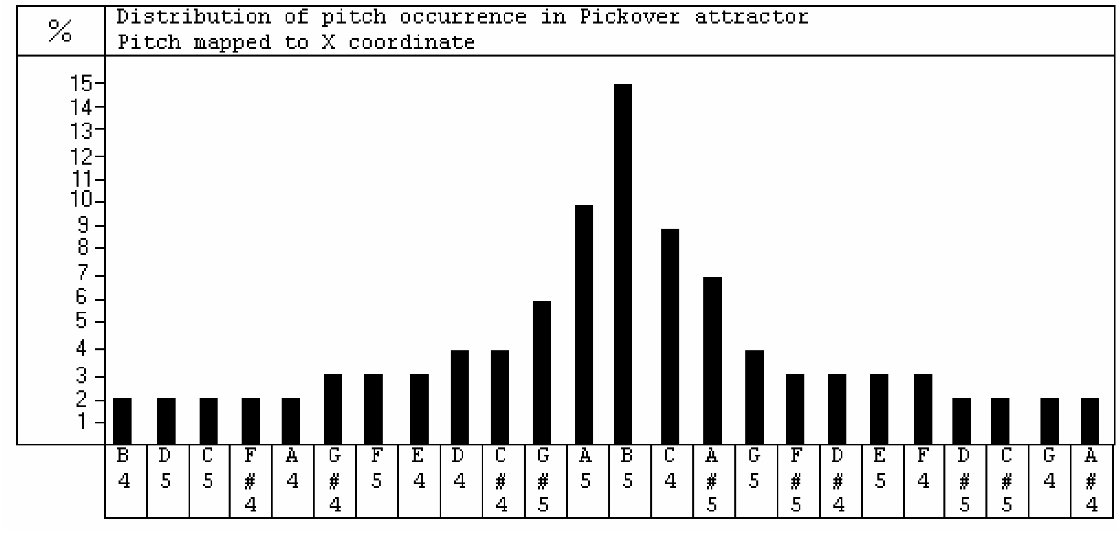


Figure 22. Pitch distribution in Pickover attractor (X coordinate)

The Rössler attractor, proposed by Otto E. Rössler[18], is a continuous flow in three dimensions. It is closely related to the Lorenz attractor, a mathematical model of a weather system developed by Edward Lorenz at the MIT.[19] The Rössler attractor is defined by the following differential equations:

$$x_{n+1} = x_n - y_n * dt - z_n * dt$$

$$y_{n+1} = y_n + x_n * dt + a * y_n * dt$$

$$z_{n+1} = z_n + b * dt + x_n * z_n * dt - c * z_n * dt$$

[18]H. O. Peitgen, ed., *Chaos and Fractals: New Frontiers of Science* (New York: Springer, 1992), 686-696.

[19]Lorenz, "Deterministic non-periodic flow," 130-141.

$x_0 = y_0 = z_0 = 1;$

$dt = 0.04;\ a = 0.2;\ b = 0.2;\ c = 5.7$

Although continuous flow systems like this one are characterized by continuous, unbroken orbits, this fact must be approximated in the computer by orbits of discrete points separated in time by a small amount, dt. This is achieved by solving the difference equations (actually transforming them into difference equations) by a method known as *numerical integration*.[20]

When plotted, Rössler attractor looks as follows:

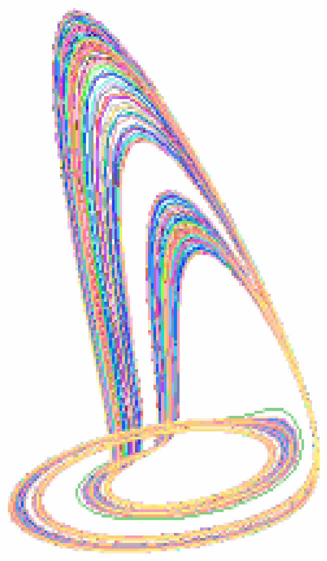
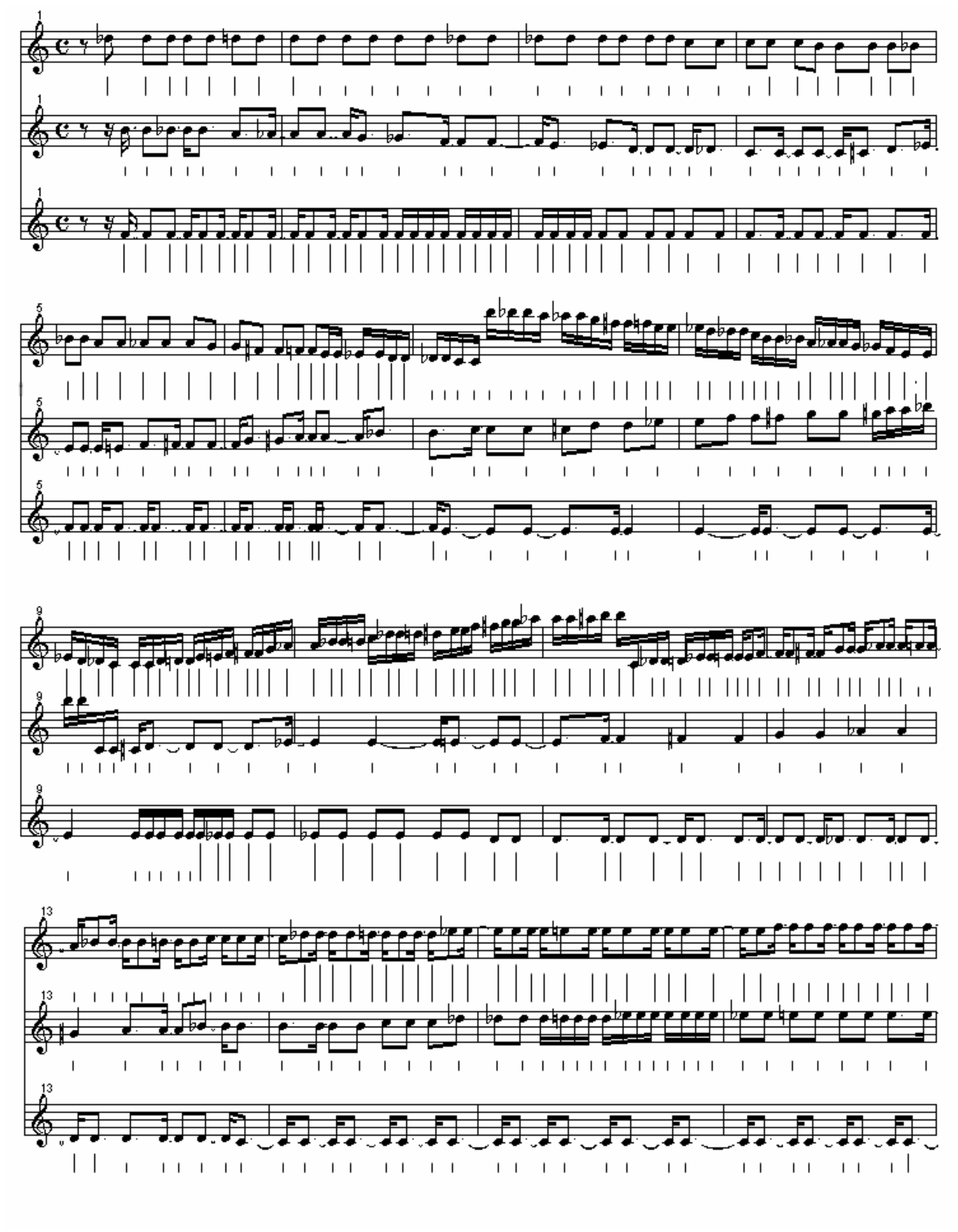


Figure 23. Rössler attractor

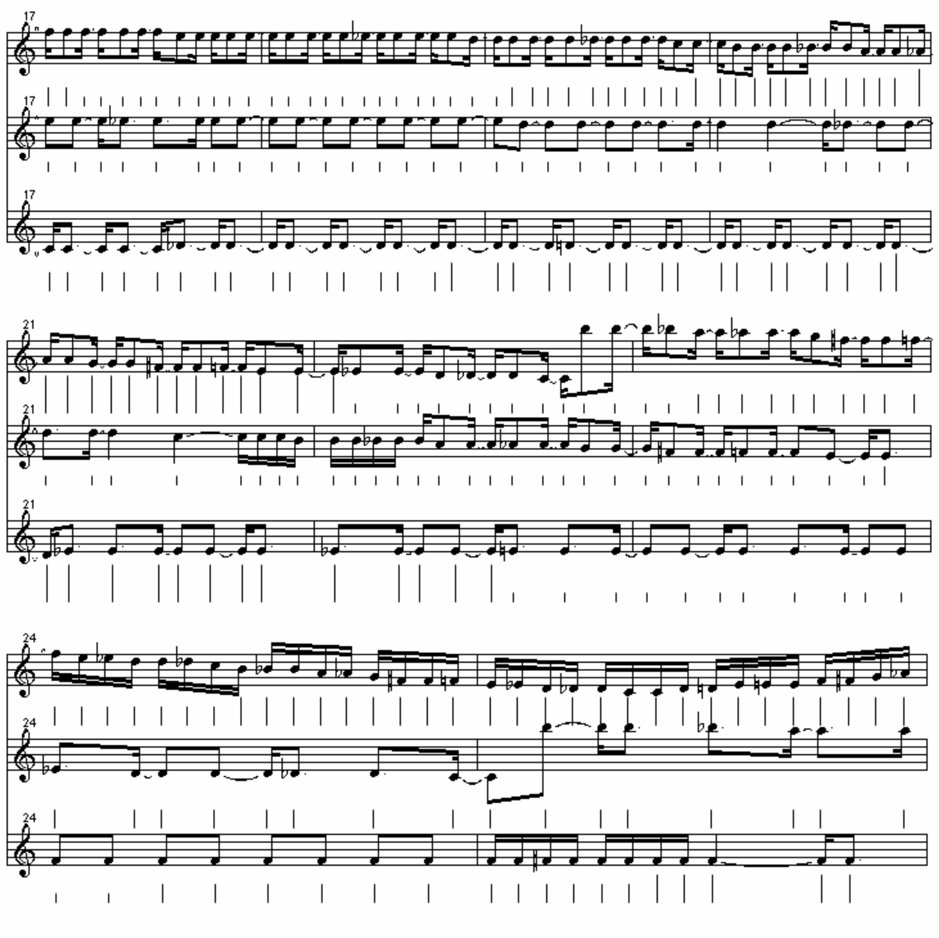
[20]Bidlack, "Chaotic Systems as Simple (but Complex) Compositional Algorithms," 40.

So far, we have been mapping all parameters to a single voice. We can also map each coordinate to a different voice, thus creating a polyphonic texture (this technique can also be applied to two-dimensional attractors.) The musical transcription of the Rössler attractor maps each coordinate (X, Y, Z) to three different event sets (as in the Pickover attractor above) in three different voices. In voice one, pitch is mapped to the X coordinate, rhythm to Y and dynamic to Z; in voice two, pitch is mapped Y, rhythm to Z and dynamic to X; voice three maps pitch to Z, rhythm to X and dynamic to Y.

The outcome is as follows:



Musical Example 23. Music from Rössler attractor



Musical Example 23 (continued)

Note how the continuous nature of the orbits of the attractor are realized musically as scales going up and down at various speeds and dynamics. Polyphonic mappings produce an aural sensation, which reflects more closely the spatial nature of these attractors.

Higher dimensional systems offer more degrees of freedom in the mapping process. A four-dimensional chaotic system, such as the Hénon-Heiles system[21], provides four variables which can be mapped, for instance, onto four different event sets, such as pitch, duration, dynamic, and timbre. The manner in which these higher-dimensional systems are mapped does not differ from their lower-dimensional siblings.

Another kind of dynamical systems, called *iterated function systems* (IFS,) developed by M. Barnsley[22], has also been proven to be a very fruitful source for algorithmic composition.[23]

The music generated by the mapping of dynamical systems can be varied by changing their equations' controlling parameters. Two mappings generated from the same system but with a tiny difference in one parameter, will in the long run generate different sequences, since the system is very sensible to its initial condition: that is the essence of chaos. Even differences in the computer implementation (programming) of the system's equations (using single-precision floating-point variables instead

[21]Ibid., 41.

[22]M. Barnsley, *Fractals Everywhere* (New York: Academic Press, 1988).

[23]M. Gogins, "Iterated Function Systems Music," *Computer Music Journal* 15(1) (1991): 40-48.

double-precision, for instance) will generate different sequences from the same initial conditions.

Fractals

The word "fractal" was coined by the Polish mathematician Benoit Mandelbrot[24], from the Latin word *fractus*, meaning fractional, or fragmented. Fractals and fractal geometry[25] were born in an effort towards developing a mathematical framework to understand the way the Nature uses and reuses the same forms redundantly in both living and non-living things. Traditional geometry has always dealt with regular forms and smooth curves (those that can be differentiated.) Forms in nature, however, can not be described faithfully with traditional geometry. As Mandelbrot puts it, "clouds are not spheres, mountains are not cones, coastlines are not circles and bark is not smooth, nor does lightning travel in a straight line."[26]

[24]Briggs, *Turbulent Mirror*, 90.

[25]B. Mandelbrot, *The Fractal Geometry of Nature* (San Francisco: W.H. Freeman, 1982).

[26]Briggs, *Turbulent Mirror*, 90.

An essential characteristic of many fractals objects is that they manifest *self-similarity* at all scales: a smaller portion of the whole object looks like the whole.

Self-similarity comes in two types: *exact*, in which magnified small parts of the object in question are identical to the whole; and *statistical*, in which a magnified portion of the object has the same statistical properties as the whole. Some fractal objects, however, such as the famous Mandelbrot Set, are not self-similar.

Fractals are both natural and mathematical objects. Fractals abound in nature; for instance, the branching of a tree has fractal structure: individual branches look like scaled-down versions of the whole tree; the same is true for the heads of cruciferous vegetables such as broccoli and cauliflower, to say for the vascular systems and lung structures of animals. Self-similarity in natural fractals is always statistical: there is no exact self-similarity in nature.

Mathematical fractals——the only ones suitable for algorithmic composition——are also governed by nonlinear equations. One would expect that the generation of such complex shapes would require complex equations as well, but in reality, they are extraordinarily simple. Since the discovery of fractals and the birth of chaos science, thousands of fractal equations have been proposed, all with their own peculiarities and characteristics. Of these, we will concentrate in the first and most famous of all: the Mandelbrot Set.

The Mandelbrot Set fractal, dubbed "the most complex object in mathematics,"[27] is formulated by this exceedingly simple iterated map:

$$Z_{n+1} = (Z_n)^2 + C,$$

where both Z and C are complex[28] variables. For each C in the complex plane, Z_0 is set to 0 and the equation is iterated. There are two possible outcomes for Z:

1. $Z_n \to \infty$ as $n \to \infty$
2. Z_n remains bounded (finite) as $n \to \infty$.

To create the fractal image, each point C is assigned a color according to the number of iterations required to send the point to infinity, or a default color (generally black) if the point remains fixed. For practical purposes, a threshold value *t* and maximum number of iterations *m* are

[27]Ibid.

[28]A complex number has the form a + bi, where a and b are any real numbers and i is the imaginary unit ($\sqrt{-1}$).

defined. The iterative process is stopped when $|Z_n| \geq t$, or $n > m$. The Mandelbrot set is defined as the set of all values of Z which do not escape to infinity.

Here is a plotting of the Mandelbrot Set:

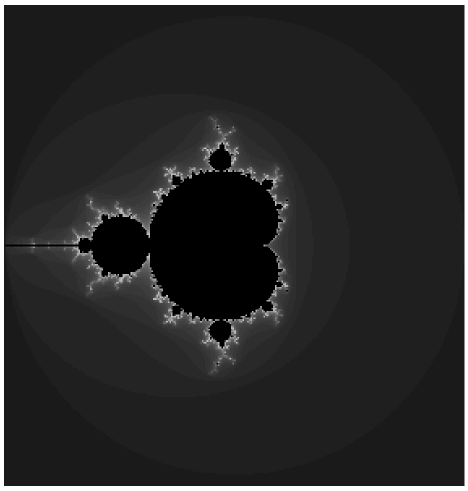


Figure 24. The Mandelbrot Set

Because between any to complex numbers there is always another complex number, we can perform mathematical "zooms" on the fractal, revealing its complexity and infinitely detailed structure. What follows are six successive zooms on the Mandelbrot Set:

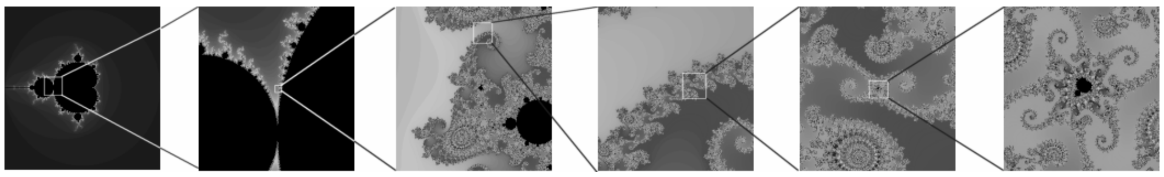


Figure 25. Mandelbrot Set zooms

It is an arresting thought to realize that these images, so aesthetically pleasing, are derived from such a simple procedure, and are, in essence, just a set of numbers.

Mapping the Mandelbrot Set to music turns out to be more complicated than mapping chaotic attractors. Each iteration of the equations defining a chaotic system corresponds to a point in the corresponding system's chaotic attractor. It is generally sufficient to plot a few hundred iterations to perceive the shape of the attractor. Mapping chaotic attractors onto music consists in mapping each iteration of the equations to a musical event space. On the other hand, in a fractal image, for *every* point of the image, its generating equation must be iterated a number of times (usually thousands) to decide whether the point escapes to infinity or not. Calculations in fractals are therefore increased a thousandfold.

A method to map fractals is to calculate the iteration values of the set of points in a one-dimensional slice within the two-dimensional structure of the fractal, and map those values onto a musical event space. This is performed as follows:

- Take any two points within the fractal image, a starting point (x, y) and an end point (Δx, Δy.)
- Calculate the iteration values for points that lay in the segment from (x, y) to (Δx, Δy,) linearly (the more points we take, the higher the "resolution" of the slice.)
- Map the iteration values of these set of points onto a musical event space.

As an example, we will map a slice from the Mandelbrot set. Our event space, for simplicity, will be a four octave chromatic scale (from C_2 to B_5——48 pitches——, ordered from 0 to 47) plus silence for the points that do not escape to infinity (the black regions on the Set.) We will choose sixteenths as note values. Additionally, if two consecutive points have the same iteration value, their mapped pitches are tied. Iteration limit is set to 1000. The segment will be divided in 600 points. Visually, we will map the following segment on the Mandelbrot set:

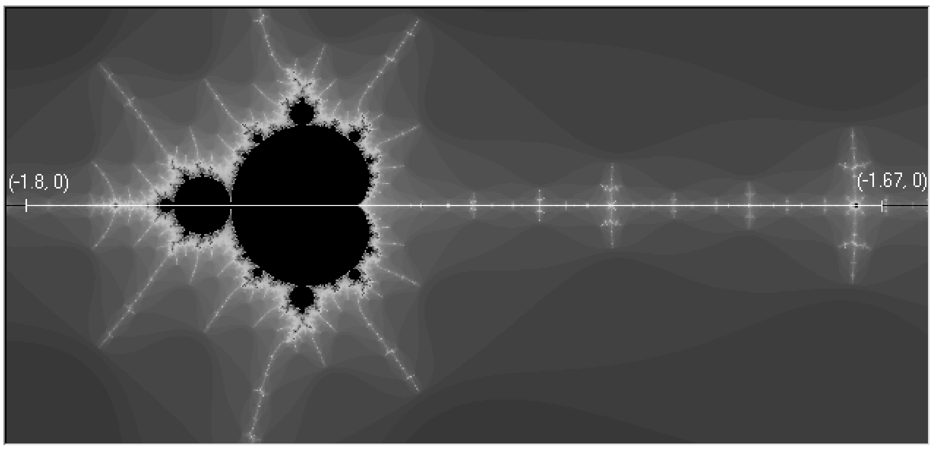
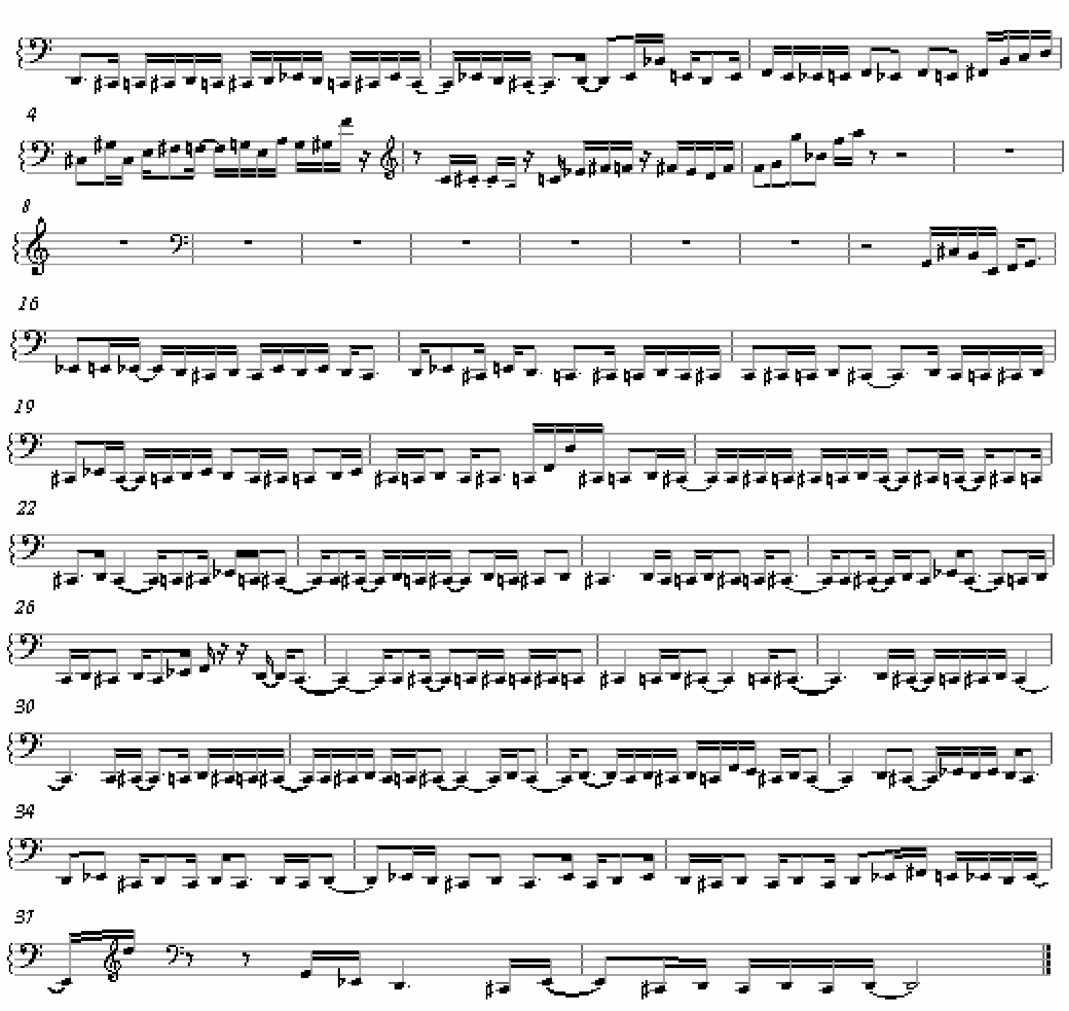


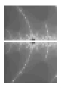
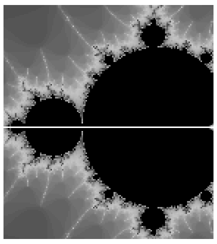
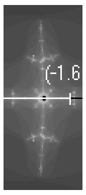
Figure 26. Mandelbrot Set mapped section

The mapped segment goes from (-1.8, 0) to (-1.67, 0.) Iteration values are mapped to our 48-element event set using the normalized mapping method. This assures that the behavior of the iteration values is faithfully transcribed to our event space. The generated sequence goes as follows:



Musical Example 24. Music from the Mandelbrot Set

Comparing both the segment on the Mandelbrot Set and its musical transcription, one can readily see how features on the mapped slice are realized musically. For instance, we observe sections of silence in measures 4-5, 6-15, and 37. These correspond to sections on the segment for points that do not escape to infinity, just as we decided on our mapping scheme. These sections on the segment (see Fig. 26) are:

,  and 

, respectively.

Note that most of the time the music is in the low register, this is a because the iteration values at those points remain low.

This technique could be extended by mapping a large number adjacent segments and superimpose then polyphonically. This would effectively perform a "scan" of the surface of the fractal. This method would undoubtedly map more faithfully the structure of the whole fractal than just a single one-dimensional slice. Of course, we are not restricted to pitch only, the same mapping method can be applied simultaneously to duration, dynamic, timbre, etc.

The Mandelbrot Set is just one among the thousands of fractal equations that are known today. This mapping technique can be applied to those equations in a similar way.

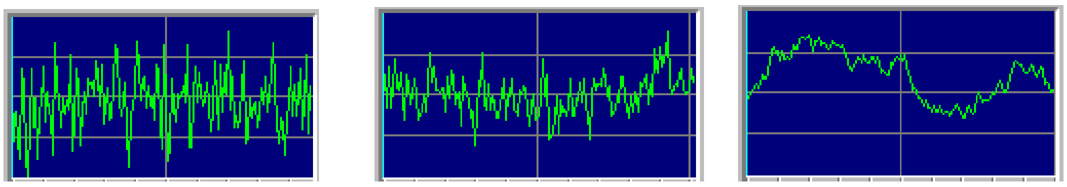
Noise

Paradoxically, noise can also be a source for algorithmic composition. Noise can be categorized into three basic types: white, pink and Brownian. These are differentiated on how their power spectrum (measured in dB) varies as a function of the frequency (measured in Hz.) The power of white noise is distributed evenly over all frequencies. In technical terms, its power spectra P(f) as a function of the frequency f behaves like: P(f) = $1/f^{a}$, where the exponent a is 0. For instance, white noise at a sampling rate of 44,100 Hz has as much power between 100 and 600 Hz as between 20,100 and 20,600 Hz. Frequencies are completely uncorrelated in white noise. White noise is what we hear as static on a radio.

Brownian noise——also known as "brown" noise——is quite the opposite; its frequency spectrum is highly correlated, resembling a random-walk in three dimensions, since the frequency fluctuations at a point in time depend on previous fluctuations; its power spectra P(f) is close to $1/f^2$.

1/f noise——also known as flicker noise, or "pink" noise——whose power spectra is 1/f (the exponent a in this case is very close to 1)——is somewhat middle ground between white and brown noises. 1/f noise has an even distribution of power when the frequency is mapped in a logarithmic scale. There is the same amount of power between 100 and 500 Hz than between 1000 and 5000 Hz. Frequencies in 1/f noise are not as correlated as in Brownian noise nor random as in white noise. To our ears it sounds like a "natural," even pleasant noise. In fact, 1/f noise appears everywhere in nature: the sound of a waterfall, the sound of rain, the sound of bees in a honey comb, are all 1/f noises.

A graphic of the frequency density of these noises help visualize the differences among them:



White noise 1/f noise Brownian noise

Figure 27. Spectra of white, Brownian, and 1/f noises

In 1975, Richard F. Voss and John Clarke at the University of California analyzed several recordings of music and speech.[29] They concluded that the audio power of

[29]J. Clarke and R. F. Voss, "1/f noise in music and speech," *Nature* 258 (1975): 317-318.

the music, e.g. the power delivered to the speakers, was very close to that of 1/f noise. Their research examined very different types of music, from Bach's *Brandenburg Concerti* to Scott Joplin's piano rags.

However, Voss and Clarke's results sparked controversy among other researches. Their main concern was that Voss and Clarke used long stretches of recorded material, some of which exceeded twelve hours in duration. These recordings were amalgamations of pieces by different composers, of greatly contrasting styles, edited together in sequence and interspersed with spoken announcements and comments. The argument posed against this study was simply that these "medleys" of different musics and sounds did not truly represent music in its essential manifestation, as a single, uninterrupted piece. The contenders argued that a single musical piece, being the largest unit of artistic significance, should have been the study model, instead of an arbitrary series of disparate excerpts.

Jean-Pierre Boon and other researchers formulated a different technique for the quantitative analysis of music.[30] Instead of analyzing recordings of music, a synthesizer was interfaced with a computer, and the

[30]J. P. Boon and O. Decroly, "Dynamical Systems Theory for Music Dynamics," *Chaos* 5(3) (1995): 501-508.

composition was played on the synthesizer by a performer. The pieces were digitally stored in the computer; this involved discretization of the pitch and duration. Data processing was then used to construct a phase portrait of the music. Nineteen classical pieces from all musical eras and four jazz pieces were used. Two other types of sequences were tested as well for comparison with the pieces: repeated ascending and descending scales and a sequence of 5000 notes based on a white noise algorithm.

The analysis of these pieces in this manner produced interesting results. When the values a for the spectral densities of the pieces ($1/f^a$) were computed, the results differed significantly from the results obtained by Voss and Clarke. This time, a seemed to vary between 1.79 and 1.97, which is closer to Brownian noise than 1/f noise. The difference was explained by the use of single pieces of music rather than long stretches. As Boon puts it, "if musical dynamics analysis is meant as a procedure to identify and characterize elements of musical significance, the single piece is the commonly recognized object to be studied. In this respect the meaning of long stretches of blended musical pieces is unclear."[31]

[31]Ibid., 507.

Algorithms for deriving music from white and brown noises are remarkably simple. "White" music can be easily created by simply mapping the output of a random number generator to an equiprobable collection of events in an ordered event space. Musical Example 1 is, in fact, an example of music derived from white noise. We have also encountered an algorithm which simulates Brownian noise: the *random-walk* Markov chain. We recall that this type of Markov chain (whose probability matrix has nonzero entries on either side of its main diagonal and zeroes everywhere else,) creates correlated sequences of events, which simulates the Brownian noise. 1/f noise, however, turns out to be more difficult to generate and hence be mapped into musical event spaces.

Naturally, Voss and Clarke were the first to experiment generating music from 1/f noise.[32] The first method consisted in generating 1/f noise through electronic means. The electrical voltages generated through this procedure were sampled, quantized, and converted to series of numbers whose spectral density was that of 1/f noise. These numbers were then mapped (using the normalized

[32]J. Clarke and R. F. Voss, "'1/f noise' in Music: Music from 1/f Noise" *Journal of the Acoustic Society of America* 63 (1978): 258-263.

mapping method) to event sets of pitch (a two octave chromatic scale,) and duration. They also devised an algorithmic method to simulate the spectral density of 1/f noise.

In a 1978 issue of *Scientific American*, Martin Gardner describes a simple algorithm involving three dice that emulates the spectral density 1/f noise.[33]

A non-linear equation whose output resembles 1/f noise, proposed by M. Schroeder[34], is

$$X_{n+1} = X_n * \lambda + \sqrt{(1-\lambda^2)} * r$$

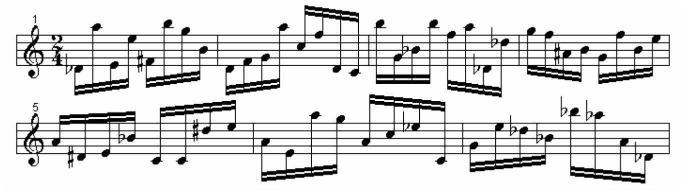
where λ is any number in the interval (0, 1) and r is a random value chosen for every iteration of the equation. The value of λ determines the "quality" of the output in relation to real 1/f noise. Chapter V on this thesis provides the source code of a C++ implementation of Voss' algorithm for producing 1/f noise.

---

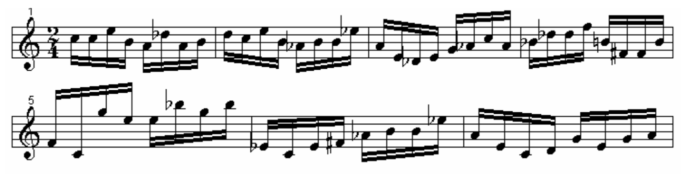
[33]M. Gardner, "White and Brown Music, Fractal Curves and 1/f Fluctuations," *Scientific American* (April 1978): 16-32.

[34]M. Schroeder, *Fractals, Chaos, Power Laws: Minutes from an Infinite Paradise* (New York: W. H. Freeman, 1991), 178.

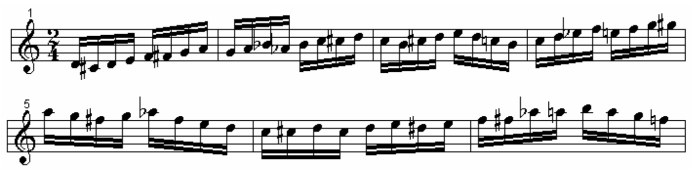
What follows are three musical examples derived from white, Brownian, and 1/f noises. The event space is the same for all: a two-octave chromatic scale from C_4 to B_5.



Musical Example 25. Music from white noise



Musical Example 26. Music from 1/f noise



Musical Example 27. Music from Brownian noise

The white noise music (Musical Example 25) was generated by mapping the output of a random number generator function (normalized mapping.)

The Brownian noise music (Musical Example 27) was created by mapping the output of a first-order Markov chain in which each pitch has an equal probability of being followed by its two immediate or preceding pitches: for instance, G_4 can be followed by F_4, $F\#_4$, G#, or A_4. This simulates the characteristic three-dimensional random-walk spectrum of Brownian noise.

Lastly, 1/f noise music (Musical Example 26) was generated by mapping the output of the non-linear equation proposed by Schroeder (above), with a value of $\lambda = 0.5$ (normalized mapping.)

Notice how the note-to-note relationship dramatically resembles the spectral density of the corresponding noises from which they are derived.

1/f noise is especially useful for the generation of sequences of musical events whose correlation is halfway between an aleatoric process and a random-walk Markov chain process.

Number theory algorithms

Number theory is one of the oldest branches of pure mathematics, as well as one of the most extensive. It concerns questions about whole numbers or rational numbers (fractions in which numerator and denominator are both whole numbers.)

Number theory has existed since the Pythagorean, which believed everything in reality could be explained with whole numbers. Their discovery of the square root of 2——an irrational number, meaning that it can never be described as the ratio between two whole numbers——provocated a crisis among them; they kept this discovery a secret, believing that it would be dangerous for the common people to know it.

There are many sources from number theory which can be used in algorithmic composition. We will study three of these sources: the Morse-Thue sequence; the "3n+1" number sequences; and the prime number series.

The Morse-Thue Sequence.

0110100110010110100101100110 1001...

This number sequence was first discovered by Axel Thue 1912 in his study of formal languages and rediscovered in 1917 by Marston Morse while studying the dynamics of surfaces geodesics.[35]

[35]M. Morse, "Recurrent Geodesics on a Surface of Negative Curvature," *Trans. Amer. Math. Soc.* 22 (1921): 84-110.

The Morse-Thue sequence (above) can be constructed both recursively and non-recursively. The recursive method uses the following substitution map:

1 -> 10

0 -> 01

Starting with a single 0, we get:

$0 \rightarrow 01 \rightarrow 0110 \rightarrow 01101001 \rightarrow 0110100110010110$, etc.

It can also be constructed non-recursively from the set of the natural numbers {0, 1, 2, 3, 4...} expressed in base 2 and taking the "digital root" (the sum of the ones modulo 2,) as follows:

0	1	2	3	4	5	6	7	8	9	10...
						↓				
0	1	10	11	100	101	110	111	1000	1001	1010...
						↓				
0	1	1	0	1	0	0	1	1	0	0...

One of the most striking characteristics of this sequence is that it exhibits self-similarity: removing all the even terms of the sequence leaves it unchanged:

$$0\underline{1}1\underline{0}1\underline{0}0\underline{1}1\underline{0}0\underline{1}0\underline{1}1\underline{0}1\underline{0}0\underline{1}0\underline{1}1\underline{0}0\underline{1}1\underline{0}1\underline{0}0\underline{1}...$$

$$0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 0\ ...$$

Also, removing every other pair of numbers leaves it unchanged:

$$01\underline{10}10\underline{01}10\underline{01}01\underline{10}10\underline{01}01\underline{10}01\underline{10}10\underline{01}...$$

$$01\quad 10\quad 10\quad 01\quad 10\quad 01\quad 01\quad 10\quad ...$$

We will use the non-recursive method to map the Morse-Thue sequence to musical events, as follows.

Consider the sequence of natural numbers expressed in binary form (base 2):

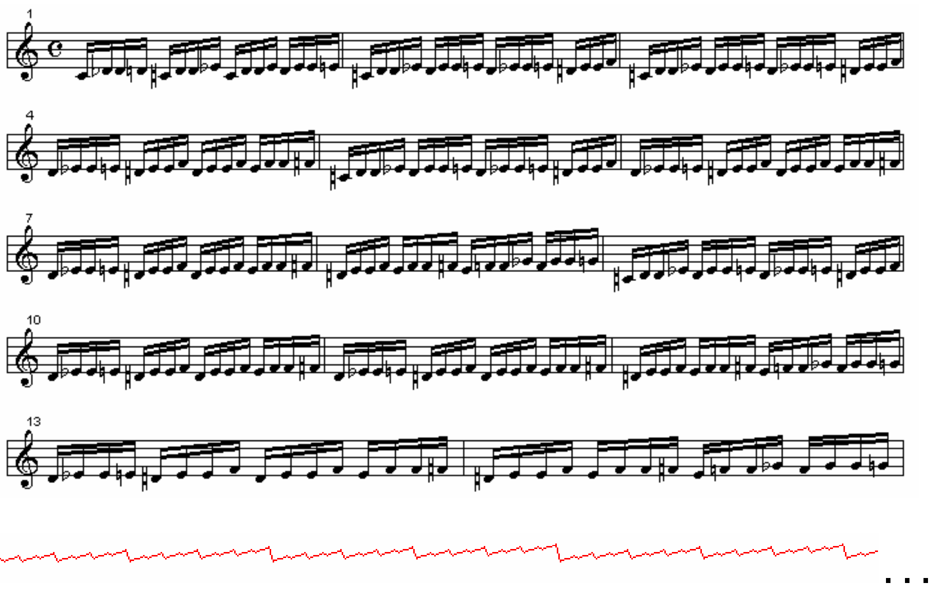
0	1	2	3	4	5	6	7	8	9	10...
						↓				
0	**1**	**10**	**11**	**100**	**101**	**110**	**111**	**1000**	**1001**	**1010**...

Next, add the digits (in decimal base) of each member of the sequence, as follows:

0	1	10	11	100	101	110	111	1000	1001	1010...
						↓				
0	**1**	**1**	**2**	**1**	**2**	**2**	**3**	**1**	**2**	**2**...

This transformed sequence is the one we will map to our musical event space. As with previous examples, we will choose a two-octave chromatic scale from C_4 to B_5 (24 pitches.) Mapping is applied performing a modulo 24 operation on each term of the sequence. For simplicity, the duration of the pitches has been fixed to sixteenths.

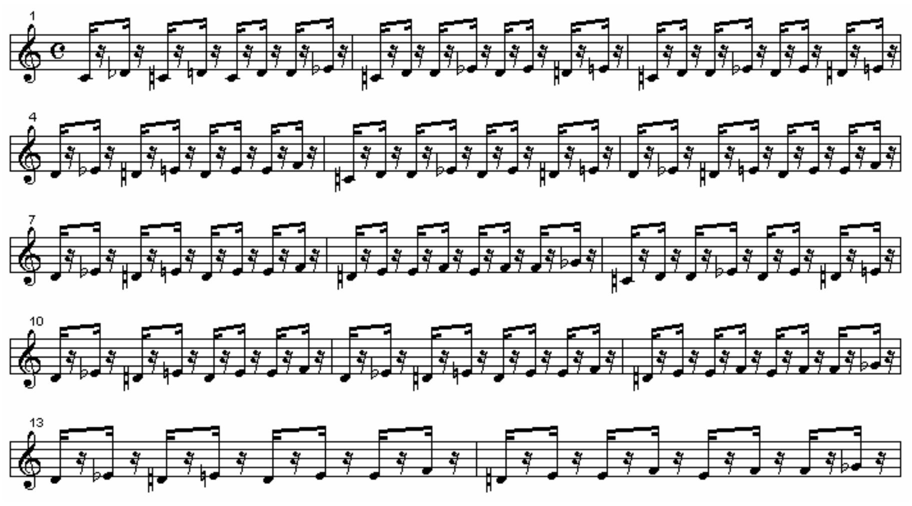
The generated sequence is as follows:



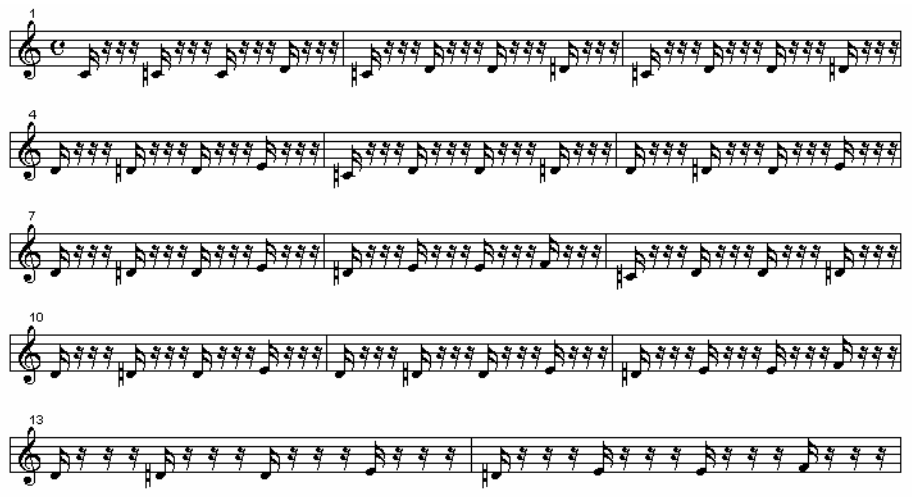
Musical Example 28. Music from the Morse-Thue sequence

Below the music there is a graphical representation of the sequence. Notice the contour resemblance between the music and the graphic.

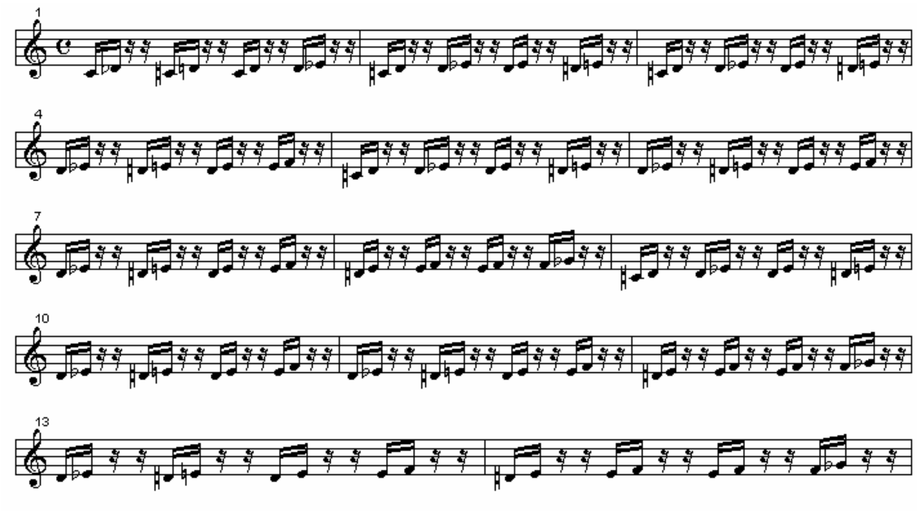
To show how the self-similarity of the sequence is maintained in the musical transcription, we will perform the same transformations on the music than we did on the sequence before. Removing every other note leaves the sequence unchanged:



Performing the same operation on this already truncated sequence, still leaves the sequence unchanged:



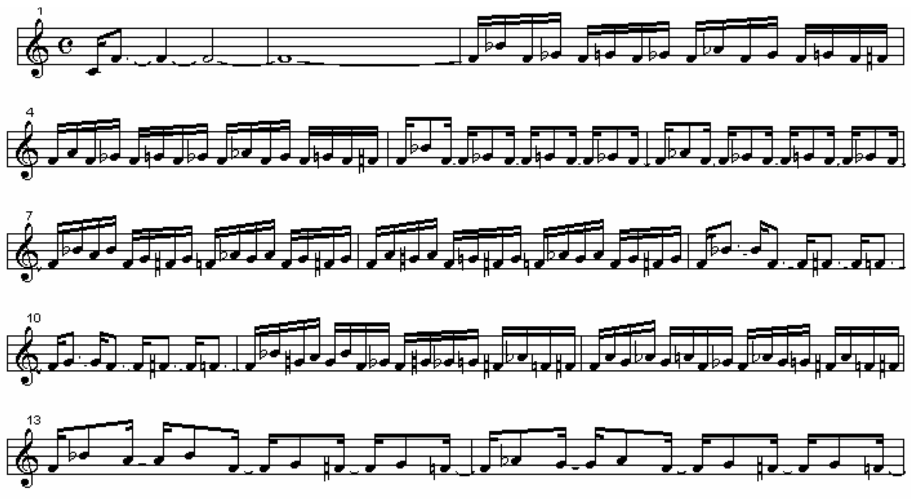
Removing every other couple of notes also leaves the sequence invariable:



We can extend the musical potential of the Morse-Thue sequence in several ways. First, we can multiply every term in the natural number sequence by a constant value. This produces surprising results. For instance, we can choose the constant value 31:

```
0     1     2     3     4     5     6     7     8     9     10...
                  multiply every term by 31
                              ↓

0     31    62    93    124   155   186   217   248   279   310...
                      express in base 2
                              ↓

0     11111 111110      1011101     1111100     10011011    10111010
      11011001    11111000    100010111 100110110...
                  add the ones (in base 10)
                              ↓

0     5     5     5     5     5     5     5     5     5      5...
```

The resulting sequence is as follows (pitches that repeat consecutively are tied on all forthcoming examples):

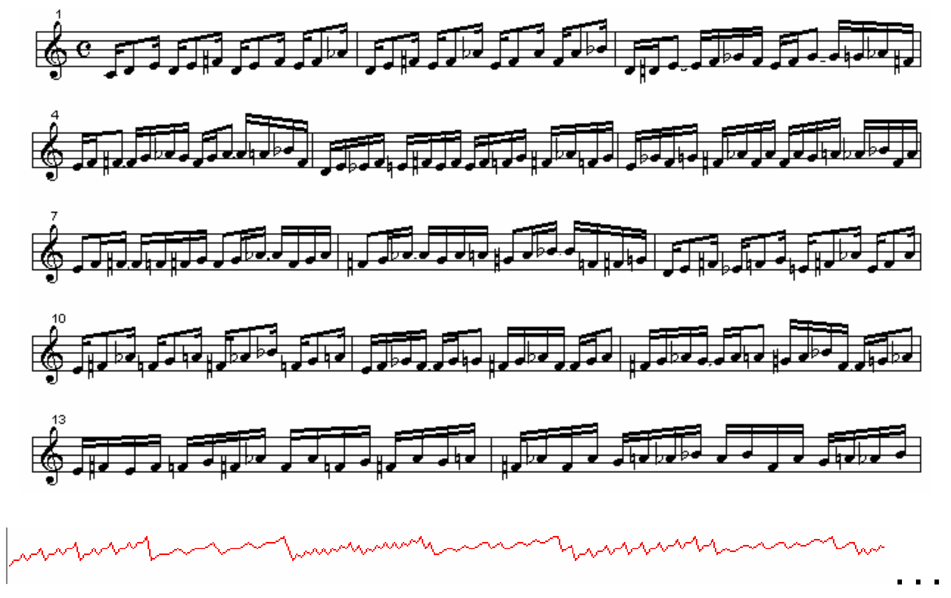


...

Musical Example 29, base 2 multiplier 31

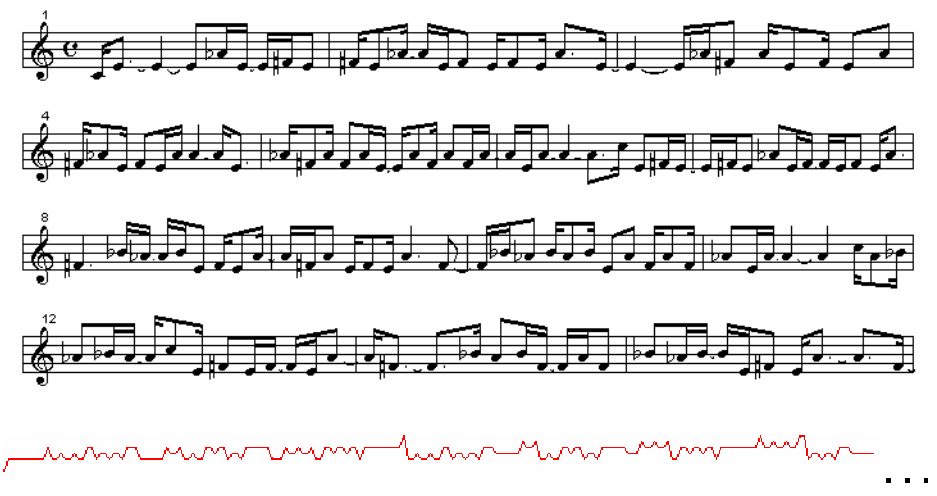
Observe how the very long note at the beginning corresponds to the series of 5s that repeat in the sequence. There are exactly 32 repeating fives. In our music event space element number 5 is mapped to an F_4. The sequence then evolves in more interesting ways. Note the structure of expanding and contracting patterns that are generated.

Changing the constant generates different sequences. The following was generated multiplying every term by 33:

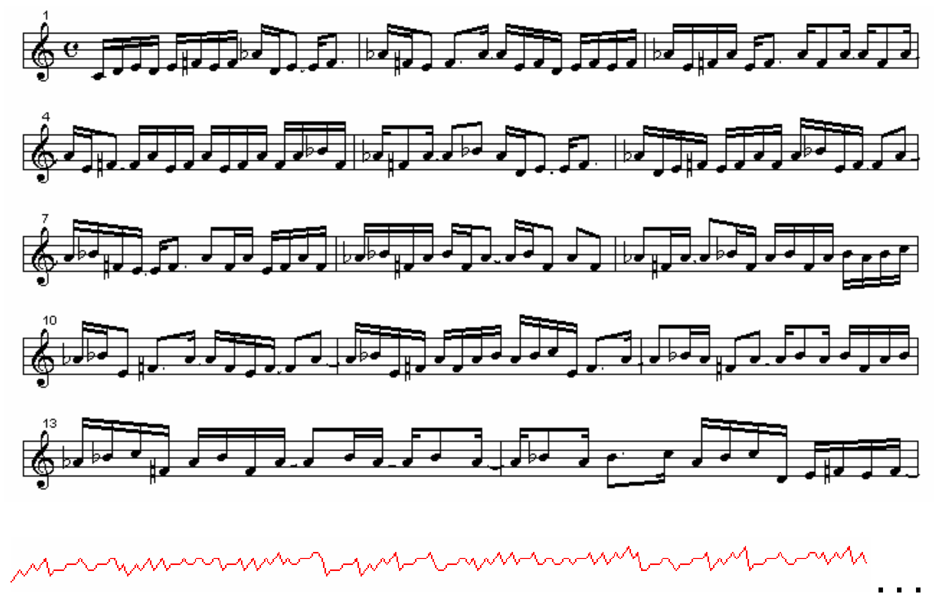


Musical Example 30, base 2 multiplier 33

We can extend this technique by expressing the natural numbers in bases other than binary base. The following two examples were generated choosing base 3 and multipliers 8 and 10, respectively:

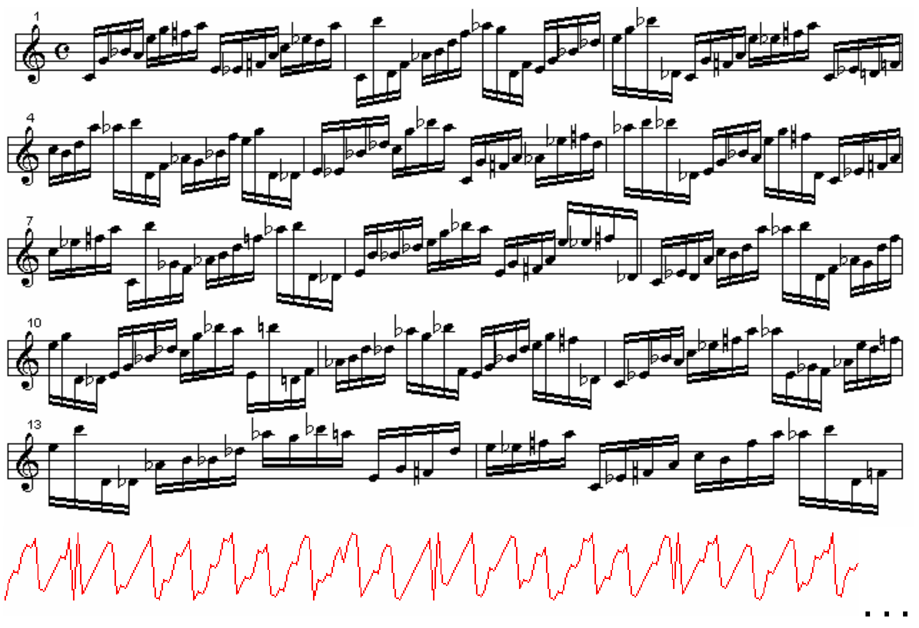


Musical Example 31, base 3 multiplier 8



Musical Example 32, base 3 multiplier 10

Notice how each base-multiplier pair generates a unique sequence. In general, higher basses generate sequences with elements more spread out in the musical event space:



Musical Example 33, base 29 multiplier 311

Some combinations of base-multiplier gives particularly interesting results, some of such combinations are

- Base *n* and multiplier $n^k \pm 1$, where $k = \{1, 2, 3, 4...)$
- Base *n* and multiplier $n! \pm k$, k close to *n* (this is practical for small bases only)

By extending our musical event space and our mapping schemes, Morse-Thue sequences offer virtually infinite ground for musical exploration.

3n+1 numbers

Also known as *the hailstone problem*[36], these number sequences are generated starting with an integer *n* and recursively performing the following operations:

If *n* is even, divide *n* by 2 (*n*/2.)

If odd, multiply *n* by 3 and add 1 (3**n* + 1)

For instance, starting with *n* = 15, we get the following cycle:

15, 46, 23, 70, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1...

Although it has never been proven, it is conjectured that all numbers eventually "fall" to 1 after a finite number of iterations, entering a never ending 4-2-1 loop. Before entering this loop, the numbers go up and down, like hailstones do in a cloud, hence their name. Note that the series for the number 15 (above) encompasses the series for all numbers in between (46, 23, 70, etc..) This can be generalized to any number *n.*

[36]J. Lagarias, "The 3x+1 Problem and its Generalizations," *American Mathematical Monthly* 92 (1985): 3-23.

The number of steps required to hit the number 1 (including the starting number itself) and hence enter the 4-2-1 cycle is the size of the cycle for any particular number n. This can be symbolized as:

$$H_s(n)$$

Another interesting factor concerning this series is the maximum number reached in the sequence generated by n,

$$H_{max}(n)$$

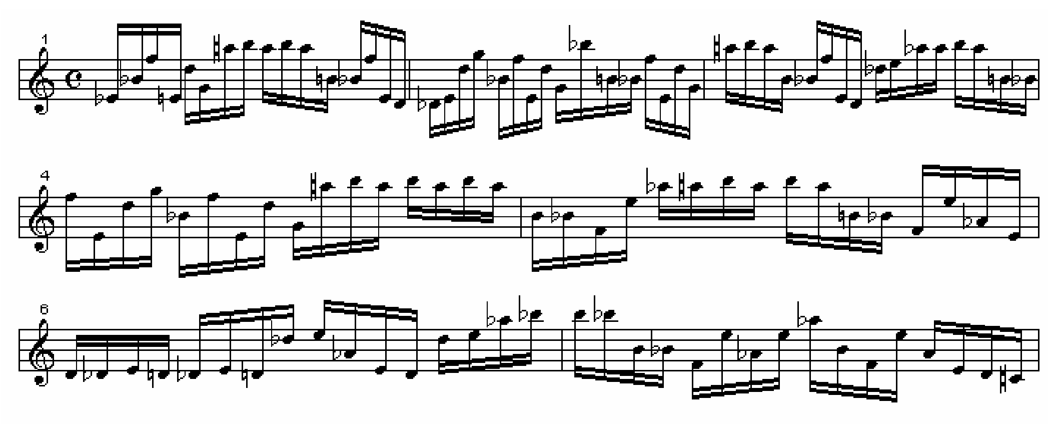
The number 9,232 seems to be the highest point reached by many numbers and a preferred track down to 1. The reason still remains a mystery. Among the first 10,000,000 numbers, around 38% of them hit this value before falling to 1.

The seemingly chaotic behavior of these sequences yields, however, patterns when mapped to musical events.

Mapping hailstone sequences to musical events is a straightforward procedure. Considering a musical event space of n ordered elements, we simply perform a modulo n operation to the particular number in the sequence to be mapped:

event number = (number in the sequence) mod n

For instance, the number 27 ($H_s(27) = 112$) produces the following sequence (our event space will be, as usual, a two-octave chromatic scale from C_4 to B_5, with sixteenth note values chosen arbitrarily):

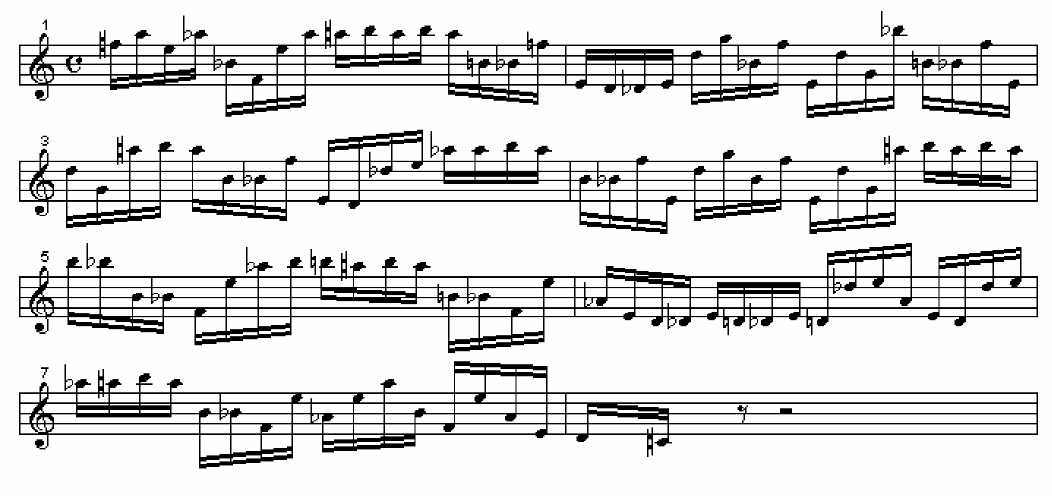


Musical Example 34. Music from number 27

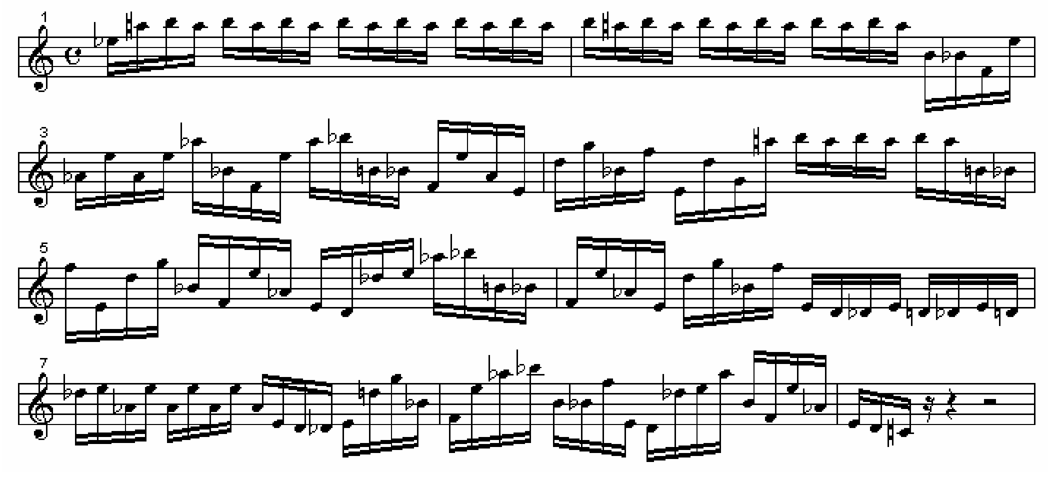
A close inspection of this sequence reveals it is far from random. There are many patterns, such as the repeating $A^{\#}_5$ - B_5 - $A^{\#}_5$ - B_5, and B_4 - B^b_4 – F_4 - E_5. The sequence seems to be highly disjunct except for small clumps around $A^{\#}_5$ - B_5, and E_4 - D_4 - $C^{\#}_4$.

It is interesting to notice that some pitch classes, such as $F\#_4$, $F\#_5$, A_4, and A_5, do not occur.

Two other example will reveal more about this number sequences. This time we will map numbers 666, $H_S(666) = 114$; and 65535 (2^{16}-1,) $H_S(65535) = 131$:



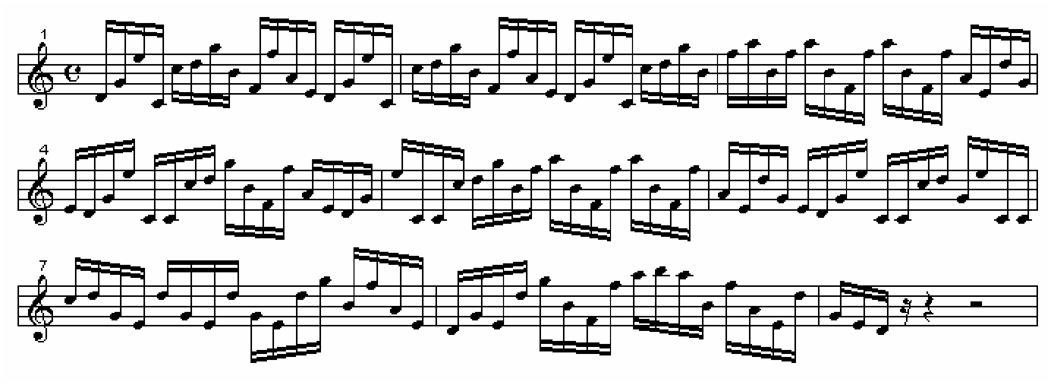
Musical Example 35. Music from number 666



Musical Example 36. Music from number 65535

There is something altogether questionable about these sequences, which seem to generate similar patterns for all given numbers. We observe, for example, that pitch class E_4 can only be followed by pitch classes D_4 or D_5. Pitch class $A^{\#}_5$ can only be followed by pitch classes B_5 and B_4. Similarly, pitch class $A^{\#}_4$ can only be followed by F_4 and F_5. Furthermore, F_4 can only be followed by E_5 and F_5 only by E_4. There seems to be a very rigid scheme on what elements can follow others. It becomes evident that one element can only be followed at most by two others. This is indeed very interesting, because we could construct a Markov chain that simulates this process, where some events have a two 0.5 probability entries and others a single 1.

The number of elements of the mapped event space, that is, the modulo operation applied, determines what kind of patterns arise. To illustrate this, let us map number 65535 again, this time to a two-octave *major* scale from C_4 to B_5, which has 14 elements as opposed to 24, therefore performing a modulo 24 operation:



Musical Example 37. Music from number 65535, major scale.

The patterns are now different, but similar restrictions as to what events can be followed by others still apply.

Different hailstone numbers could be used, for instance, to structure sequences of musical events where events can only be followed by a certain number of other events in the event space. This would be analogous to generating different event sequences from the same Markov chain.

3*n* + 1 numbers could be extended easily to 5*n* + 1, 7*n* + 1, or, more generally, a*n* + k, where both a and k are whole numbers. One of these cases, 3n - 1, generates sequences that end in one of three possible loops (instead of a single 4,2,1 loop):

1,2,1,2,1,2...

5,14,7,20,10...

17, 50, 25, 74, 37, 110, 55, 164, 82, 41, 122, 61, 182, 91, 272, 136, 68, 34, ...

These extended hailstone processes can be mapped in a similar way to musical event spaces, each of them generating sequences with particular characteristics.

The Prime Number Series

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37...

Prime numbers are one of most important and most studied subjects in number theory mathematics.[37] A prime number is a whole number that can only be divided evenly by itself and 1.

Prime numbers are the building blocks of all numbers. Every whole number has a unique factorization into prime numbers. Take the number 429, for instance; its unique factorization is 13*11*3. The distribution of prime numbers on the number line is one of the unsolved mysteries of mathematics. They do not seem to follow any recognizable pattern. A feasible formula for producing the

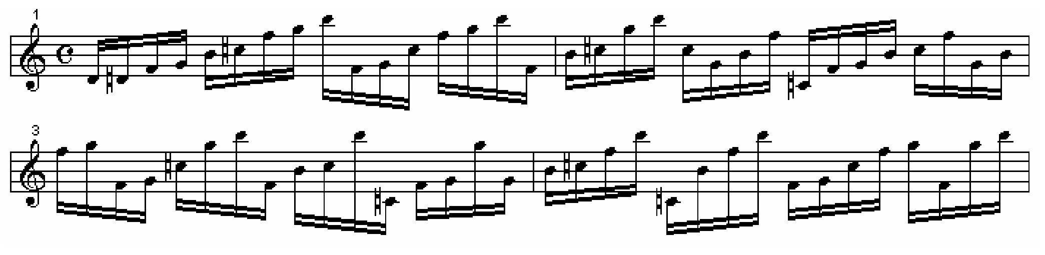
[37]Clawson, *Mathematical Mysteries*, 145-162.

prime numbers series has never been discovered. We know, however, that an infinity of prime numbers exist, that all prime numbers except 2 are odd, and that all prime numbers after 5 end in either 1, 3, 7, or 9. We also know that their density in the number line decreases as numbers get larger and larger.

There are many ways to map the prime number series to a musical event space. A simple procedure consists on mapping directly the series to a event space of *n* ordered members, performing a modulo *n* operation on the series:

event space member index = (prime factor) mod *n*

Considering the series of the first 64 prime numbers and applying this mapping to our usual event space of a two-octave chromatic scale, we get the following:



Musical Example 38. Music from prime numbers

Except for the two beginning pitches, the whole sequence maps to only 8 notes ($C^{\#}_4$, F_4, G_4, B_4, $C^{\#}_5$, F_5, G_5, and B_5.) This arises from the particular distribution of prime numbers among *n* columns (in this case 24):

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
    x x   x   x        x     x           x     x           x
          x   x              x           x     x           x
          x            x     x                 x           x
                             x
              x        x                 x
  x       x   x        x     x           x
                                .
                                .
```

Only columns 1, 5, 7, 11, 13, 17, 19, and 23 (except the particular case of primes 2 and 3) correspond with prime values. These columns correspond to pitches $C^{\#}_4$, F_4, G_4, B_4, $C^{\#}_5$, F_5, and G_5, B_5 in the event space. Note also that while the same pitches appear over and over again, they never produce a stable pattern. This is a consequence of the quasi-random distribution of primes in the number line.

A different mapping scheme, proposed by Armand Turpel in his computer program *Make Prime Music*[38], shows the prime number series from a different perspective. Beginning with prime number 7, triplets of consecutive prime numbers are performed a modulo 5 operation and subtracted 1 (since no prime after 5 ends in 5, this operations always yields a number between 0 and 3.) This triplet of transformed numbers is then interpreted as a base 4 number, which is then converted to base 10 and mapped to our event space:

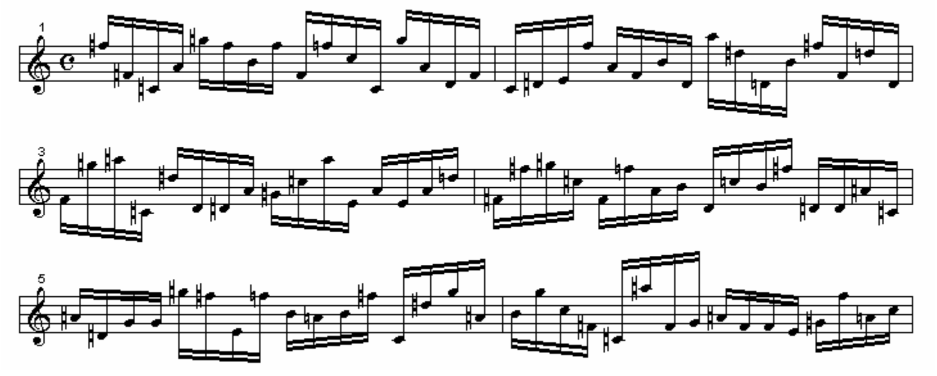
```
7     mod 5 = 2; 2-1 = 1 \
11    mod 5 = 1; 1-1 = 0  > 102 in base 4 is 18 in base 10
13    mod 5 = 3; 3-1 = 2 /
                                    `

17    mod 5 = 2; 2-1 = 1 \
19    mod 5 = 4; 4-1 = 3  > 132 in base 4 is 30 in base 10
23    mod 5 = 3; 3-1 = 2 /
```

The resulting sequence 18, 30... is mapped to the members of the event space via another modulo operation:

event number = (number in sequence) *mod* (number of events)

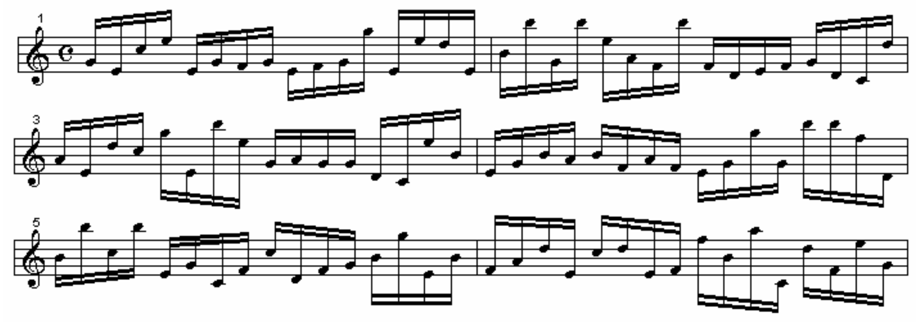
This mapping scheme performed in our two-octave chromatic scale event space, yields the following sequence:

[38]Make Prime Music, Armand Turpel; available from http://www2.vo.lu/homepages/armand/index.html; Internet.



Musical Example 39. Music from prime numbers, alternate mapping.

The same sequence mapped to a two-octave *major* scale becomes:



Musical Example 40. Music from prime numbers, major scale.

These two sequences involve 288 consecutive prime numbers (each note represents 3 prime numbers.) It is apparent from these sequences that the event-to-event relationship in these sequences is highly unpredictable.

Nevertheless they are not random. This unpredictability arises, once again, from the mysterious distribution of primes along the number line.

For instance, in Musical Example 39, there seems to be an excess of pitch classes $F^{\#}_4$, A_4, B_4, and $F^{\#}_5$ (events 6, 9, 11, and 18, respectively,) and a deficiency of pitch classes E_5, A_5, and B_5 (events 16, 19, and 23.)

In a truly random sequence of events, all events would have the same probability of occurrence, and the event-to-event relationship would be absolutely uncorrelated.

The application of different mappings schemes to the same process shows that, although the generated sequences differ, there is an underlying structure common to all of them.

Cellular Automata

Cellular automata are a very rich source for algorithmic composition. Compositional applications of cellular automata have been explored in the work of Hunt, Kirk, and Orton[39], Millen[40], and Miranda.[41]

Cellular Automata theory[42] was introduced by John von Neumann in the 1960s as a model of biological self-reproduction.[43] Cellular automata are *discrete* dynamical systems, which is to say that the space, time and properties of the systems can only have a finite number of states.

Cellular automata have two fundamental characteristics:

---

[39]A. Hunt, R. Kirk, and R. Orton, "Musical Applications of a Cellular Automata Workstation," in *Proceedings of the 1991 International Computer Music Conference* (San Francisco: International Computer Music Association, 1991), 165-168.

[40]D. Millen, "Cellular Automata Music," in *Proceedings of the 1990 International Computer Music Conference* (San Francisco: International Computer Music Association, 1990), 314-316.

[41]Miranda, "Music Composition Using Cellular Automata," 105-117.

[42]Wolfram, *Theory and Applications of Cellular Automata.*

[43]von Neumann, *Theory of Self-Reproducing Automata.*

1. A regular, n-dimensional lattice, where each cell in this lattice has a discrete state at a given time.
2. A dynamical behavior, controlled by a set of rules. These rules decide the state of the cells for subsequent time steps (generations,) depending on the state of neighboring cells.

Cells in a cellular automata are like memory devices, which store the automata's states. The simplest case involves only two possible states for each cell, usually 0, "dead", or 1, "alive." In more complex cellular automata, cells can have many more states. Cells are arranged in a n-dimensional lattice. In the case of one-dimensional cellular automata——the simplest types——cells are arranged on a single line; two-dimensional automata cells are arranged on a flat grid (a two-dimensional matrix); three-dimensional automata in a cubic matrix; and so forth.

Theoretically, cellular automata can exist in any dimension, however, cellular automata in more than two dimensions are exceedingly difficult to visualize.

The rules that control the dynamical behavior of the cellular automata act upon each cell in the lattice having into account the states of each cell's neighboring cells.

These rules will decide the state of each cell in subsequent time steps (generations.)

There are three basic types of cell neighborhoods to be considered when defining a set of rules (here we will consider a two-dimensional cellular automaton):

1. The so-called *von Neumann* neighborhood, where each cell has four neighbors (North, South, East, and West cells,) the radius of this neighborhood is 1, since only adjacent cells are considered:

```
  n
n c n
  n
```

2. The *Moore* neighborhood, where each cell has eight neighbors (N, NE, NW, S, SE, SW, E, and W cells.) The radius is also 1:

```
n n n
n c n
n n n
```

3. The extended Moore neighborhood, where cells beyond the radius of the Moore neighborhood are considered. The radius can be 2 or larger:

```
n n n n n
n n n n n
n n c n n
n n n n n
n n n n n
```

For a one-dimensional automata, the von Neumann and Moore neighborhoods are identical:

n **c** n

The extended Moore neighborhood for a one-dimensional automata would be:

n n **c** n n

The rules that determine the dynamical behavior of the system can be categorized into two classes:

1. The state of a given cell is determined by examining the state of all neighboring cells individually (including the given cell itself.)
2. *Totalistic* rules, where the state of any given cell is determined by the sum of the states of its neighboring cells.

The number of possible rules in a given cellular automaton depends on the number of states per cell and the number of neighbors per cells. The set of possible rules grows exponentially as more states and neighbors are allowed.

According to S. Wolfram, cellular automata behavior can be classified into four basic classes[44]:

- Class I. Limit point. After a finite number of generations, a stable, unchanging configuration is achieved by the system.
- Class II. Limit cycle. These types of automata evolves to a stable state where patterns repeat periodically.
- Class III. The system generates aperiodic, chaotic patterns from nearly all starting conditions. Patterns can resemble self-similar fractal curves.
- Class IV. The behavior of the system is complex but not chaotic. This class is the only one capable of performing *universal computation*, meaning that it is able to carry out any finite algorithm.

One-dimensional Cellular Automata

We will consider a simple, one-dimensional cellular automaton proposed in 1982 by S. Wolfram.[45] This type of cellular automaton, because of its simplicity, has been

[44]S. Wolfram, "Universality and Complexity in Cellular Automata," *Physica D* 10 (1984): 1-35.

[45]S. Wolfram, "Statistical Mechanics of Cellular Automata," Caltech preprint CALT-68-915 (May 1982).

extensively studied. It has been proved that even one-dimensional cellular automata are able to perform universal computation.[46]

Each cell is allowed two state possible states (0, dead, or 1, alive.) The neighborhood is of the Moore type: each cell has two neighbors, the cells to the immediate left and right. This type of cellular automaton allows for 256 possible rules: all three cells in the neighborhood are allowed two-states, which gives 8 (2^3) possible combinations of cell states, and each of these can decide upon two possible states for the core cell in the next generation. The total number of possible rules is thus 2^8 = 256, as follows:

possible cell configurations:	111	110	101	100	011	010	001	000
	↓	↓	↓	↓	↓	↓	↓	↓
	1 or 0	1 or 0	1 or 0	1 or 0	1 or 0	1 or 0	1 or 0	1 or 0

Rules are encoded in an 8-bit string. For instance, rule 30 is encoded as 00011110:

[46]K. Lindgren and M. G. Nordahl, "Universal Computation in Simple One-dimensional Cellular Automata," *Complex Systems* 4 (1990): 299-318.

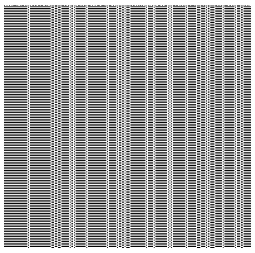
$$\begin{array}{cccccccc}
111 & 110 & 101 & 100 & 011 & 010 & 001 & 000 \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 0
\end{array}$$

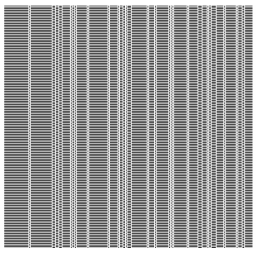
Rules decide the state of each cell in the next generation depending on the state of itself and its neighbors in the current generation.

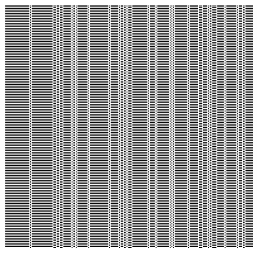
One-dimensional automata evolve on a single line of cells. To better appreciate the development of the whole system, subsequent generations (time steps) are presented one after the other, like an unfolding rug. Each "thread" of this rug represents one time step, or generation, of the cellular automaton. What follows shows the dynamical development of this one-dimensional automaton for four different rules. Nine hundred generations were computed for each system. The lattice is 300 cells wide. Cells at the borders of the lattice neighbor each other (the lattice has a circular geometry.)

Rule 1, Class I:

$$\begin{array}{cccccccc} 111 & 110 & 101 & 100 & 011 & 010 & 001 & 000 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}$$

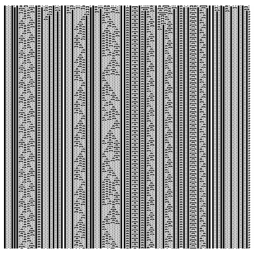

Gens. 0-299

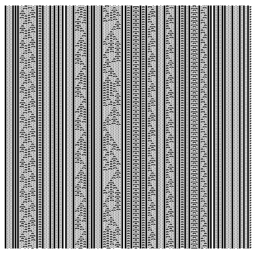

Gens. 300-599

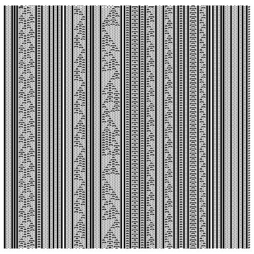

Gens. 600-899

Rule 73, Class II:

$$\begin{array}{cccccccc} 111 & 110 & 101 & 100 & 011 & 010 & 001 & 000 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{array}$$

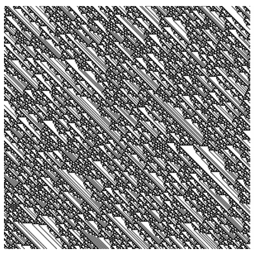

Gens. 0-299

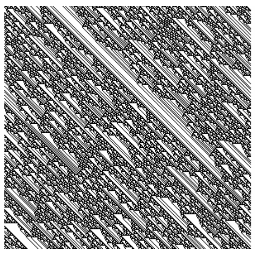

Gens. 300-599

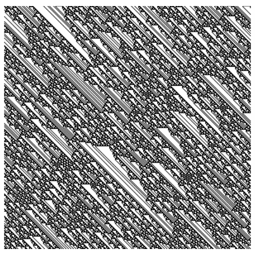

Gens. 600-899

Rule 120, Class III:

$$\begin{array}{cccccccc} 111 & 110 & 101 & 100 & 011 & 010 & 001 & 000 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \end{array}$$

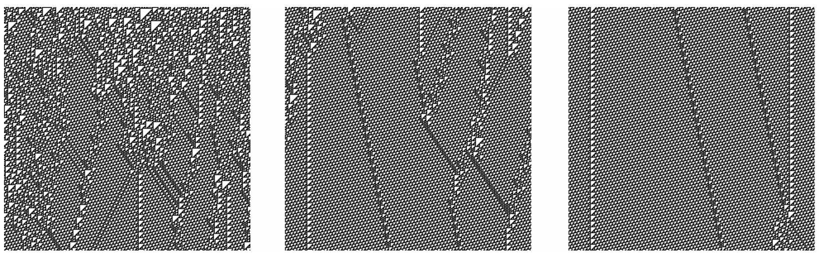

Gens. 0-299


Gens. 300-599


Gens. 600-899

Rule 110, Class IV:

111	110	101	100	011	010	001	000
↓	↓	↓	↓	↓	↓	↓	↓
0	1	1	0	1	1	1	0


Gens. 0-299 Gens. 300-599 Gens. 600-899

For generation 0, the cells' states were initialized randomly.

Mapping one-dimensional cellular automata to music involves transcribing the automata's evolution to a musical event space. The following mapping scheme is proposed by this writer.

Each generation will be mapped to the members of our choice musical event space. Since each generation involves a collection of cells, a direct correspondence must be made between each possible configuration of cells in the lattice and the event space. To achieve this, we must encode each possible combination of cells in the lattice with a unique number. Let s be the number of possible states for a cell, n the number of cells on the lattice (ordered from 0 to n-1,) and t_p the state of a cell in position p (p ranges from

0 to n-1.) Next, each possible configuration of cells in a generation is uniquely encoded numerically as follows:

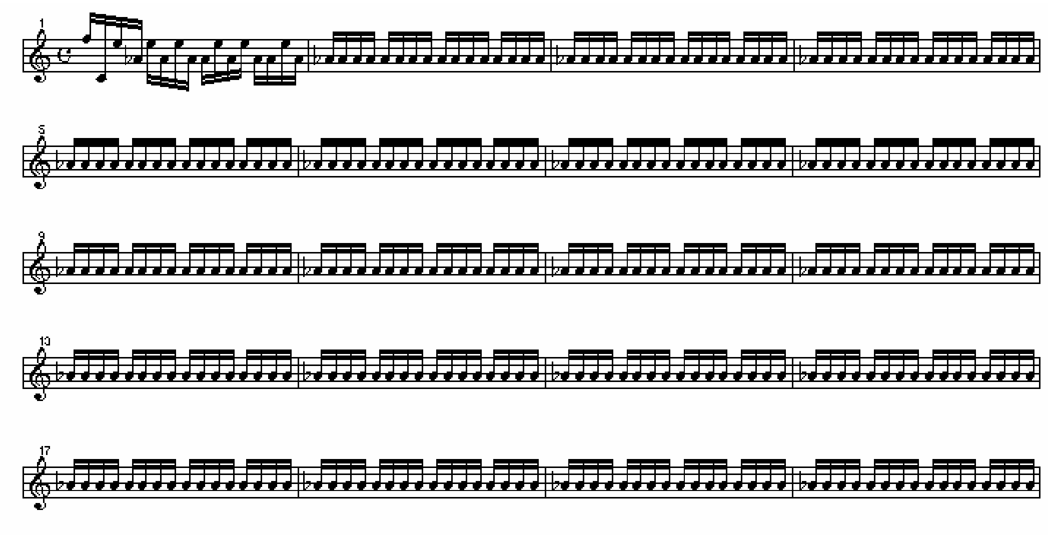
$$u = t_0 * s^0 + t_1 * s^1 + t_2 * s^2 + \ldots + t_{n-1} * s^{n-1}$$

This ensures each configuration of cells is assigned a unique number or "tag." This number is then mapped to our musical event space via a modulo operation, as follows:

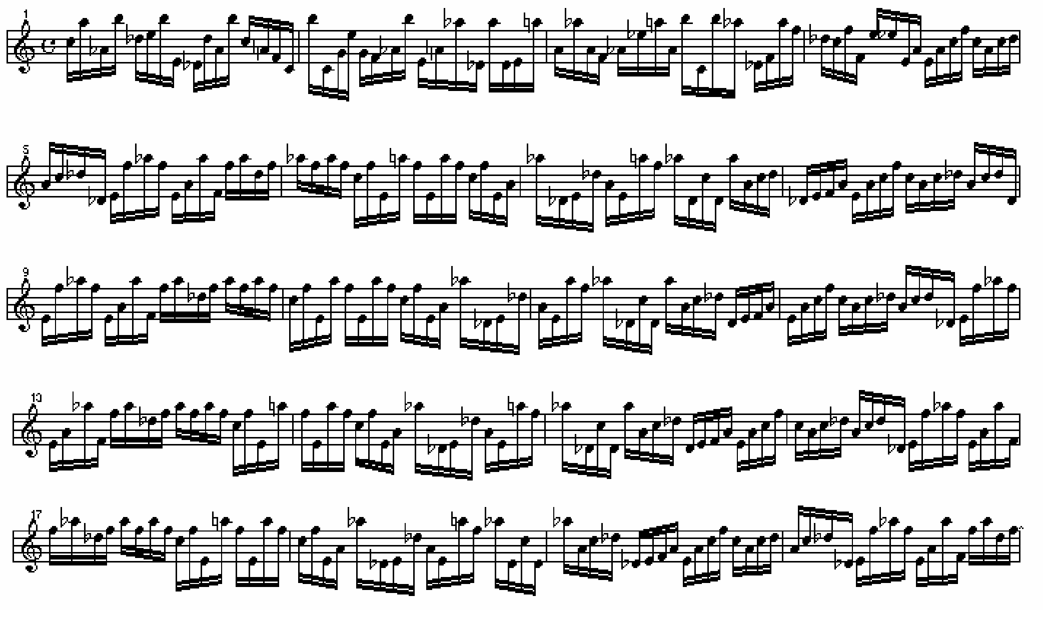
$$\text{event number} = u \bmod (\text{number of elements in event space})$$

Following this scheme, four configurations corresponding to rules 172 (Class I,) 73 (Class II,) 30 (Class III,) and 54 (Class IV) were mapped to a musical event space consisting of a two-octave chromatic scale from C_4 to B_5. The lattice has 64 cells. The number of possible cell configurations is therefore 2^{64}.

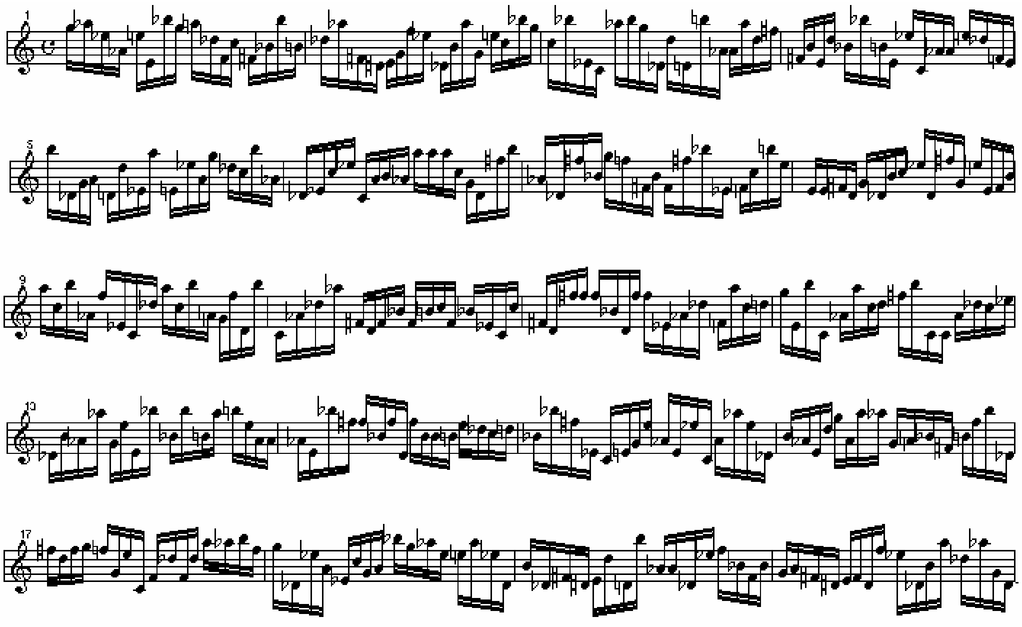
Rules were chosen to show how the characteristics of the different classes are transcribed into music. Three hundred and twenty generations were mapped from each configuration (for these examples, note values were chosen arbitrarily.) The musical transcription is as follows:



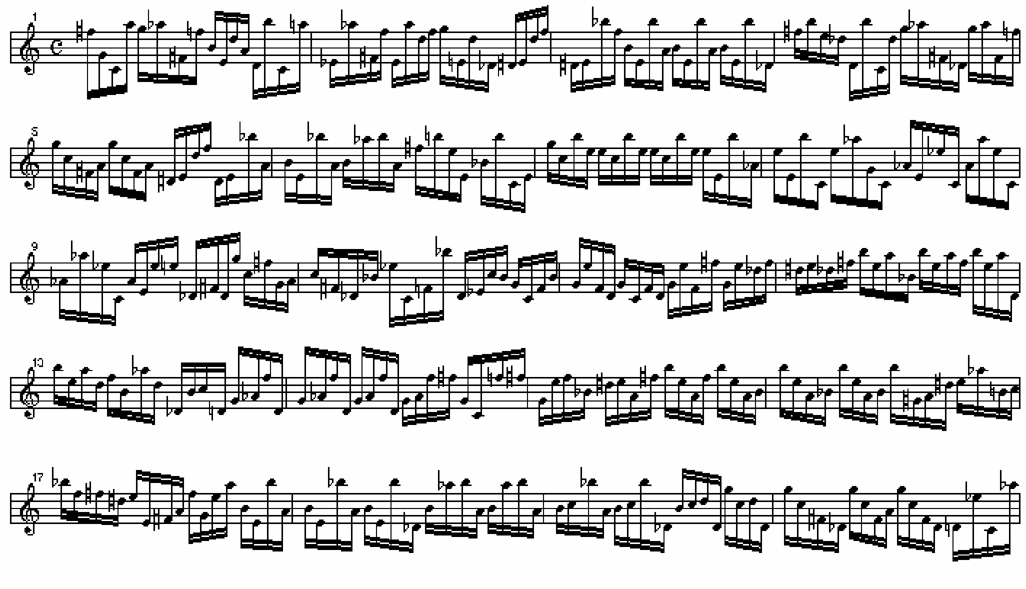
Musical Example 41. Music from rule 172, Class I.



Musical Example 42. Music from rule 73, Class II.



Musical Example 43. Music from rule 30, Class III.



Musical Example 44. Music from rule 54, Class IV.

The similitude of this sequences to the characteristics of the corresponding automata class is extraordinary. Note how Musical Example 41 quickly stabilizes to a unique member of the event space (A^{b}_{4}.) This is characteristic of Class I automata, which have single-point attractors.

Musical Example 42 shows how the limit cycle characteristic of Class II is transcribed into music. Note the repeating 60-note cycle starting with the third beat of m. 4.

The chaotic behavior of Class III is demonstrated in Musical Example 43. This music is very similar to the "white noise" music (Musical Example 25,) as well as to the Logistic equation mapping for a value of $\mu = 4$ (Musical Example 10.) No discernible patterns can be found. It resembles a randomly chosen collection of pitches, although, of course, the procedure is completely deterministic.

Musical Example 44 is by far the most interesting and, this writer believes, the most musically pleasing. This example shows the behavior of Class IV automata. The music is highly patterned, in direct correspondence with the automata's capacity of information propagation. It does not have the predictability of Musical Examples 41 and 42

(Classes I and II,) nor the random-like character of Musical Example 43 (Class III.) It must be stressed that each pitch (or event) in the example sequences represents one *generation* in the development of the automata, and thus, a definite configuration of cells. This mapping scheme preserves the automata's characteristics when mapped to musical event spaces.

By increasing the number of states per cell, the possible set of rules grows exponentially, and thus the type of cellular automata that can be generated——allowing 3 states per cell instead of two increases the number of possible rules from 256 to 7,625,597,484,987 (3^{27}.) The possibilities for musical experimentation are endless.

Two-dimensional cellular automata

Two-dimensional cellular automata evolve in a two-dimensional matrix or grid of cells. The behavior of two-dimensional automata is much more complex and offers even more possibilities for experimentation. Like one-dimensional automata, 2-D automata are governed by a set of rules that ultimately determine the behavior of the system.

We will consider here the most studied 2-D cellular automata: the so-called *Game of Life*.

The *Game of life* is a Class IV automata invented in 1970 by mathematician John Conway. It has been extensively studied since it was first presented in a 1970 issue of *Scientific American*.[47] It is a two-state cellular automaton, that is, cells in the grid have only to possible states: 0 (or dead,) and 1 (alive.) Each cell in the lattice has eight neighbors (Moore type neighborhood.) The rules that determine the dynamical behavior of the automata are exceedingly simple:

- Rule of birth: if a cell is in state 0 (dead) and has exactly 3 neighbors in state 1 it changes its state to 1 (comes to life) in the next time step.
- Rule of survival: if a cell is in state 1 and has 2 or 3 neighbors in state 1 it remains in state 1 in the next time step, otherwise it changes to state 0 (dies.)

Many persistent structures, some propagating, have been discovered in this cellular automaton[48] (the simplest one is known as the *glider*.) In addition, it has been

[47]M. Gardner, "Mathematical Games: the Fantastic Combinations of John Conway's new Solitary Game 'Life'," *Scientific American* 223 (April 1970): 120-123.

[48]Ibid., 122.

shown that these structures can be combined to perform arbitrary information processing. This means that the automata is capable of universal computation. Among the persistent structures, the simplest one is known as the *block*. It consists of for cells grouped as follow:

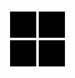


Figure 28. Block.

Another very common structure is the *blinker*. This structure is not static, but has a period-2 oscillation:

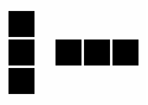


Figure 29. Blinker.

The *glider* is the simplest propagating structure. It is a 5-cell structure that propagates diagonally (not shown) every 4 generations, or time steps.

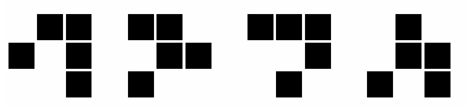


Figure 30. Glider.

The *Game of life* Cellular Automaton can be a good source for musical experimentation, since it allows universal computation and pattern propagation.

See the Appendix for a complete discussion of the proposed mapping process and an analysis of a piece generated by this cellular automaton.

Genetic Algorithms

Almost a century and a half ago, Charles Darwin discovered that organisms that remained immutable in a changing environment would be unable to adapt to the new circumstances and therefore die.[49] Darwin observed that, as environmental conditions changed, organisms that were better adapted to the new environment survived and gave birth to offspring that inherited those beneficial traits, while the non-adapted ones died. Darwin called this process "natural selection," and considered it to be the manner by which species evolved in nature. This process is also known as "survival of the fittest." Fitness is never a fixed quantity. An individual's fitness depends on many factors: a changing ecosystem, competing species and competing members of the population, etc. As the environment changes, the fitness of an individual is also affected.

Darwin did not know the process by which the parents' characteristics were inherited by their offspring. He merely observed it. Almost a century passed until, in 1953, Francis Crick and James Watson discovered the

[49]C. Darwin, *On the Origin of Species* (Cambridge, MA: Harvard University Press, 1964).

deoxyribonucleic acid molecule (DNA,) responsible for the process of biological inheritance.[50] Every bit of information concerning an individual is encoded in this tiny double helix, from hair color to potential illnesses.

Simple organisms such as bacteria and fungi reproduce *asexually* by duplicating themselves. This creates individuals that differ very little from their progenitors. Most higher forms of life, however, reproduce *sexually*, where offspring inherit characteristics of both father and mother. Sexual reproduction increases variation within a species.

The DNA molecule is a sequence of four basic building blocks called nucleotides. The human DNA molecule, for instance, consists of over 3 billion of these nucleotides. These nucleotides are grouped to form *genes*. Genes encode the information to build the proteins and enzymes that determine all characteristics of an individual. Genetic information from both father and mother are mixed at the time of reproduction to form a new individual.

Natural selection changes the frequency of genetic information within a population, but it can not produce new

[50]F. H. C. Crick and J. D. Watson, "Molecular Structure of Nucleic Acids. A Structure for Deoxyribose Nucleic Acid," *Nature* 171 (1953): 737-738.

genes: *mutation* provides a way. Mutation is a random change in an organism's genes. The mechanisms that provoke mutations are many, such as radiation and cosmic rays. While it is unlikely that a random mutation will improve an organism that is well-adapted to the environment, in the long run mutation provides a way through which genetic information evolves.

Genetic algorithms in a computer emulate the way organisms reproduce and evolve in nature. It has been only recently that researches have started to simulate biological structures in software.[51] Fortunately, genetic algorithms need not to have into account as many variables as exist in real life. Whereas nature shows tremendous flexibility, the purpose of a computer algorithm is basically to find a specific answer to a specific problem.

A genetic algorithm must implement three basic ingredients: competition, survival, and reproduction.[52] The process involves four basic steps:

1. Initialization: A starting population of random "solutions" (called *chromosomes*) is created. Each

[51]S. R. Ladd, *C++ Simulations and Cellular Automata* (New York: M&T Books, 1995), 192.

[52]Ibid., 193.

bit of information which forms the chromosomes is known as an *allele*.

2. Fitness testing: Each chromosome in the population is assigned a fitness value based upon its evaluation against the problem.
3. Reproduction: Chromosomes are selected from the population to became parents of new solutions. Chromosomes with higher fitness values are more likely to be chosen. Reproduction is achieved by mixing information from parents chromosomes. Mutation can be introduced to the new solutions.
4. Next generation: A new generation of solutions is created. The process iterates back to step 2 until a population of solutions is created which satisfies the original problem.

Steps 2 and 3 are crucial. The reproductive success of chromosomes is directly linked to their fitness value. Chromosomes with a higher fitness value have a higher probability to be chosen as parents. This is a stochastic process; the outcome of a genetic algorithm is based on probability.

The most common technique for reproduction in a genetic algorithm is known as *crossover*. Crossover combines the information of both parents by randomly selecting a point (or several if the crossover is multiple) at which pieces of both parents chromosomes are combined to form an offspring:

Father chromosome	Mother chromosome	Offspring
abd|bcbag	bdd|cbabc	abd|cbabc

The simbol "|" denotes the point of crossover. Multiple cross over would be as follows:

Father chromosome	Mother chromosome	Offspring
ab|db|cb|ag	bd|dc|ba|bc	ab|dc|cb|bc

Mutation can and should be part of the reproduction process. It is introduced in the newly formed offspring by randomly changing one of its alleles. The primary purpose of mutation is to introduce variation into the population. However, it should be used judiciously: too little or no mutation limits diversity on populations, which end up not evolving at all; too much mutation, on the other hand, destroys the value of selection by fitness.

Genetic algorithms take full meaning in algorithmic music when one considers alleles as musical events and chromosomes formed by alleles as event spaces. By using a genetic algorithm on event sets (which can be predefined or generated by other algorithmic means,) the evolution of the event sets can be modeled according to a predefined scheme (the problem to be solved) decided by the composer.

Genetic algorithms turn out to be a great way to create sets of variations on a given event set. They have been successfully applied to both sound synthesis[53] and composition.[54]

The following examples illustrate how genetic algorithms can be used in music composition.

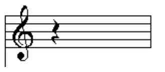
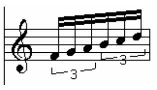
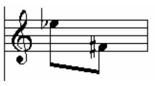
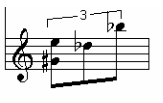
For simplicity and clarity, we will use only small population consisting of 10 individuals (chromosomes) per generation. Each "couple" will be allowed only one offspring. In addition, chromosomes (event spaces) are limited to 4 alleles (events): A, B, C, D. The total number of possible chromosomes is then $4^4 = 256$. These restrictions are imposed only for demonstration purposes. A truly functional example would use a much larger set of alleles and longer chromosomes. Our limited example,

[53]Beauchamp, Horner, and Haken, "Genetic Algorithms and Their Application to FM Matching Synthesis," 17-29.

[54]Goldberg and Horner, "Genetic Algorithms and Computer-Assisted Music Composition," 479-482.

however, is sufficient to demonstrate the power of this technique.

Our "alleles" are defined as follows:

A: , B: , C: , D: 

Our genetic algorithm has the following characteristics:

1. Initialization: A starting population of ten four-allele chromosomes will be generated randomly. A possible chromosome may be, for instance, ABBC.
2. One-point crossover reproduction technique will be employed.
3. The probability of mutation in offspring is set to 0.10 (one chance in ten.)
4. The fitness value is assigned according to the following scheme:
   - 4.1 Chromosomes consisting of four different alleles have the highest fitness value (0.30.)
   - 4.2 Chromosomes which contain alleles A and B are assigned a fitness value of 0.20.

4.3 Chromosomes consisting of four identical alleles are assigned the lowest fitness value (0.05.)

4.4 Chromosomes containing three identical alleles are assigned a fitness value of 0.08.

4.5 Chromosomes containing two identical alleles are assigned a fitness value of 0.10.

4.6 Other possible combinations are assigned a fitness value of 0.12.

The purpose of this fitness scheme is to promote populations (or sets of event spaces, musically speaking) that are very varied (containing chromosomes with different alleles.)

The genetic algorithms performs as follows:

1. Each chromosome in the initial population is evaluated according to the fitness scheme above and assigned a fitness value.
2. Ten couples of chromosomes from the current generation are selected stochastically according to their fitness values. Parents with a higher fitness value will be more likely to be selected.

3. Each couple reproduces to generate a single offspring. The point of crossover is generated randomly.
4. Chance of mutation is applied stochastically to each offspring.
5. Each offspring is evaluated against the fitness scheme and assigned a fitness value. This is generation 1.
6. The process is iterated from step 2 until we get a population that satisfies our fitness scheme.

This genetic algorithm was applied to our event sets of four-allele chromosomes, creating 10 generations with the following results:

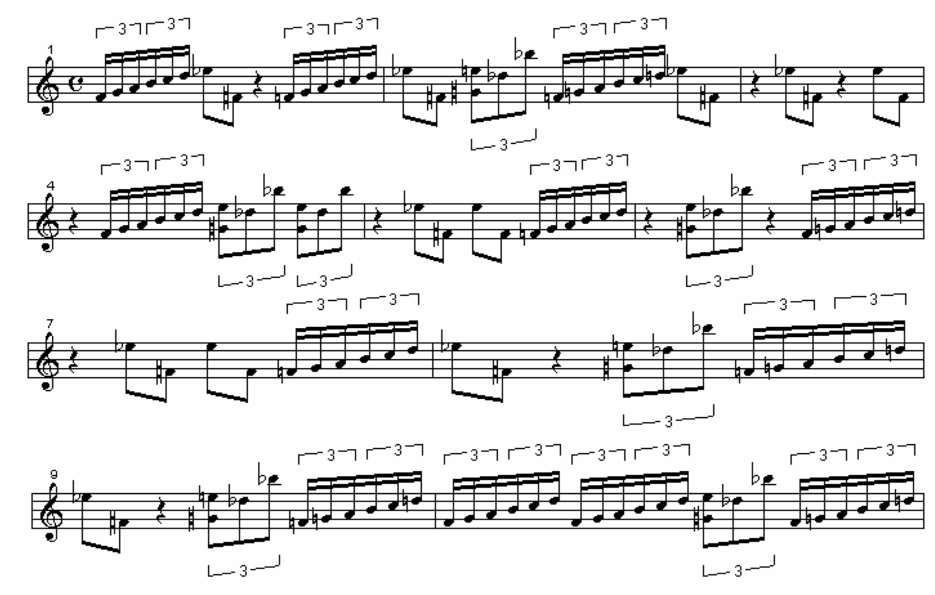
Generation 0 (randomly generated):

BCAB CDBC ACAC ABDC ACCB ADAB ACCB CADB CADB BBDB

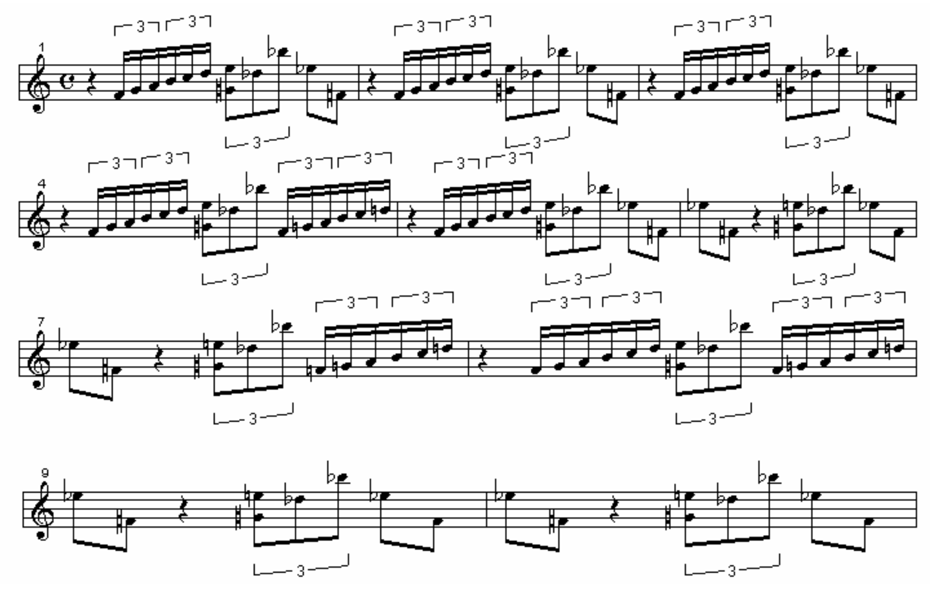
Generation	Chromosomes
1	ABDB ADAC ACCB BBDB ABDC CDBB CADB BCAC ADAC ABDC
2	BCAC CADB ABDB CADB CDBC CADC ABDC ABDB ABDC ABDC
3	ABDC ABDC ABDC CDBB ABDB CADC ABDC CADB ABDC BCAB
4	ABDC ABDC ABDC ABDB ABDC CADC CADB ABDB CADC CADC
5	CADB CADC ABDC ABDB ABDC ABDB ABDC CADC ABDC ABDC
6	ABDC CADC ABDC ABDC CADC CADB CADC ABDB CADC ABDC
7	CADB ABDB CADC ABDC ABDA ABDC ABDC ABDC CADB ABDC
8	ABDB CADC ABDB ABDC CADC ABDB CADC ABDC ABDB ABDC
9	ABDC ABDB ABDC CADB ADDC ABDB CADC CADC ABDC ABDC

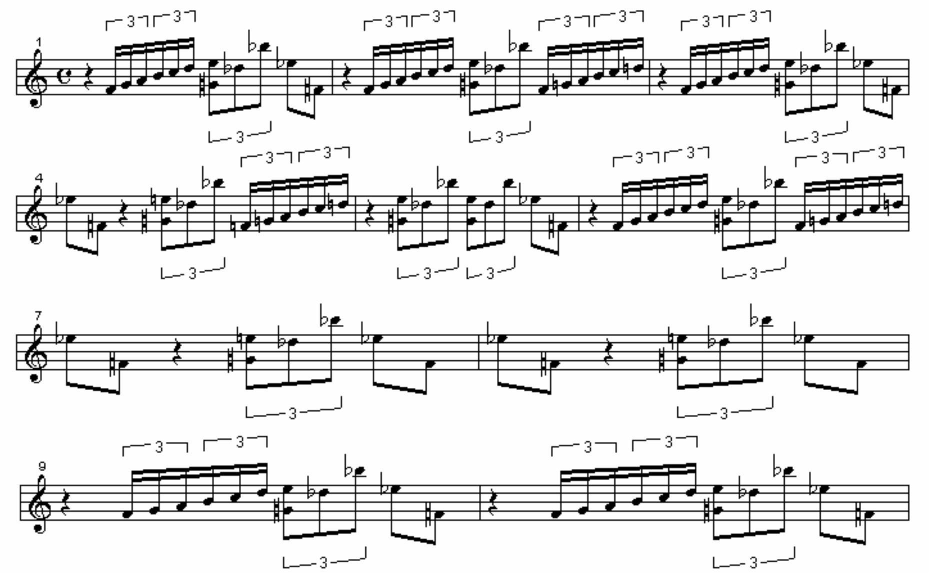
Note that because the genetic information and population were very limited (only four alleles per chromosome, and ten individuals per generation,) variation in individuals is severely limited too. Too little genetic variety and small populations cause inbreeding, just as in real life. However, these examples are sufficient to show how a genetic algorithm works.

What follows is a musical transcription of generations 0, 4 and 9:



Musical Example 45. Generation 0


Musical Example 46. Generation 4


Musical Example 47. Generation 9

Each measure represents a chromosome. Observe how generations evolve according to our fitness testing. For this example, chromosomes with a larger number of different alleles had more chance of surviving. For instance, chromosomes ACAC and BBDB in the initial population did not have a very high probability of reproducing and died. However, by generation 10 the population is dominated by chromosome ABDC, which has the highest fitness value.

Changing the fitness scheme allows us to create different variations on populations. As an example, we will now use the following fitness scheme on the same types of chromosomes.

1. Chromosomes which contain allele A two or more times get a fitness value of 0.30.
2. All other combinations get a value of 0.10.

Since allele A in our musical event space is a rest, we want to promote sequences in which silence is a dominant factor.

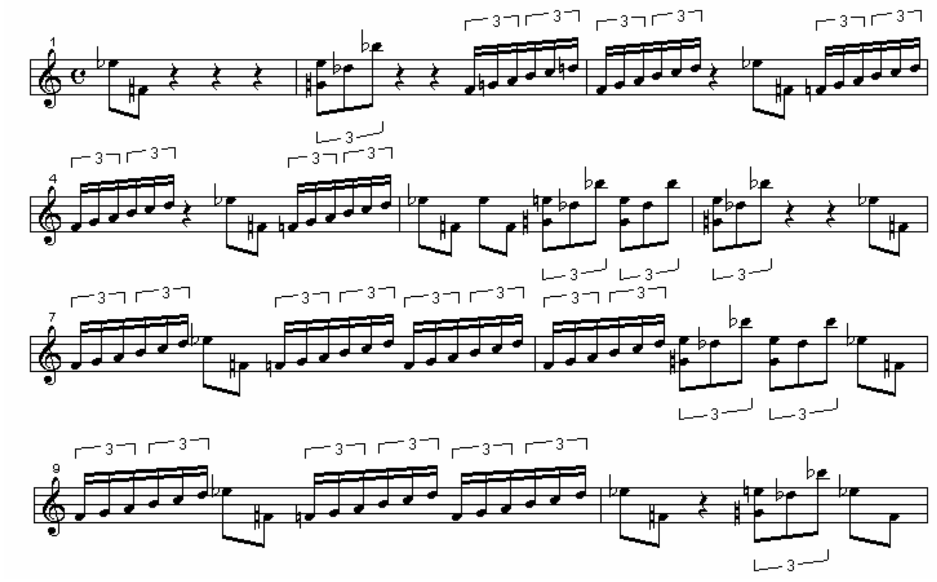
The generated populations are as follows:

Generation 0 (randomly generated):

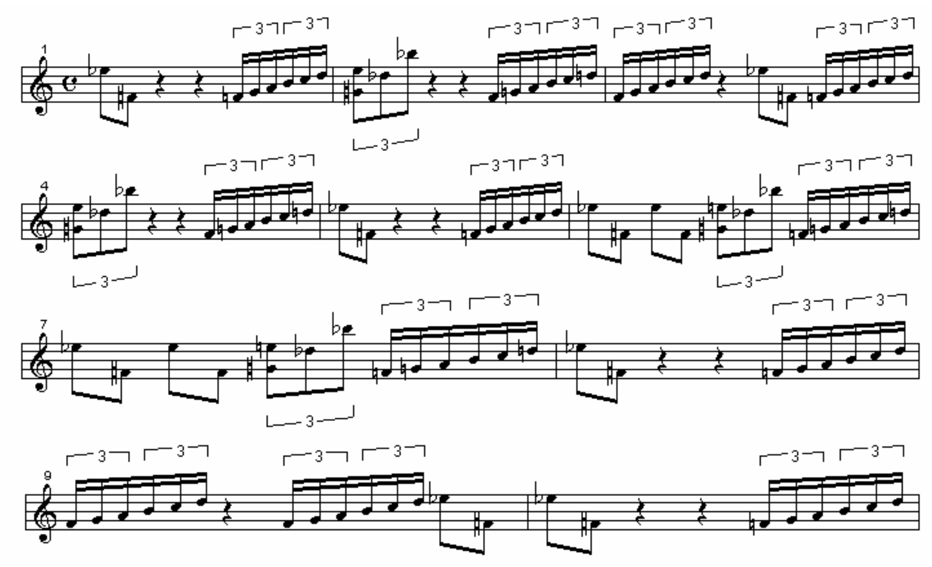
CAAA DAAB BACB BACB CCDD DAAC BCBB BDDC BCBB CADC

Generation	Chromosomes
1	BCBD CADB DAAB CAAB CCDC BACB CAAB BDDB CCDB CCDB
2	CAAB CCDB CAAB CCDB CCDB CCDB DAAB BACB BDDB DAAB
3	CCDB CCDB DAAB DAAB CAAB BACB CAAB DAAB CCDB CCDB
4	CAAB DAAB BACB DAAB CAAB CCDB CCDB CAAB BACB CAAB
5	BACB BACA DAAB CAAB DAAB DAAB DAAB CAAB CCDB DAAB
6	CAAB DAAB CAAB DAAB BACA DAAB DAAB DAAB DAAB CAAB
7	DAAB CAAB CAAB DAAB DAAB CAAB BAAB CAAB CAAB CAAD
8	DAAB CAAB CAAB DAAB CAAD CAAB CAAC DAAB CAAC CAAB
9	CAAB CAAC CAAC CAAC CAAB DAAB CAAB CAAD CAAD CAAB

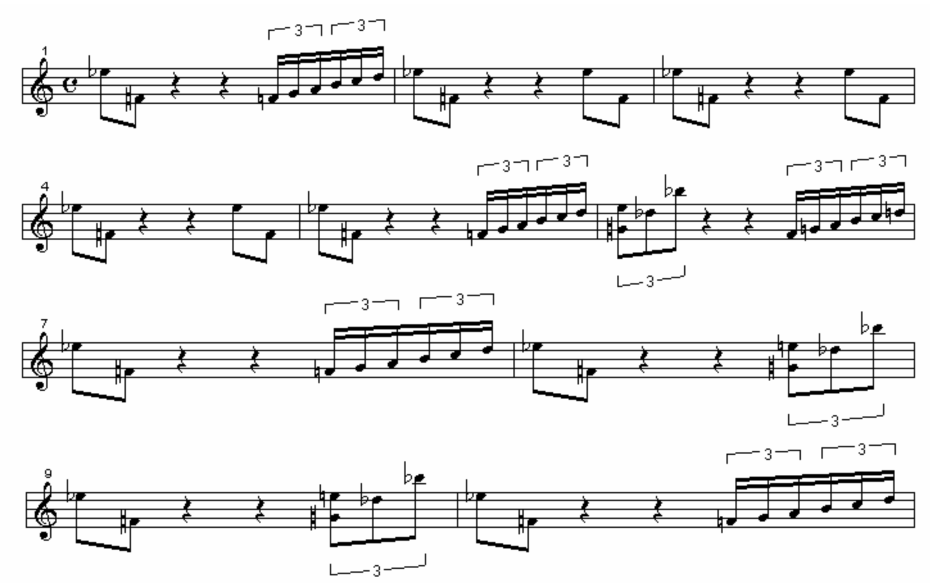
The musical transcription of generations 0, 4 and 9 follow:



Musical Example 48. Generation 0



Musical Example 49. Generation 4



Musical Example 50. Generation 9

It is evident how the music evolves according to our determined scheme. Once again, because of our limited resources, the population became very inbred by generation ten.

The true power of genetic algorithms only manifests itself when we deal with large populations of chromosomes, in which the genetic information (the variety of alleles) is large, too. Musically, this means larger event spaces with many members.

L-systems

L-systems belong to a branch of mathematics known as "formal grammars." They were first studied in 1968 by Aristid Lindenmayer——hence their name——as the basis for an axiomatic theory of development and as a tool for a realistic modeling of living organisms.[55] L-systems can be categorized in two basic types: deterministic context-free and context-sensitive. The basic formulation of a deterministic, context-free L-grammar is as follows:

$$G = \{A,\ P,\ \alpha\}$$

where A is the alphabet of the system (the set of all symbols, including the empty symbol or *null*, ε,) P is the finite set of substitution rules. A substitution rule (i, j) $\subset P$ is symbolized as $i \rightarrow j$, where $i \subset A$ and $j \subset A$; i is called the *predecessor* of the substitution rule and j the *successor* of the substitution rule. α, $\alpha \subset A$, is the axiom of the system, which can not be the null symbol ε. The substitution rules are applied to the axiom

[55]Lindenmayer, "Mathematical Models for Cellular Interactions," 280.

recursively, thus generating new sets of symbols——called *production strings*. The number of times the rules are applied to the subsequent production strings is known as the *recursion level*.

This process is best illustrated by a concrete example:

variables: X, Y Axiom: X	Substitution Rules: X→Y, Y→XY
Production strings (recursion level: 7)	
(0) X (Axiom)	
(1) Y	
(2) YX	
(3) YXY	
(4) XYYXY	
(5) YXYXYYXY	
(6) XYYXYYXYXYYXY	
(7) YXYXYYXYXYYXYYXYXYYXY	

Counting the number of symbols in each production string yields the following number series:

1,1,2,3,5,8,13,21...

which is the well-known Fibonacci series.

Context-sensitive L-systems include *stochastic* and *hierarchical* grammars, and *parametric* extensions. In these types of implementations the substitution rules take into account the state of surrounding, neighboring symbols when they are applied. This allows for a much more realistic and flexible modeling of organisms.

Stochastic techniques were introduced in L-systems to emulate Natures' non-deterministic, random growth patterns.

This implementation of L-systems maps a set of probabilities $\{p_1, p_2, p_3, \ldots p_n\}$ with a set of substitution rules $\{P_1, P_2, P_3, \ldots P_n\}$, including a symbol s, $s \subset A$, as the predecessor. For example,

$$\begin{aligned} P_1\!: s &\rightarrow (1/4)\ a \\ &\rightarrow (3/4)\ b \end{aligned}$$

means the symbol s has .25 (25%) probability of being substituted by a and .75 (75%) probability of being substituted by b. The sum of probabilities for any given symbol should always equal 1. Markovian processes can be represented by stochastic grammars. For instance, the same probability matrix for the Markov process discussed in the section *Markov Chains* (page 69) can be represented by the following stochastic grammar:

P_1: Event 1 $\rightarrow$ (1) Event 2
P_2: Event 2 $\rightarrow$ (.45) Event 1
$\rightarrow$ (.55) Event 2
P_3: Event 3 $\rightarrow$ (.28) Event 2
$\rightarrow$ (.16) Event 3
$\rightarrow$ (.28) Event 5
$\rightarrow$ (.28) Event 8
P_4: Event 4 $\rightarrow$ (.16) Event 1
$\rightarrow$ (.52) Event 4
$\rightarrow$ (.16) Event 6
$\rightarrow$ (.16) Event 9
P_5: Event 5 $\rightarrow$ (.50) Event 6
$\rightarrow$ (.50) Event 7
P_6: Event 6 $\rightarrow$ (.66) Event 5
$\rightarrow$ (.34) Event 6
P_7: Event 7 $\rightarrow$ (.20) Event 2
$\rightarrow$ (.70) Event 6
$\rightarrow$ (.10) Event 7
P_8: Event 8 $\rightarrow$ (.25) Event 1
$\rightarrow$ (.75) Event 9
P_9: Event 9 $\rightarrow$ (1) Event 10
P_{10}: Event 10 $\rightarrow$ (.70) Event 8
$\rightarrow$ (.30) Event 9

The selection of the axiom α is arbitrary and can be decided stochastically as well.

Parametric extensions involve the association of numerical parameters and mathematical expressions to the process of applying the substitution rules to symbols. This allows the incorporation of conditional statements that determine the application of a substitution rule over another based on numerical parameters. For instance:

$$\begin{array}{lll} P_1\text{:} & a\text{:} & x < 0 \,,\ a \rightarrow b \\ & & x = 0 \,,\ a \rightarrow c \\ & & x > 0 \,,\ a \rightarrow d \end{array}$$

the symbol a is replaced by b if parameter x is less than 0, by c if x is 0, or by d if x is greater than 0. The following example generates the famous Fibonacci series:

$P1$: $\mathrm{F}(i,\ j) \rightarrow \mathrm{F}(j,\ i+j)$

α: F(1,1)

Production string: F(1,1)$\rightarrow$F(1,2)$\rightarrow$F(2,3)$\rightarrow$F(3,5)$\rightarrow$ F(5,8)...

The formulation of hierarchical grammars is similar to that of deterministic, context-free grammars, with the key difference that the successors of substitution rules, instead of being symbols, may be entire grammars, with their corresponding sets of symbols and production rules. The use of hierarchical grammars, together with stochastic and parametric techniques allows the representation of very complex structures.

Because of their recursive nature——a task perfectly suited to the computer——L-systems were extensively studied by computer scientists soon after Lindenmayer proposed them, although mainly from the abstract mathematical point of view of formal grammar theory. However, the true power of L-systems manifests when a specific meaning is assigned to the symbols in the production strings.

The first concrete application of L-systems was in the field of computer graphics, as an aid in the representation of a wide variety of fractals and the in simulation of plant growth.[56]

[56]Lindenmayer and Prusinkiewicz, *The Algorithmic Beauty of Plants*.

The graphical interpretation of L-systems borrows the concept of "turtle graphics" from the LOGO programming language.[57] LOGO (based on the LISP programming language) was developed at the MIT and first appeared in 1967. LOGO was primarily designed as a learning tool. Turtle graphics (also known as *turtle geometry*[58]*)* became the most popular of LOGO environments. Turtle graphics is the computer screen version of a small robot that can move around on the floor according to directives sent to it from a computer. Turtle graphics in the computer screen are used to draw lines, shapes, pictures, etc.

The *state* of a turtle is represented mathematically by three numbers (i, j, α,) where (i, j) is the *position* of the turtle in the plane (in Cartesian coordinates,) and α is the angle determining the *direction* of the turtle.

The most basic implementation of turtle geometry in the computer screen has the following elements:

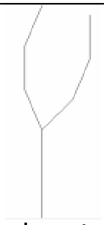
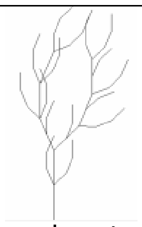
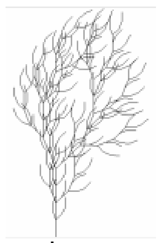
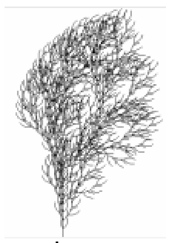
---

[57]The LOGO Foundation; available from http://el.www.media.mit.edu/logo-foundation/index.html; Internet; accessed 17 November 2000.

[58]H. Abelson and A. diSessa, *Turtle Geometry. The Computer as a Medium for Exploring Mathematics* (Cambridge, MA: The MIT Press, 1986).

- *Angle* α: the angle determining the turtle's direction.
- *Angle increment* ν.
- *Step size* δ: the distance (usually in screen pixels) the turtle travels in the screen when executing a drawing command.
- The *Forward* command (F): instructs the turtle to move forward a step of length δ. The position of the turtle changes to from (i, j) to (i', j'), where $i' = i + \delta*\cos(\nu)$ and $j' = j + \delta*\sin(\nu)$. A segment is drawn between coordinates (i, j) and (i', j').
- *f*: Like the *F* command, it moves the turtle to the new position (i', j'), but without drawing a line.
- +: Changes the state of the turtle to (i, j, $\alpha+\nu$).
- -: Changes the state of the turtle to (i, j, $\alpha-\nu$).

Assuming an initial state (i_0, j_0, α), where ν and δ are constant, then the picture set of lines drawn by the turtle according to a production string *S* is called the *turtle interpretation* of *S*. To illustrate this process an example follows:

Axiom: F Basic Angle α: 90° Angle Increment ν: 23°	Substitution Rules: $F=FF-[-F+F+F]+[+F-F-F]$
Graphical Representation	
 (Recursion Level 1)	 (Recursion Level 2)
 (Recursion Level 3)	 (Recursion Level 4)

Recursion level zero starts with the axiom as the current production string. Each successive iteration produces longer and more complicated production strings, which are then read sequentially as set of drawing commands to produce the image.

The production strings corresponding to the recursion levels in the above example have 8, 172, 1,388, and 11,116 symbols, respectively. It is evident the resemblance these graphics have with plants. They also manifest fractal structure and self-similarity at all scales.

Hierarchical, stochastic, and parametric L-systems have become an indispensable tool for modeling the complexity of plant morphology as closely as possible.

A musical interpretation of L-systems consists in assigning musical meaning to the symbols in the production strings.

The following discussion is based on David Sharp's program LMUSe[59], which interprets L-system symbols as musical parameters.

LMUSe, written for MS-DOS and Java, implements both context-free and context-sensitive as well as stochastic models of L-systems. LMUSe is a three-dimensional implementation of L-systems. In LMUSe, the turtle's state consists of the turtle's spatial position (*i*, *j*, and *k* coordinates,) a *forward vector* (*fi, fj, fz*) that holds the direction the turtle is facing, an *up vector* (*ui, uj, uz*) that holds the direction the "top" of the turtle's head is pointing to, and a *left vector* (*li, lj, lz*) that points to the direction to the turtle's left.

The turtle's state also holds the current length (δ,) line thickness, and basic turning angle (α.)

[59]LMUSe Ver. 0.7b, Released 24 December 1995, David Sharp; Available from http://www.geocities.com/Athens/Academy/8764/lmuse/lmuse.html; Internet.

In addition to the basic commands F, f, +, and -. LMUSe implements many others that affect the state and direction of the turtle. Here is a summary of LMUSe's direction and movement commands:

Direction commands:

Command	**Effect**
+	Turn left around *up* vector
+(x)	Turn x degrees left around *up* vector
-	Turn right around *up* vector
-(x)	Turn x degrees right around *up* vector
&	Pitch down around *left* vector
&(x)	Pitch x degrees down around *left* vector
^	Pitch up around *left* vector
^(x)	Pitch x degrees up around *left* vector
<	Roll counterclockwise around *forward* vector
<(x)	Roll x degrees left around *forward* vector
>	Roll clockwise around *forward* vector
>(x)	Roll x degrees clockwise around *forward* vector
|	Turn 180 degrees around *up* vector
%	Roll 180 degrees around *i* vector
$	Roll until horizontal
~	Turn/pitch/roll in a random direction
~(x)	Turn/pitch/roll in random direction to a maximum of x degrees

Movement/Play commands:

Command	**Effect**
F	Draw full length (δ) forward (Play)
F(x)	Draw x length forward (Play)
f	Move forward full length (Rest)
f(x)	Move forward x (Rest)
Z	Draw half length forward (Play)
Z(x)	Draw x length forward (Play)
z	Move forward half length (Rest)
z(x)	Move forward x (Rest)
g	Move forward full length (Rest)
g(x)	Move forward x (Rest)
.	Do not move

Increment/Decrement commands:

Command	**Effect**
"	Increment length (δ) (times 1.1)
'	Decrement length (times 1/1.1)
"(x)	Multiply length (times x)
'(x)	Multiply length (times 1/x)
;	Increment angle (α) (times 1.1)
:	Decrement angle (times 1/1.1)
:(x)	Multiply angle (times x)
;(x)	Multiply angle (times 1/x)
?	Increment thickness (times 1.4)
!	Decrement thickness (times 1/1.4)
?(x)	Multiply thickness (times x)
!(x)	Multiply thickness (times 1/x)

In addition to these commands LMUSe implements a "state stack" which serves as a kind of memory device preserving the turtle's state at any given time. The stack is a necessary structure so that the turtle can go back to a previous state, thus allowing for branching structures. Musically, the stack plays a key role for creating polyphonic textures. The stack commands are as follows:

Command	**Effect**
[	Push current state but not event time
]	Pop current state but not event time
{	Push current state and time
}	Pop current state and time
\	Push event time
/	Pop event time

In LMUSe, pitch, duration and volume (dynamics) can be mapped independently to the components of the position, forward, up, and left vectors as well as to the values of the line-thickness, state-length (δ), and draw length (length affected by modifiers.) LMUSe also implements commands specific to musical parameters only:

Command	**Effect**
t(x)	Transpose up (+x) or down (-x) by x semitones
t	Do not transpose
d(x)	Multiply note durations by x
d	Multiply note durations by 1.0(cancels d(x))
v(x)	Multiply note velocities (dynamics) by x
v	Multiply note velocities by 1.0 (cancels v(x))
Stack commands	**Effect**
T(x)	Push pitch transposition multiplier x onto pitch transposition stack
T	Pop transposition amount from pitch transposition stack
D(x)	Push duration multiplier x onto the duration multiplier stack
D	Pop duration multiplier from duration multiplier stack
V(x)	Push velocity multiplier x onto the velocity multiplier stack
V	Pop velocity multiplier from velocity multiplier stack

Pitch is mapped from predefined scales, which can be determined by the composer. Durations are mapped from a continuos scale, thus allowing very complex rhythms. MIDI velocities (dynamics) are also mapped from a continuous scale.

To illustrate the whole process an example follows.

Basic Parameters:

Recursion Level	5
Basic Angle	23°
Axiom	X

Substitution rules	X=F[+X]F[-X]+<&X
	F=F{F}

Line Thickness	50

Mapping scheme:

Parameter	**Mapped to**
Pitch	State Position (*i* component)
Duration	Drawn length
Dynamic	Forward vector (*fi* component)
Scale	Chromatic
Instrumentation	2 Flutes, Oboe, Clarinet, Cello

Production string to be mapped (recursion level 5):

```
F{F}{F{F}}{F{F}{F{F}}}{F{F}{F{F}}{F{F}{F{F}}}}[+F{F}{F{F}}{F{F}{F{F}}}[+F{F}{F{F}}[
+F{F}[+F[+X]F[-X]+<&X]F{F}[-F[+X]F[-X]+<&X]+<&F[+X]F[-X]+<&X]F{F}{F{F}}[-
F{F}[+F[+X]F[-X]+<&X]F{F}[-F[+X]F[-X]+<&X]+<&F[+X]F[-X]+<&X]+<&F{F}[+F[+X]F[-
X]+<&X]F{F}[-F[+X]F[-X]+<&X]+<&F[+X]F[-X]+<&X]F{F}{F{F}}{F{F}{F{F}}}[-
F{F}{F{F}}[+F{F}[+F[+X]F[-X]+<&X]F{F}[-F[+X]F[-X]+<&X]+<&F[+X]F[-
X]+<&X]F{F}{F{F}}[-F{F}[+F[+X]F[-X]+<&X]F{F}[-F[+X]F[-X]+<&X]+<&F[+X]F[-
X]+<&X]+<&F{F}[+F[+X]F[-X]+<&X]F{F}[-F[+X]F[-X]+<&X]+<&F[+X]F[-
X]+<&X]+<&F{F}{F{F}}[+F{F}[+F[+X]F[-X]+<&X]F{F}[-F[+X]F[-X]+<&X]+<&F[+X]F[-
X]+<&X]F{F}{F{F}}[-F{F}[+F[+X]F[-X]+<&X]F{F}[-F[+X]F[-X]+<&X]+<&F[+X]F[-
X]+<&X]+<&F{F}[+F[+X]F[-X]+<&X]F{F}[-F[+X]F[-X]+<&X]+<&F[+X]F[-
X]+<&X]F{F}{F{F}}{F{F}{F{F}}}{F{F}{F{F}}{F{F}{F{F}}}}[-
F{F}{F{F}}{F{F}{F{F}}}[+F{F}{F{F}}[+F{F}[+F[+X]F[-X]+<&X]F{F}[-F[+X]F[-
X]+<&X]+<&F[+X]F[-X]+<&X]F{F}{F{F}}[-F{F}[+F[+X]F[-X]+<&X]F{F}[-F[+X]F[-
X]+<&X]+<&F[+X]F[-X]+<&X]+<&F{F}[+F[+X]F[-X]+<&X]F{F}[-F[+X]F[-X]+<&X]+<&F[+X]F[-
X]+<&X]F{F}{F{F}}{F{F}{F{F}}}[-F{F}{F{F}}[+F{F}[+F[+X]F[-X]+<&X]F{F}[-F[+X]F[-
X]+<&X]+<&F[+X]F[-X]+<&X]F{F}{F{F}}[-F{F}[+F[+X]F[-X]+<&X]F{F}[-F[+X]F[-
X]+<&X]+<&F[+X]F[-X]+<&X]+<&F{F}[+F[+X]F[-X]+<&X]F{F}[-F[+X]F[-X]+<&X]+<&F[+X]F[-
X]+<&X]+<&F{F}{F{F}}[+F{F}[+F[+X]F[-X]+<&X]F{F}[-F[+X]F[-X]+<&X]+<&F[+X]F[-
X]+<&X]F{F}{F{F}}[-F{F}[+F[+X]F[-X]+<&X]F{F}[-F[+X]F[-X]+<&X]+<&F[+X]F[-
X]+<&X]+<&F{F}[+F[+X]F[-X]+<&X]F{F}[-F[+X]F[-X]+<&X]+<&F[+X]F[-
X]+<&X]+<&F{F}{F{F}}{F{F}{F{F}}}[+F{F}{F{F}}[+F{F}[+F[+X]F[-X]+<&X]F{F}[-F[+X]F[-
X]+<&X]+<&F[+X]F[-X]+<&X]F{F}{F{F}}[-F{F}[+F[+X]F[-X]+<&X]F{F}[-F[+X]F[-
X]+<&X]+<&F[+X]F[-X]+<&X]+<&F{F}[+F[+X]F[-X]+<&X]F{F}[-F[+X]F[-X]+<&X]+<&F[+X]F[-
X]+<&X]F{F}{F{F}}{F{F}{F{F}}}[-F{F}{F{F}}[+F{F}[+F[+X]F[-X]+<&X]F{F}[-F[+X]F[-
X]+<&X]+<&F[+X]F[-X]+<&X]F{F}{F{F}}[-F{F}[+F[+X]F[-X]+<&X]F{F}[-F[+X]F[-
X]+<&X]+<&F[+X]F[-X]+<&X]+<&F{F}[+F[+X]F[-X]+<&X]F{F}[-F[+X]F[-X]+<&X]+<&F[+X]F[-
X]+<&X]+<&F{F}{F{F}}[+F{F}[+F[+X]F[-X]+<&X]F{F}[-F[+X]F[-X]+<&X]+<&F[+X]F[-
X]+<&X]F{F}{F{F}}[-F{F}[+F[+X]F[-X]+<&X]F{F}[-F[+X]F[-X]+<&X]+<&F[+X]F[-
X]+<&X]+<&F{F}[+F[+X]F[-X]+<&X]F{F}[-F[+X]F[-X]+<&X]+<&F[+X]F[-X]+<&X
```

The *graphical* representation of the above production string is as follows:

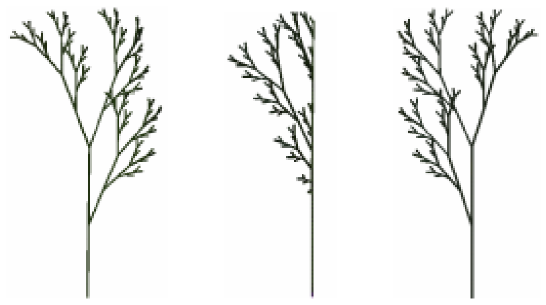
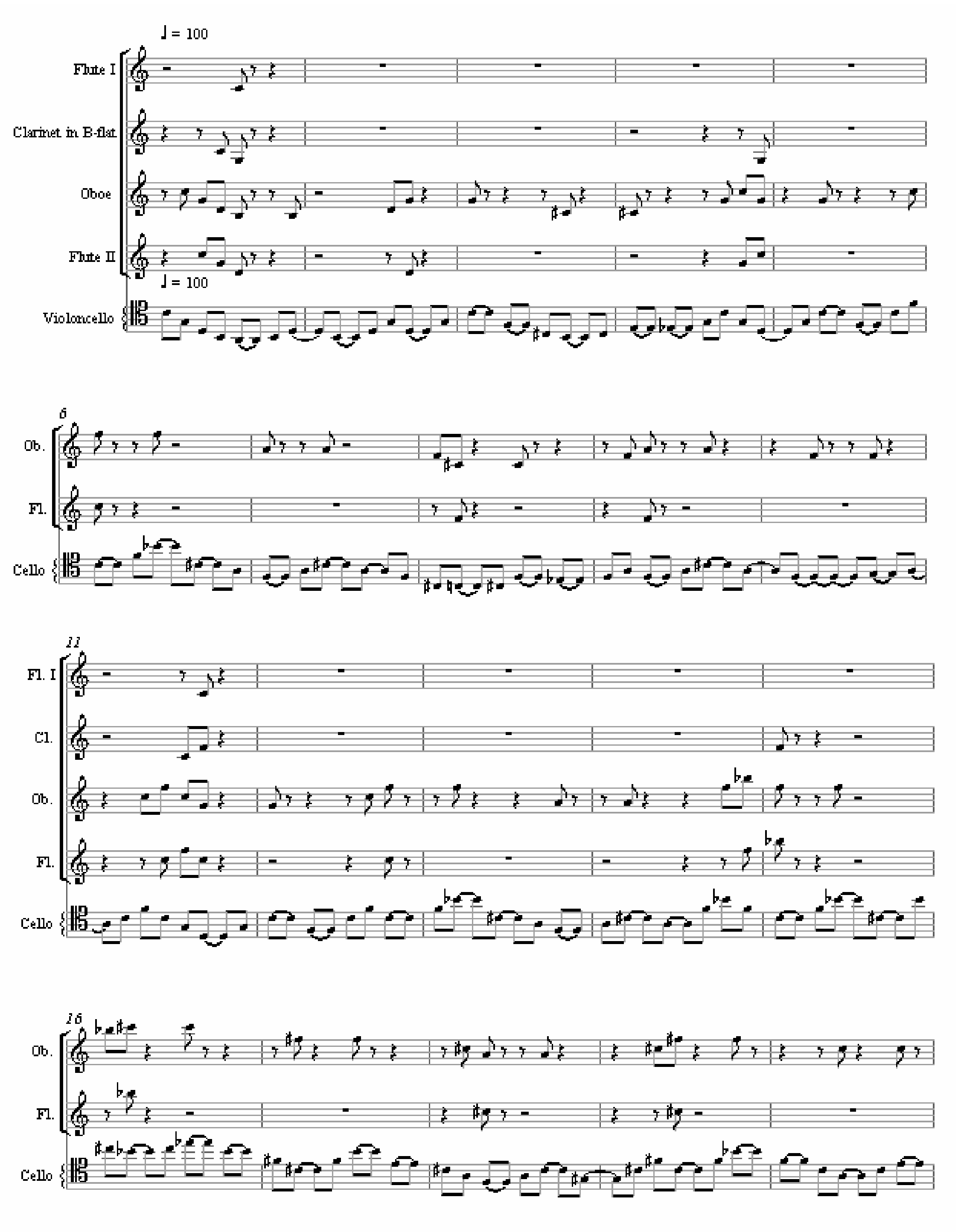

(front) (side) (back)

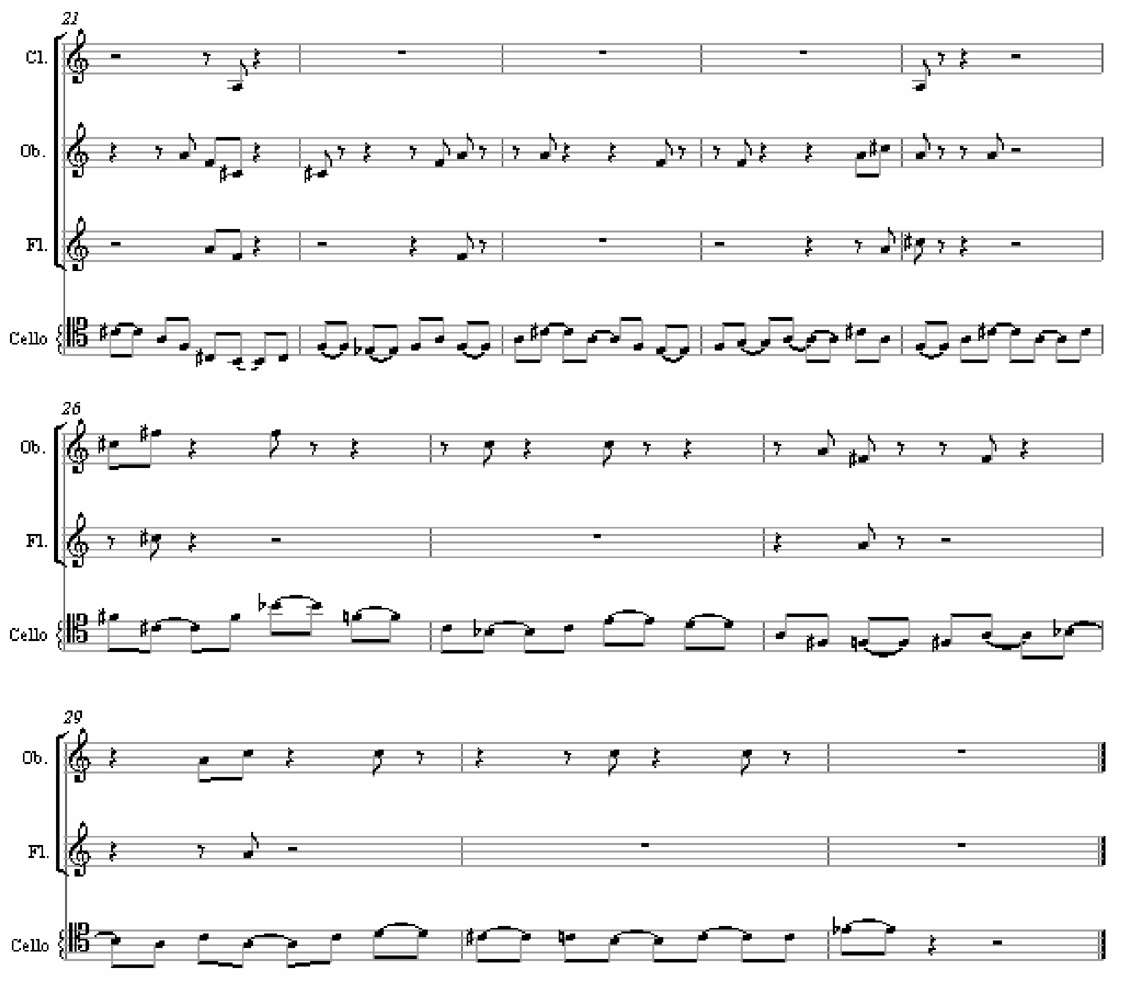
Figure 31. L-system graphical rendition.

Three different viewpoints are shown in order to appreciate the three-dimensional structure. Note the self-similar branching.

The musical rendition of this production string, according to our mapping scheme is as follows:



Musical Example 51. Music from L-systems



Musical Example 51 (continued)

Note how the five levels of branching in the graphical rendition are mapped to five-voice polyphony. The first level of branching is mapped to the cello, the second to the oboe, the third to the flute II, the fourth to the clarinet and the fifth to the flute I. The choice of instruments was the composer's (this writer) decision. The rhythmic structure was kept simple because duration was mapped to a parameter (drawn length) that remained constant in the production string. Consecutively equal pitches are tied.

It is remarkable how the structure of the graphical rendition is preserved also in the musical mapping. For instance, note how the graphical interpretation has three main structures (labeled 1, 2, and 3):

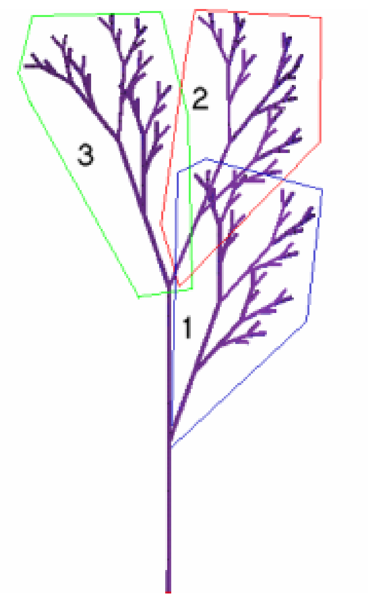
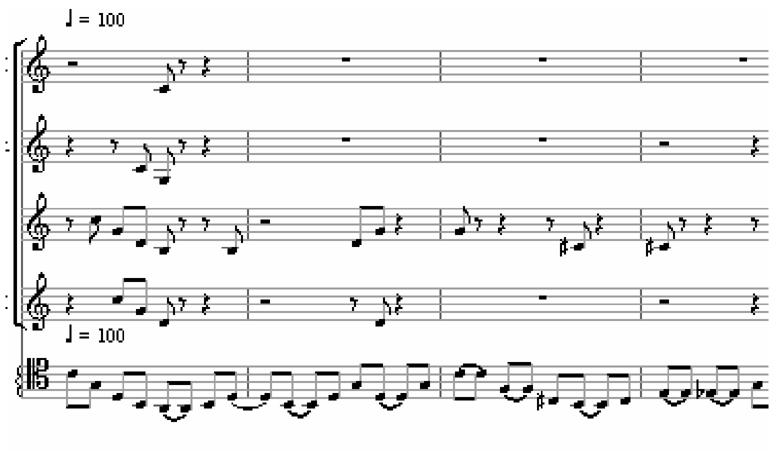


Figure 32. Structure of L-system model.

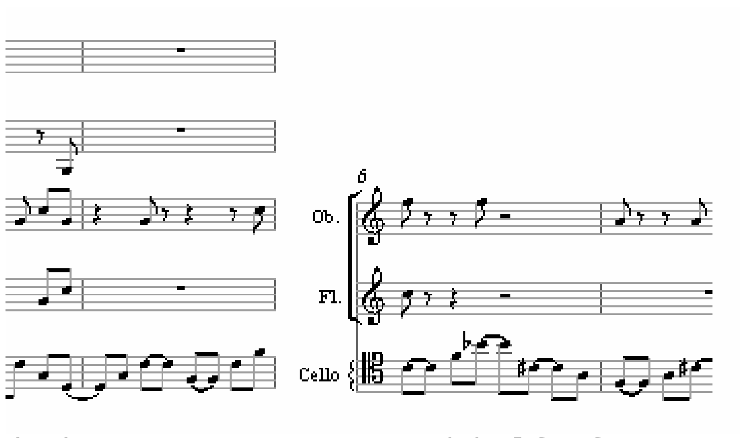
Each of these structures are constituted, in turn, of three sub-structures which are, in fact, scaled down copies of the whole. This process of structures within structures continues until the fifth level of recursion.

The three main structures correspond exactly to mm.

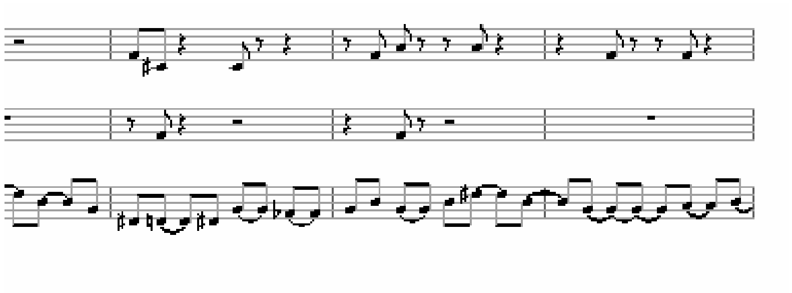
1-10, 11-20, and 21-30 in the score, respectively. The second level of structure is also evident in the musical transcription. Each of the three main sections can also be structured into three subsections. The following examples show the three sub-structures of the first section (mm. 1-10.)



(sub-structure 1; mm. 1-3 and first half of m. 4)



(sub-structure 2; second half of m. 4, mm. 5-6 and first half of m. 7)



(sub-structure 3; second half of m. 7 and mm. 8-10)

An even closer inspection of these sub-structures reveals yet smaller layers of structure, which go down almost to the note-to-note level. The self-similarity of these structures (both in the graphical image and in the musical rendering) are not exact, but only approximate. For this particular example, this is a consequence of the substitution rules and angle chosen. In a more elaborate example, *statistical* self-similarity can be achieved on a much higher level by using stochastic rules. Statistical self-similarity plays an important role in the musical rendition of L-systems, because it yields more variation and makes the music less monotonous and expected.

L-systems are a vast field for musical experimentation. S. Mason and M. Saffle have studied their application to music composition.[60] Some composers, such as Gary Lee Nelson in his *Summer Song* for solo flute, have

[60]S. Mason and M. Saffle, "L-Systems, Melodies and Musical Structure," *Leonardo Music Journal* 4 (1994): 53-58.

already utilized L-systems in their work. By manipulating production rules and mapping schemes, the composer is given an almost inexhaustible source for musical inspiration.

CHAPTER IV

Software and computer resources for algorithmic composition.

The following are software programs, many available in the Internet, that implement most of the algorithmic procedures discussed in previous chapters. The great majority of these programs write MIDI files from the algorithmic processes they perform, which can be later loaded in notation and sequencer programs for further refinement of the musical material. MIDI (an acronym for Musical Instrument Digital Interface) is a communications protocol of the music industry that allows instruments and sequencers (or computers running sequencer software) to communicate with each other to play and record music. Each MIDI interface has 16 multitimbral channels to which all MIDI information (pitch, duration, dynamic, instrument, etc.) is directed. In addition, each channel is polyphonic, the number of voices depending on the quality of the MIDI board. A MIDI file is a computer file format that stores MIDI information (pitches, tempo, durations, instruments, etc.) in a very compact manner. These files can be loaded into music notation or sequencing programs for manipulation and editing.

FractMus 2000

Author: Gustavo Diaz-Jerez.

Freeware.

Platform: Windows

Availability: http://www.geocities.com/fractmus

MIDI file support: YES

Description:

This program uses twelve algorithms from chaotic dynamics (Logistic map, Gingerbread man attractor, Hènon attractor, Lorenz attractor, Hopalong and Martin attractors,) noise (1/f noise,) cellular automata (Wolfram Cellular Automata) and number theory (Morse-Thue sequence, Earthworm sequence, 3n+1 numbers.) It also includes a linear random number generator which can be useful for stochastic processes. Algorithms can be assigned to all MIDI voices independently. All musical parameters can be controlled by the user (scales, durations, dynamics, instruments, etc.,) which allow a wide variety of mappings. Independent sections (events,) each including their own set of algorithms, musical parameters and mappings, can be defined within the program.

A Musical Generator

Author: MuSoft Builders.

Shareware.

Platform: Windows

Availability: http://www.musoft-builders.com

MIDI file support: YES

Description:

Musical material is drawn from a great number of sources, including chaotic maps (Hopalong, Lorenz, Martin, Gingerbread man, Henon, and Polar attractors); fractals (Mandelbrot, Julia, Barnsley, Lambda, Newton, and Spider sets); noise (white, Brownian, fractional Brownian); Lindenmayer L-systems, mathematical constants (such as π and e); images, text, and numerical data provided by the user as well as mathematical functions defined by the user. Algorithms can be assigned independently to each MIDI voice. Only one event is allowed per composition (one set of algorithms mapped to musical parameters.)

Art Song and MusicLab I - Music from Chaos

Author: David Strohbeen.

Shareware.

Platform: Windows

Availability: http://www.fractalmusiclab.com

MIDI file support: YES

Description:

Strange attractors from chaotic dynamics using Iterated Function Systems (fractals) and Quadratic Equations as well as images provided by the user. The user has control over all musical parameters, such as durations, dynamics, range of pitches, etc. Parameters can be changed in real time while playing. In addition, new functions can be defined by the user within the program.

Gingerbread

Author: Phil Thompson.

Freeware.

Platform: Windows

Availability: http://www.organised-chaos.com/oc/ginger/ginger.html

MIDI file support: YES

Description:

This program maps Mandelbrot and Julia Set fractals functions to musical parameters. Each MIDI voice has its own graphical window where the user can manipulate and zoom the fractal image. The user can then click on a particular section of the image from where the music is then generated. The program allows the composer complete control of musical parameters such as scales, dynamics, note durations, tempo, etc.

Mandelbrot music program

Author: Yo Kubota.

Freeware.

Platform: Windows

Availability: http://www.fin.ne.jp/~yokubota/

MIDI file support: YES

Description:

Based in the Mandelbrot set exclusively. The program maps iteration values to user defined musical parameters.

Chaos von Eschenbach

Freeware.

Platform: Windows

Availability: http://www.cisnet.or.jp/home/magari/

MIDI file support: YES

Description:

The Mandelbrot set, Julia sets, white noise and logistic equation. Wide variety of mappings available.

FMusic

Author: David H. Singer.

Freeware.

Platform: Windows

Availability: http://members.aol.com/dsinger594/caman

MIDI file support: YES

Description:

This program maps Lindenmayer L-Systems and one-dimensional cellular automata to musical parameters defined by the user. It generates five-voice polyphony only. The user has complete control on how the mapping is performed.

Tangent and QuasiFractalComposer

Author: Paul Whally.

Shareware/Freeware.

Platform: Windows

Availability: http://www.randomtunes.com

MIDI file support: YES

Description:

Stochastic and deterministic methods. Parameters for these processes are decided by the user. Instrumentation, scales, note durations, dynamics, and other musical parameters can also be set by the user.

Lorenz

Author: Jose Navarro.

Shareware.

Platform: Windows

Availability: http://globalia.net/janc/

MIDI file support: YES

Description:

This program employs the Lorenz attractor function to generate sequences.

CAMUS and CAMUS 3D

Author: E. Miranda, Kenny McAlpine and Stuart Hoggar (CAMUS,) Glasgow University, Centre for Music Technology (CAMUS 3D.)

Freeware.

Platform: Windows

Availability: http://www.maths.gla.ac.uk/~km/dsysmus.htm

MIDI file support: YES

Description:

Two- and three-dimensional cellular automata.

MusiNum – The music in the numbers.

Author: Lars Kindermann.

Freeware.

Platform: Windows

Availability: http://www.forwiss.uni-erlangen.de/~kinderma/musinum

MIDI file support: YES

Description:

Morse-Thue number sequence and numerical data provided by the user.

Make Prime Music

Author: Armand Turpel.

Freeware.

Platform: MS-DOS

Availability: http://www2.vo.lu/homepages/armand/index.html

MIDI file support: YES

Description:

Prime number series. Three different mappings of the series are allowed. Four-part counterpoint is generated.

Genetic Spectrum Modeling Program

Author: Ray Jurgens.

Freeware.

Platform: MS-DOS

Availability: http://autoinfo.smartlink.net/ray/

MIDI file support: YES

Description:

This program combines cellular automata, genetic algorithms and 1/f noise to generate fractal melodies. The user has control over the scales used, number of cells (for cellular automata processes,) number of generations to run, different probability settings such as mutation rate, etc. The program randomly generates a series of notes that are completely uncorrelated. The series of notes represent a group of cells that obey Cellular Automata rules. Each generation is genetically modified to turn the cells into a specific spectral shape which mimics 1/f noise. This process involves random mutation of the cells. This program generates only one voice per set of parameters.

The Well Tempered Fractal

Author: Robert Greenhouse.

Shareware.

Platform: MS-DOS

Availability: http://www-ks.rus.uni-stuttgart.de/people/schulz/fmusic/wtf

MIDI file support: YES, but non-standard

Description:

Chaotic attractors (Hopalong, KAM torus, Mira, Chebychev functions, Julia maps, etc.) The user controls musical parameters such as scale, range of pitches, durations, etc.

LMuse

Author: David Sharp.

Freeware.

Platform: MS-DOS, Java (All platforms)

Availability:

http://www.geocities.com/Athens/Academy/8764/lmuse/lmuse.html

MIDI file support: YES

Description:

Lindenmayer L-Systems. The program generates a graphical representation of the l-system rule set from which the music is derived. The user can define many different types of mapping. One composition per rule set is generated. The user can only decide the scale from which the notes are drawn. Other parameters are decided by the program according to the l-system rule set.

LoShuMusic and FibonacciBlues

Shareware.

Platform: Macintosh

Availability: ftp://mirror.apple.com/mirrors/Info-Mac.Archive/art/fibonacci-blues-02.hqx and ftp://mirror.apple.com/mirrors/Info-Mac.Archive/art/loshu-music-02.hqx

MIDI file support: YES

Description:

Fibonacci number sequence and aleatoric (random) procedures.

SoftStep

Author: Algorithmic Arts.

Commercial.

Platform: Windows

Availability: http://www.algoart.com/software/index.htm

MIDI file support: YES

Description:

Fractals (Mandelbrot and Julia sets,) chaotic attractors (Mira, Martin, Henon, Chebychev, Hopalong, etc.,) stochastic processes, number-theory, user defined numerical data, etc.

Symbolic Composer

Author: Tonality Systems.

Commercial.

Platform: Macintosh

Availability: http://symcom.hypermart.net

MIDI file support: YES

Description:

Stochastic processes, chaotic maps, fractals, number-theory, etc. Many algorithmic processes not included within the program can be implemented (programmed) using its own proprietary language.

Random Phase Music Generator

Author: Tak-Shing Chan.

Freeware.

Platform: Linux.

Availability:

http://www.engr.newpaltz.edu/~chan12/phase.html

MIDI file support: YES

Description:

Music is generated through phasing of randomly generated patterns——the process of looping the same pattern at slightly different speeds——, so they will slowly shift out of synchronization. The user has control over musical parameters such as scales, note durations, tempo, as well as the length of the repeating pattern.

Csound

Freeware.

Platform: All

Availability: http://mitpress.mit.edu/e-books/csound/fpage/getCs/getCs.html

MIDI file support: YES

Description:

Csound is a open computer language dedicated to music composition and sound design. It is so versatile that it allows the implementation (programming) of any algorithmic process one can think. In addition to the possibility of programming user-defined algorithms, there are hundreds of third-party plug-ins (finished programs) available to everyone. These include stochastic processes, fractals, number-theory, etc.

Silence

Author: Michael Gogins

Freeware.

Platform: All

Availability: http://www.pipeline.com/~gogins/

MIDI file support: YES

Description:

Lindenmayer L-systems, iterated function systems, chaotic attractors. This programs requires the installation of the CSound program.

AC Toolbox

Author: Paul Berg

Freeware.

Platform: Windows

Availability: http://www.koncon.nl/ACToolbox/

MIDI file support: YES

Description:

Creation of musical event spaces through stochastic functions, chaotic systems, recursion, etc. This program also generates CSound and Opcode MAX files.

Common Music

Author: Rick Taube

Freeware.

Platform: Mac, DOS, Windows, SGI.

Availability: http://www-ccrma.stanford.edu/CCRMA/Software/cm/cm.html

MIDI file support: YES

Description:

Common Music is an object-oriented music composition environment. Many algorithmic procedures can be implemented in Common Music: stochastic processes, chaotic systems, number theory, etc.

Nyquist

Freeware.

Platform: Mac, Windows, Amiga.

Availability:

http://www.cs.cmu.edu/afs/cs.cmu.edu/project/music/web/music.software.html

MIDI file support: YES

Description:

Nyquist is a sound synthesis and composition language. All kinds of algorithmic processes can be implemented in Nyquist.

KeyIt

Freeware.

Platform: Windows, Linux.

Availability: http://thompsonresidence.com/keykit/

MIDI file support: YES

Description:

KeyIt is a programming language and graphical user interface for MIDI. The user can implement many algorithmic processes, including stochastic processes, fractals, chaotic maps, etc.

ArborRhythms

Author: Alec Rogers

Shareware.

Platform: Windows

Availability: http://www.arborrhythms.com

MIDI file support: YES

Description:

Random processes and numerical algorithms.

Real Time Composition Library

Author: Karlheinz Essl

Freeware.

Platform: Mac.

Availability: http://www.essl.at/works/rtc.html

MIDI file support: YES

Description:

RTC-Lib is a collection of plug-ins for Opcode's MAX system. It implements algorithmic procedures such as stochastic processes, numerical functions, etc.

The Hierarchical Music Specification Language

Author: Phil Burk, Larry Polansky and David Rosenboom

Commercial.

Platform: Mac.

Availability: http://www.softsynth.com/hmsl/

MIDI file support: YES

Description:

HMSL is a programming language for experimental music composition. It allows the user to implement custom algorithmic processes such as stochastic, fractal, numerical, etc.

MAX

Author: Miller Puckette, ported by D. Zicarelli.

Commercial, Opcode Systems, Inc., IRCAM, and Cycling '74.

Platform: Mac.

Availability: http://www.opcode.com/products/max/

MIDI file support: YES

Description:

MAX is programming environment for music. Its object-oriented programming environment can create an infinite variety of customized applications. Any imaginable algorithmic process can be implemented in MAX.

Cmix and RTcmix (Real Time Cmix)

Author: Paul Lansky, Lance Graf, Dave Madole, Brad Garton, Doug Scott, and Eric Lyon.

Freeware.

Platform: IRIX (Silicon Graphics,) Linux.

Availability: http://music.columbia.edu/cmix/

MIDI file support: YES

Description:

Cmix is a computer language designed for sound synthesis as well as music composition. It is basically a library of C routines. RTcmix is the "real time" version of Cmix, where control of instruments can be done in real time. Any algorithmic procedure can be implemented using the RTcmix language.

Rubato

Author: Chris Tham

Freeware.

Platform: MS-DOS.

Availability:

http://members.value.com.au/christie/rub_full.html

MIDI file support: YES

Description:

Rubato is a programming, notation, and performing environment for music. It allows the implementation of algorithmic procedures.

LScore

Author: Senast Andrad:

Freeware.

Platform: All

Availability: http://listen.to/algo-comp

MIDI file support: YES

Description:

LScore is an implementation of L-systems for Csound. Requires the installation of the CSound package.

The above list, although exhaustive at the time of this writing, is by no means complete. Every day new programs and updates of existing ones become available in the Internet. Most of the URLs provided for the above programs contain links to related sites of interest. They should be consulted from time to time for new programs and updates of existing ones. Some of these URLs may change in the future. In this case, it is generally sufficient to search the program with one of the standard Internet search engines (Hotbot, Lycos, Altavista, etc.) with the program's name as the search query.

CHAPTER V

Source Code Listings.

The following are ANSI C and C++ routines that implement specific routines valuable for algorithmic composition.

Random Number Generator

The following source code implements a linear random number generator function. It has a period greater than 10^{18}. It is useful for many algorithmic processes that require strings of random numbers. The function RanDev() returns a pseudo-random number between 0 and 1. The variable "Seed" value should be initialized——using for instance the time() function, Seed = time(NULL)——every time the function is called.

```
/*** Start of code ***/

const long IM1 = 2147483563L;
const long IM2 = 2147483399L;
const long IMM1 = IM1 - 1L;
const long IA1 = 40014L;
const long IA2 = 40692L;
const long IQ1 = 53668L;
const long IQ2 = 52774L;
const long IR1 = 12211L;
const long IR2 = 3791L;
const long NTAB = 32L;
const long NDIV = 1L+IMM1/long(NTAB);

const float RNMX = 1.0f - 0.0000001f;
const float AM = 1.0f/2147483563.0f;
long Seed;


float RanDev(); /* function prototype */

/* Returns a psuedo-random number between 0 and 1 every
     time it is called
*/

float RanDev()
{
     long j,k;
     static long idum2 = 123456789l;
     static long iy = 0l;
     static long iv[32L];
     float temp;

     if (Seed<=0L)
     {
          if(-Seed<1L)
               Seed = 1L;
          else
               Seed = -Seed;

          idum2 = Seed;

          for(j = NTAB+7; j>=0; --j)
          {
               k = Seed/IQ1;
               Seed = IA1*(Seed-k*IQ1) - k * IR1;

               if(Seed<0L)
                    Seed += IM1;
               if(j<NTAB)
                    iv[size_t(j)] = Seed;
          }

          iy = iv[0];
     }
     k = Seed/IQ1;

     Seed = IA1*(Seed-k*IQ1)-k*IR1;

     if(Seed<0L)
          Seed +=IM1;

     k = idum2/IQ2;

     idum2 = IA2*(idum2-k*IQ2)-k*IR2;

     if(idum2<0L)
          idum2 +=IM2;

     j = iy/NDIV;
     iy = iv[size_t(j)] - idum2;
     iv[size_t(j)] = Seed;

     if(iy<1L)
          iy += IMM1;

     temp = AM*float(iy);

     if(temp>RNMX)
          return RNMX;
     else
          return temp;
}


/********* end of RanDev() function code *********/
```

Gaussian Random Number Generator

The following source code implements a Gaussian pseudo-random number generator. When called, the function gaussrand() returns pseudo-random numbers whose probability distribution is Gaussian:

```
/******* Start of code ***********/

#include <stdlib.h>
#include <math.h>

double gaussrand();


double gaussrand()
{
     static double V1, V2, S;
     static int phase = 0;
     double X;

     if(phase == 0) {
          do {
               double U1 = (double)rand() / RAND_MAX;
               double U2 = (double)rand() / RAND_MAX;

               V1 = 2 * U1 - 1;
               V2 = 2 * U2 - 1;
               S = V1 * V1 + V2 * V2;
               } while(S >= 1 || S == 0);

          X = V1 * sqrt(-2 * log(S) / S);
     } else
          X = V2 * sqrt(-2 * log(S) / S);

     phase = 1 - phase;

     return X;
}
```

1/f Noise

The following C++ class implements the Voss algorithm for the generation of 1/f noise.[1] It is invaluable for using 1/f noise in algorithmic composition.

Once an object of the class PinkNumber is created, calling the member function GetNextValue() returns values that resemble the spectral density of 1/f noise.

```
/******* Start of code *********/

#include <iostream>
#include <stdlib.h>
class PinkNumber
{
private:
  int max_key;
  int key;
  unsigned int white_values[5];
  unsigned int range;
public:
  PinkNumber(unsigned int range = 128)
    {
      max_key = 0x1f; // Five bits set
      this->range = range;
      key = 0;
      for (int i = 0; i < 5; i++)
 white_values[i] = rand() % (range/5);
    }
  int GetNextValue()
    {
      int last_key = key;
      unsigned int sum;
```

[1]An improved version of this algorithm——not included here because of its length——can be found at http://www.firstpr.com.au/dsp/pink-noise/phil_burk_19990905_patest_pink.c; Internet. Accessed February 10, 2001.

```
           key++;
      if (key > max_key)
 key = 0;
      /* Exclusive-Or previous value with current
value. This gives a list of bits that have
changed. */

      int diff = last_key ^ key;
      sum = 0;
      for (int i = 0; i < 5; i++)
 {
   // If bit changed get new random number for
corresponding
   // white_value
   if (diff & (1 << i))
     white_values[i] = rand() % (range/5);
   sum += white_values[i];
 }
      return sum;
    }
};

     /****** end of code *******/
```

APPENDIX

This appendix is a description of the mapping process and an analysis of the music generated from *The Game of Life*, a two-dimensional cellular automaton described in the section *Two-dimensional Cellular Automata* in CHAPTER III.

The cellular automaton will evolve in a square matrix of 40 by 40 cells. The matrix has a toroidal geometry, that is, cells in the top row neighbor cells in the bottom row, and cells in the rightmost column neighbor cells in the leftmost column. The initial cell configuration (generation 0) is as follows:

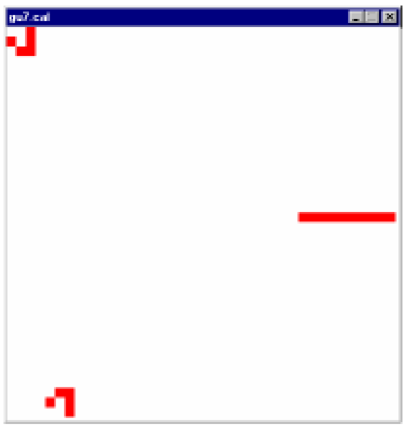


Figure 1. Initial cell configuration

This particular cell configuration was chosen because it demonstrates clearly pattern propagation and other characteristics of the cellular automaton. It contains two gliders (top left and bottom left, see Fig. 1) and a line of ten cells (middle right,) which is a parent (or predecessor) of a well-known object called the *pentadecathlon*. The pentadecathlon is a period-15 oscillator:

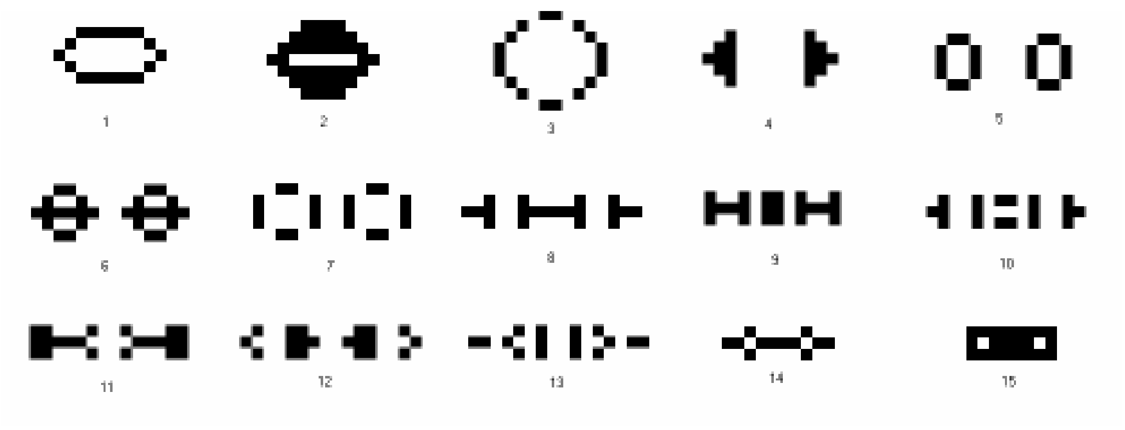


Figure 2. Pentadecathlon (period 15)

Other objects which will be important in the analysis are the *block, loaf,* and *toad*:

Figure 3. Block (stable or *still life*)

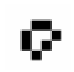


Figure 4. Loaf (stable)

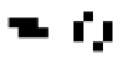


Figure 5. Toad (period-2 oscillator)

Three hundred generations in total were computed and mapped. The system achieves a stable, unchanging state after generation 289.

What follows are snapshots of the evolution of the cellular automaton every five generations.

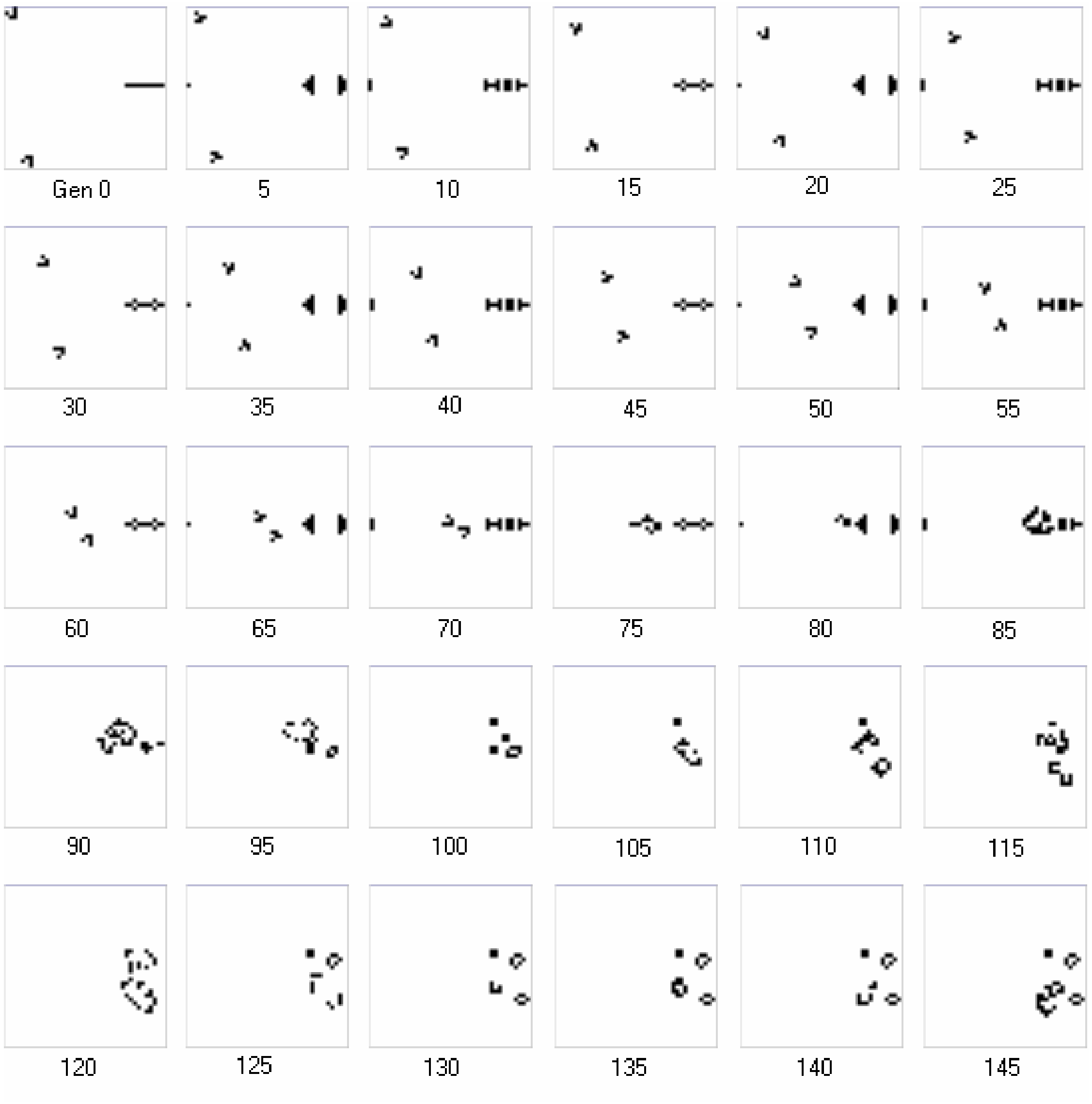


Figure 6. Evolution of the Automaton. Generations 0-145

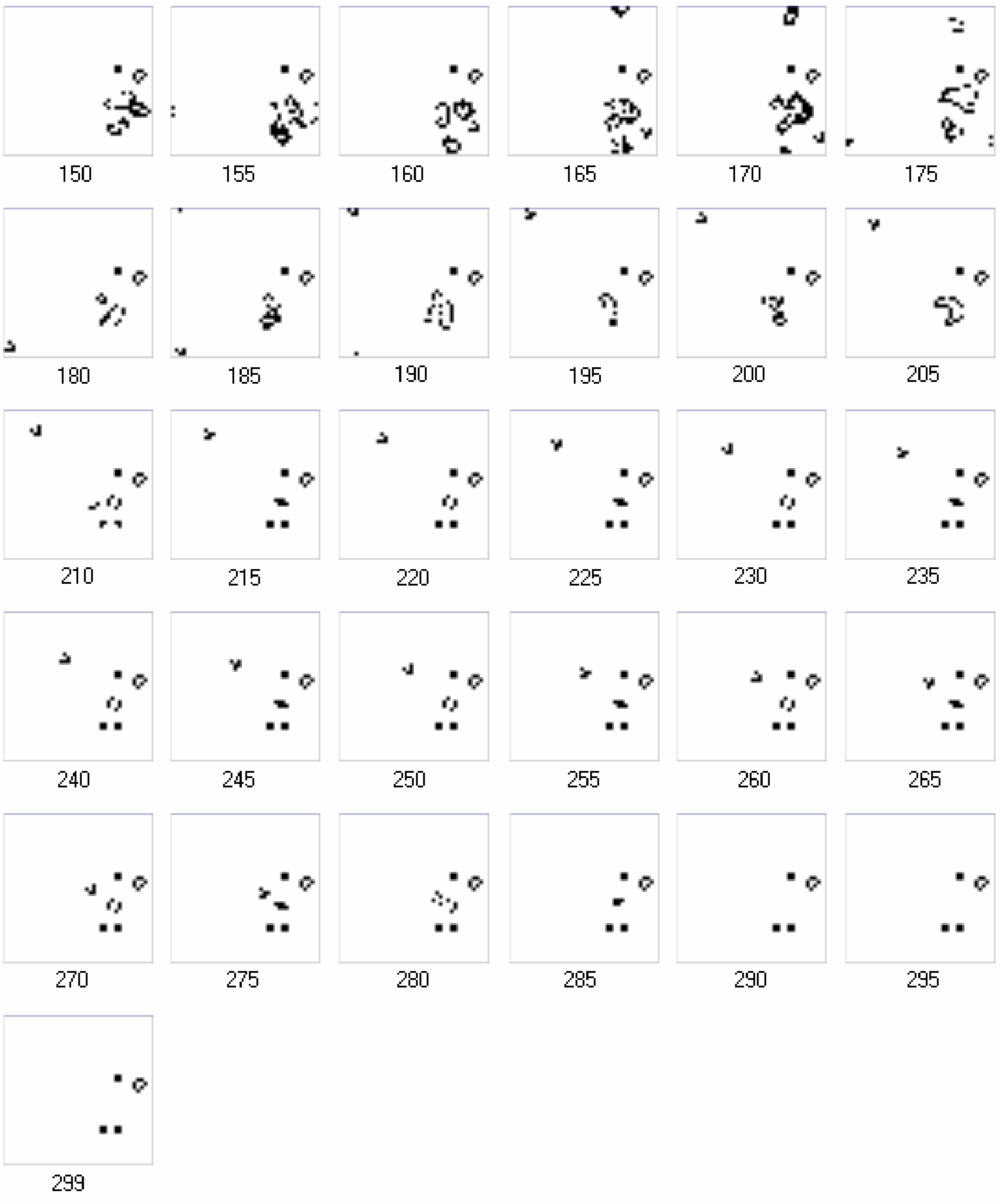


Figure 6 (Continued.) Generations 150-299

The two initial gliders, moving toward each other (generation 0,) eventually collide and get destroyed (generation 75,) disrupting the pentadecathlon in the process, which gets destroyed as well (generation 80.) From this point the automaton continues to evolve somehow chaotically. At about generation 125 a *block* and a *loaf* are created in the middle right of the matrix (see Fig. 6.) These two stable cell configurations stay undisrupted until the end (generation 299.) From generation 125, cell density increases, reaching a maximum at generation 170. At about generation 160, a new glider is created in the lower left corner of the matrix, and starts to move diagonally. From generation 212 the automaton stabilizes with three blocks, a toad, a loaf, and the moving glider. The glider moves towards the toad, colliding with it at generation 279. This interaction destroys both the glider and the toad, leaving the system with a stable configuration (three blocks and a loaf, generation 289.)

The evolution of the automaton was mapped to a polyphonic ensemble of ten instruments: wind quintet (flute, oboe, clarinet in B-flat, bassoon, and French horn,) string quartet (two violins, viola, and violoncello,) and harp.

Each generation of the automaton was mapped——using the normalized mapping method——onto event sets of durations, dynamics, and scales.

The mapping process is based upon cell density in the current generation (the number of cells in state 1.) The mapped scale, duration, and dynamic in the current generation are assigned to all instruments globally.

The durations event set is as follows:

Durations:	index	value (normalized mapping)
	0	16th
	1	16th
	2	8th
	3	quarter
	4	dotted quarter

The dynamics event set is as follows:

Dynamics:	index	value (normalized mapping)
	0	*p*
	1	*mf*
	2	*p*
	3	*f*
	4	*p*
	5	*mf*
	6	*p*
	7	*ff*

The arrangement of dynamic values employs the numerical fractal sequence 12131214..., where 1 is *p*, 2 is *mf*, 3 is *f*, and 4 is *ff.*

Polyphony is achieved by assigning 4 rows of the matrix to every instrument, sequentially from row 0 to row 39.

The flute is assigned rows 0-3, the oboe rows 4-7, and so on, as follows:

Instrument	Assigned Rows
Flute	0-3
Oboe	4-7
Clarinet	8-11
Violin I	12-15
Violin II	16-19
Viola	20-23
Violoncello	24-27
French Horn	28-31
Bassoon	32-35
Harp	36-39

graphically:

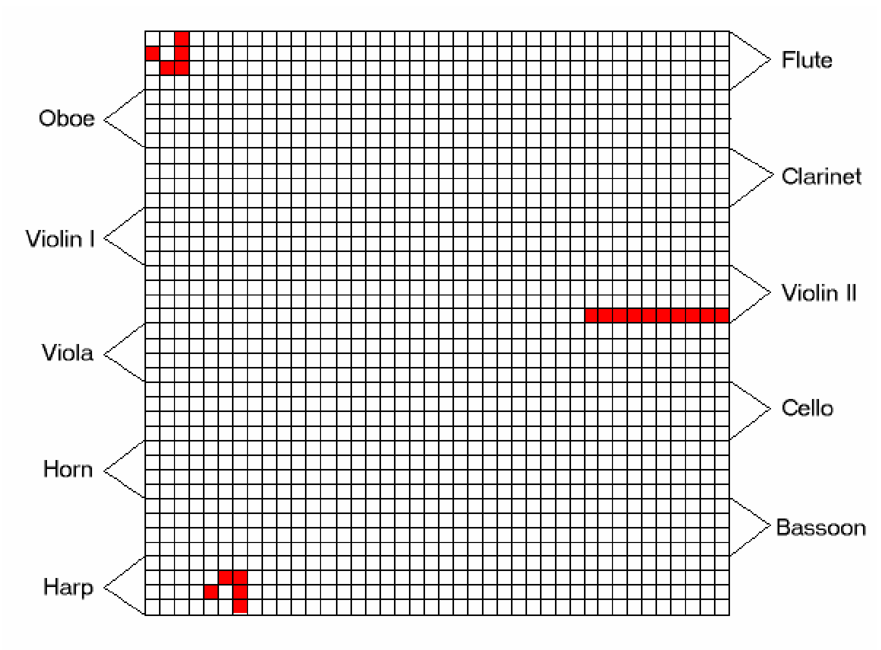


Figure 7. Polyphonic arrangement of rows

In every generation of the automaton, the cell configuration of each row is mapped to a pitch and assigned to its corresponding instrument. Consequently, every generation produces four pitches per instrument. The pitch event set is the actual scale previously mapped from the cell density of the current generation. The set of scales is as follows:

Index	Scale	Index	Scale
0	Dorian	8	Dorian
1	Acoustic	9	Acoustic
2	Dorian	10	Dorian
3	Scale3*	11	Scale3
4	Dorian	12	Dorian
5	Acoustic	13	Acoustic
6	Dorian	14	Dorian
7	Acoustic	15	Lydian

*The scale3 mode is the Locrian mode but with a lowered third degree:

$$\{0, 1, 2, 5, 6, 8, 10\}$$

The order of scales is based on the fractal sequence:

1 2 1 3 1 2 1 4 1 2 1 3 1 2 1 5...

where 1 represents the Dorian Greek mode, 2 and 4 the Acoustic scale, 3 the Scale3, and 5 the Lydian mode.

The pitch is computed as follows. A number *n* is generated from each row, through the following operation:

$$n = \sum_{i=0}^{39} s_i \cdot \phi^i$$

Figure 8. Formula for computing the pitch

where s_i is the state of the cell at position *i* (ranging from 0, the leftmost cell in the row, to 39, the rightmost cell in the row,) and ϕ is the golden mean constant 1.618033...

Next, a modulo operation is performed between the integral part of *n* and the number of notes of the scale previously decided:

pitch index in scale = [*n*] *mod* number of notes in the scale

This yields a number between 0 and the number of notes in the scale minus one. This number will be the index of the pitch in the mapped scale.

The last step is to decide the starting note of the scale. This pitch is mapped independently for every instrument from the cell density of the current generation, using the following table of values:

	Index of normalized cell density (0-7)							
	0	1	2	3	4	5	6	7
Instrument	Scale starting note event sets (pitches shown as MIDI note numbers)							
Flute	60	62	63	64	71	76	81	86
Oboe	60	62	63	64	59	64	69	74
Clarinet	60	62	51	76	71	52	57	62
Violin I	72	62	75	76	71	76	81	86
Violin II	60	62	63	64	59	64	69	74
Viola	48	50	51	52	59	52	57	62
Violoncello	36	50	51	40	59	52	45	38
Horn	48	50	51	52	59	52	57	62
Bassoon	48	38	51	52	47	52	57	62
Harp	84	74	63	76	83	64	57	38

Pitches are represented as MIDI note numbers. MIDI note 60 is middle C (C_4,) 61 is the C-sharp above, 59 is the B below, and so forth. From the scale, pitch index, and scale starting note, the specific pitch is drawn and assigned to its corresponding instrument. For example, if the scale is the Major scale, the pitch index is 4 (event number 5 in the set,) and the start note is, say, 71 (B_4,) the corresponding pitch would be F#$_5$, that is, the 5th pitch of a major scale starting on B_4.

If a row contains no cells in state 1, it is mapped to a rest. In addition, consecutive equal pitches are tied.

The choice of instrumentation, scales, range of instruments, dynamics and durations were all decided by this writer.

In sum, each generation of the automaton generates four pitches per instrument. Duration, dynamic and scales are global in every generation (the same for all instruments) and are mapped based on the cell density of the current generation. Pitches are mapped independently for every instrument, based on the cell configuration of the four rows assigned to them. To further clarify this process, a detailed explanation of the mapping of generation 0 (see Fig. 6) follows.

The maximum cell density of the system is 83, reached at generation 170. This was computed beforehand in order to apply the normalized mapping method.

First, we compute the global parameters: scale, duration and dynamic.

Cell density in generation 0 is 20. Our scale event set has 16 members (ordered from 0 to 15):

Event No.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Scale	1	2	1	3	1	2	1	4	1	2	1	3	1	2	1	5

where 1 represents the Dorian Greek mode, 2 and 4 the Acoustic scale, 3 the Scale3 mode, and 5 the Lydian mode.

Next, we apply the formula for the normalized mapping,

$$\text{Event No.} = \left[\frac{(\text{value} - \text{minval}) * \text{numevents}}{\text{maxval} - \text{minval}} \right]$$

where "value" is the current generation's cell density, 20, "minval" is the minimum possible density, 0, and "maxval" is the maximum density reached, 83. Since minval is 0 in this case, the formula reduces to

Event No. = [(current density/maximum density)*16]

that is,

Event No. = [(20/83)*16] = 3

Event No. 3 is scale No. 3: the Scale3 mode (see table above.)

The dynamic is mapped from the following event set,

Event No.	0	1	2	3	4	5	6	7
Dynamic	1	2	1	3	1	2	1	4

where 1 is *p*, 2 is *mf*, 3 is *f*, and 4 is *ff.* This event set has 8 members, thus

Event No. = [(20/83)*8] = 1

Event No. 1 is dynamic No. 2, that is, *mf*.

The duration is mapped from the following event set:

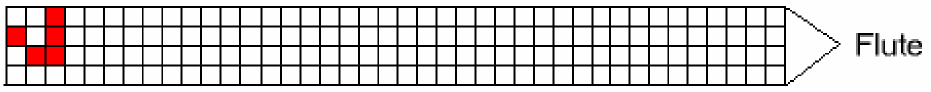
Event No.	0	1	2	3	4
Dynamic	16th	16th	8th	quarter	dotted quarter

Applying the formula yields (this event set has 5 members):

Event No. = [(20/83)*5] = 1 = 16th note values

The global parameters, therefore, Scale3, sixteenth note values, and a *mf* dynamic intensity.

Next, we compute the pitch from each of the four rows assigned to every instrument. The flute rows have the following cell configuration:



Applying the pitch formula (see Fig. 8) to each of the four rows produces the following results (note that the fourth row has no cells in it, and therefore it is mapped to a rest):

2.61803, 3.61803, 4.236063, rest

Next, we perform a modulo 7 operation on the integral part of the above values (since Scale3 has 7 notes,) yielding the following results:

2, 3, 4, rest

Lastly, we get the starting note of the scale, mapped from the following table of values:

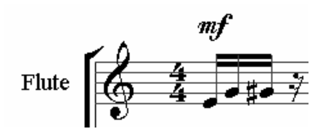
Event No.	Index of normalized cell density (0-7)							
	0	1	2	3	4	5	6	7
Instrument	Scale starting note event sets (pitches shown as MIDI note numbers)							
Flute	60	62	63	64	71	76	81	86

Event No. = [(20/83)*8] = 1 = MIDI Note 62 = D_4

Therefore, the starting pitch for the mapped scale (Scale3) is D_4. The scale from which the pitches are drawn is, thus

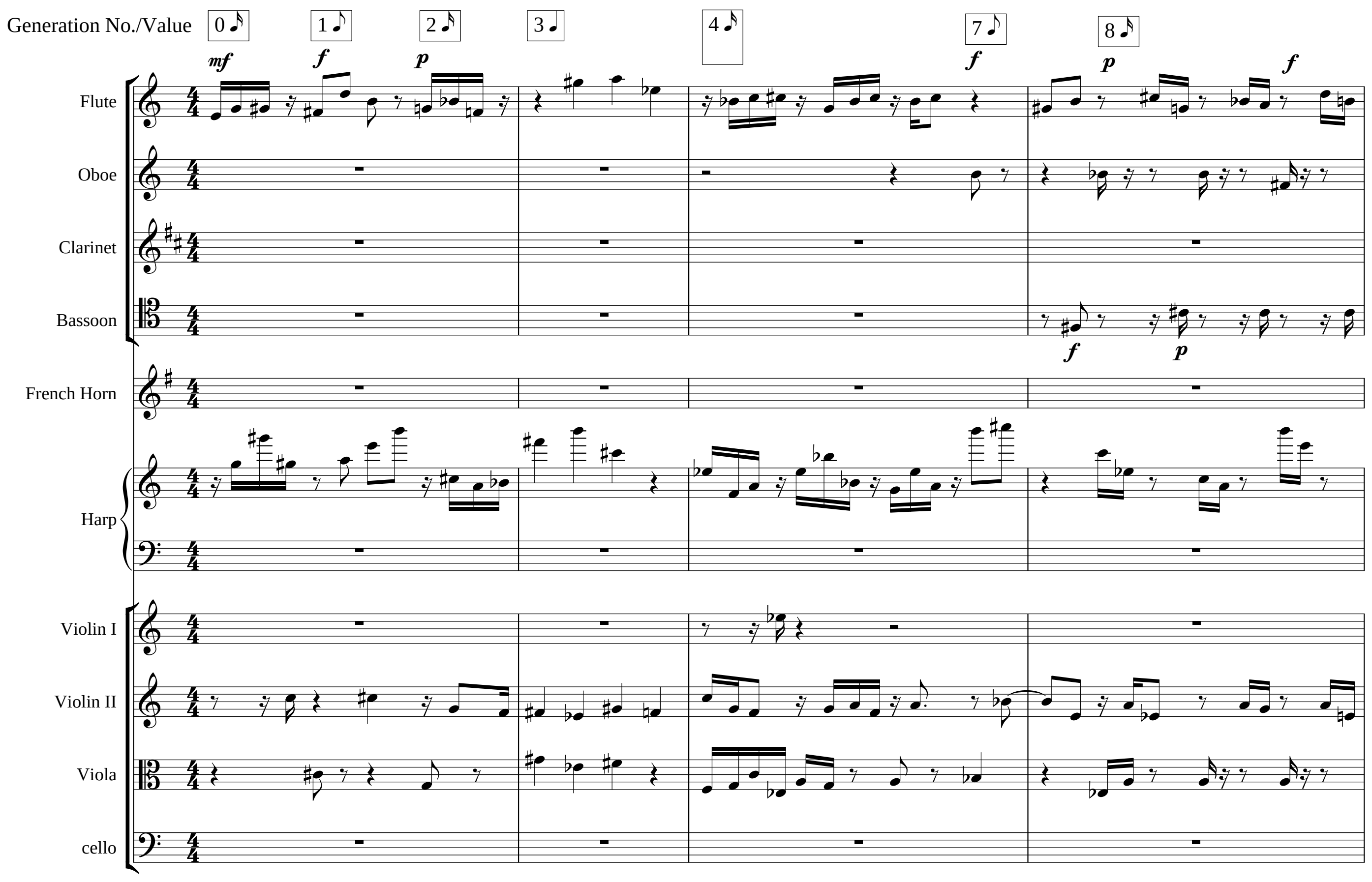
D_4, $D\#_4$, E_4, G_4, $G\#_4$, $A\#_4$, C_5

The indexes of the mapped pitches are 2, 3, 4, and a rest, which correspond to E_4, G_4, $G\#_4$, and rest. Putting everything together (dynamic, note value, and pitches) yields the following passage for the flute in Generation 0:

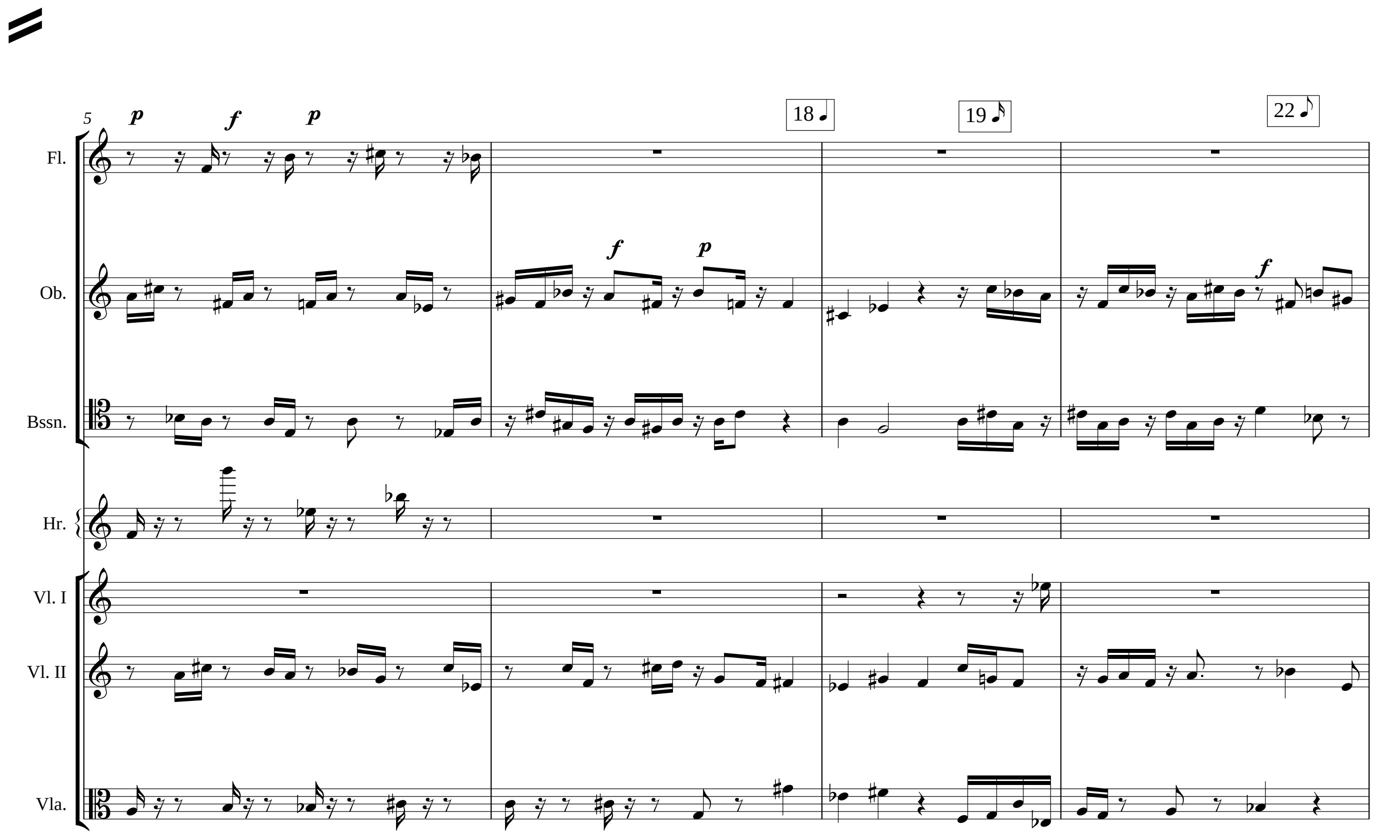


The same process is then applied to the rest of the instruments.

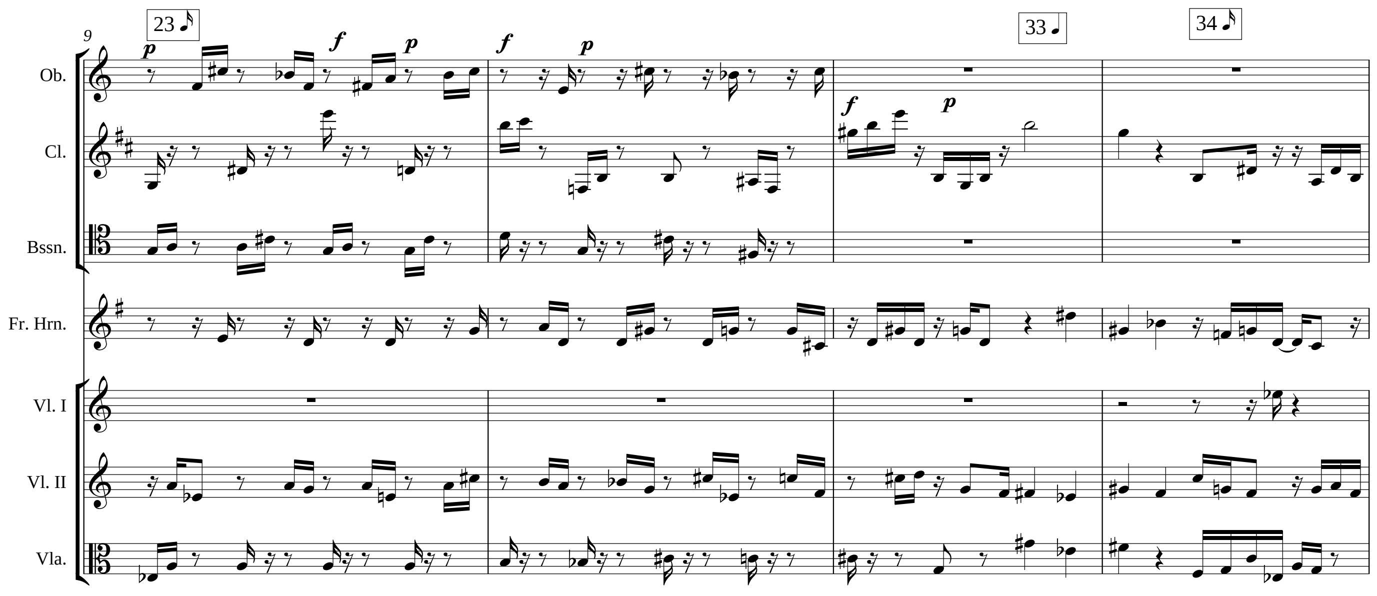
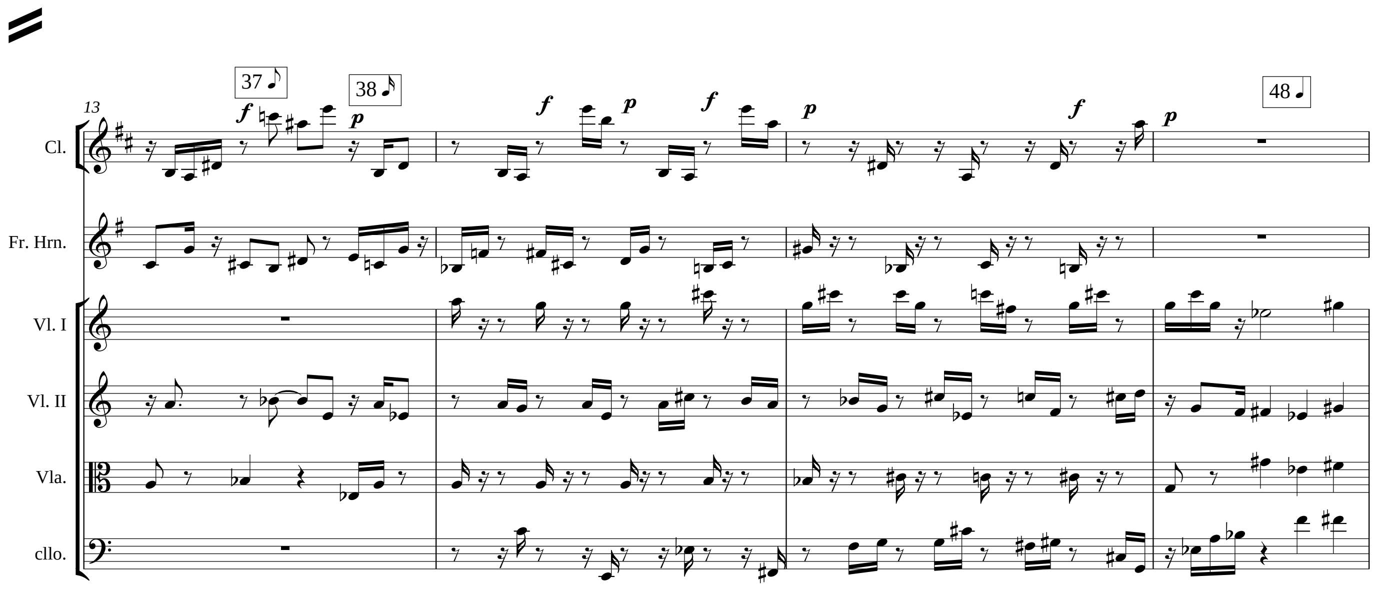
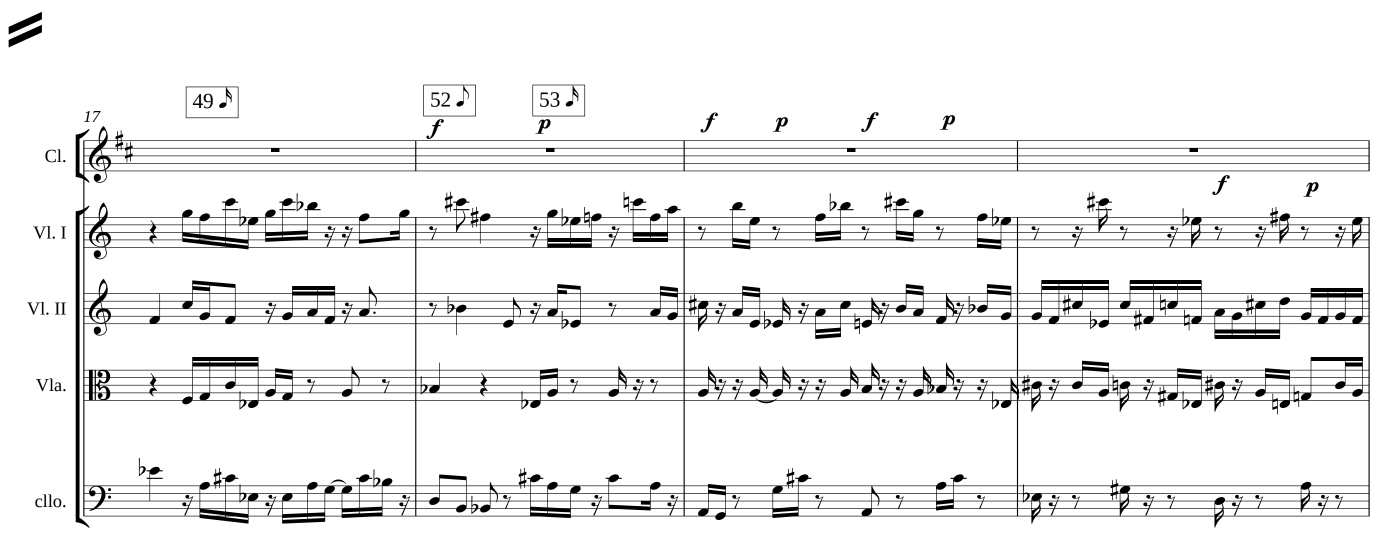
Applying this mapping scheme to all instruments in all 300 generations, produces the following score:

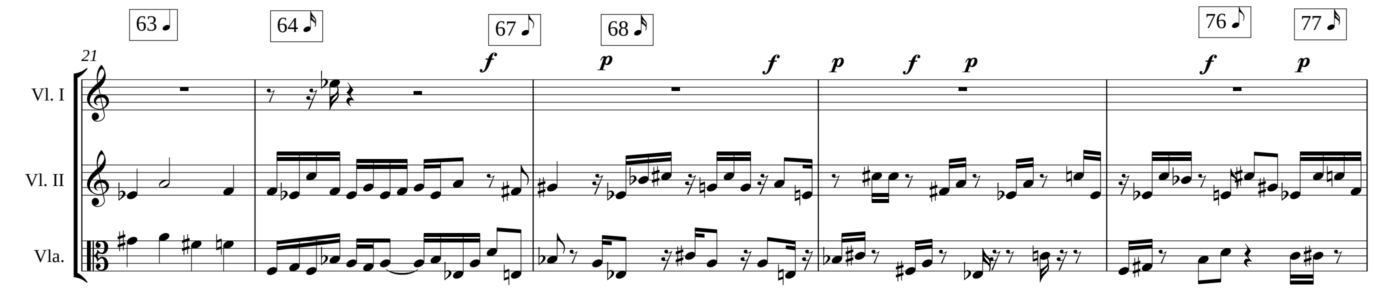


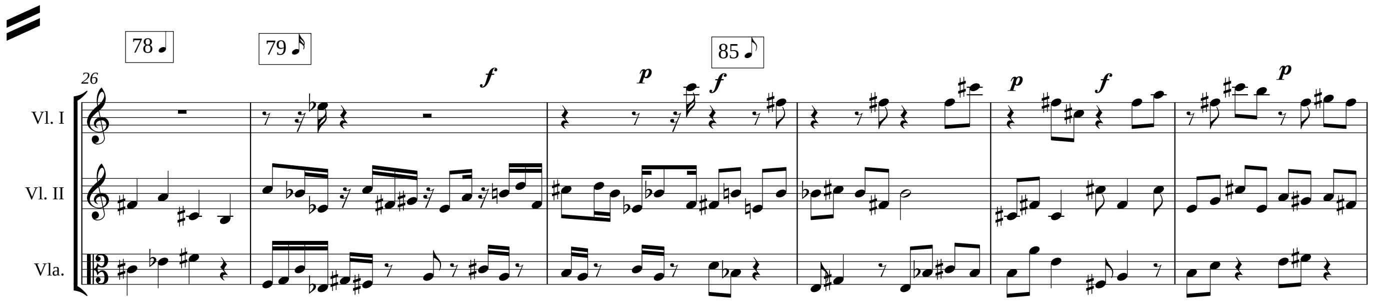
Generation No./Value
0
1
2
3
4
7
8
mf
f
p
Flute
Oboe
Clarinet
Bassoon
French Horn
Harp
Violin I
Violin II
Viola
cello

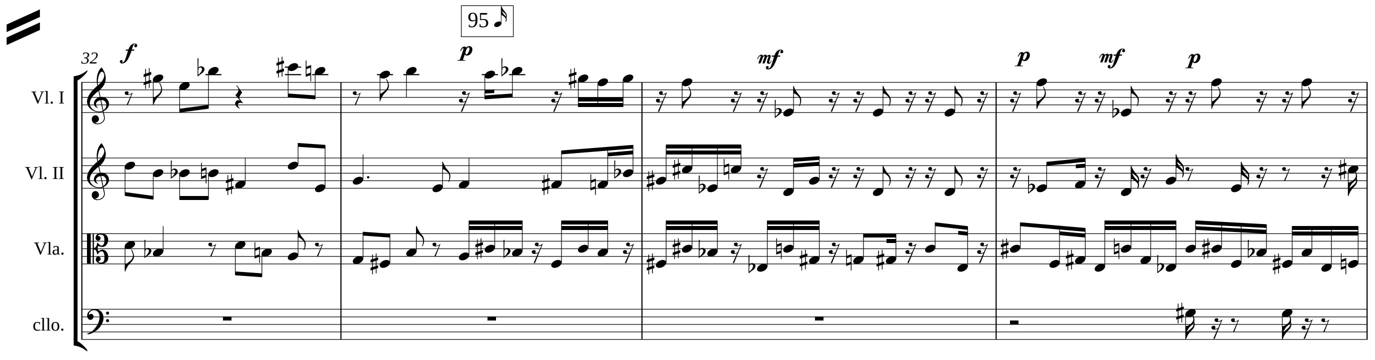


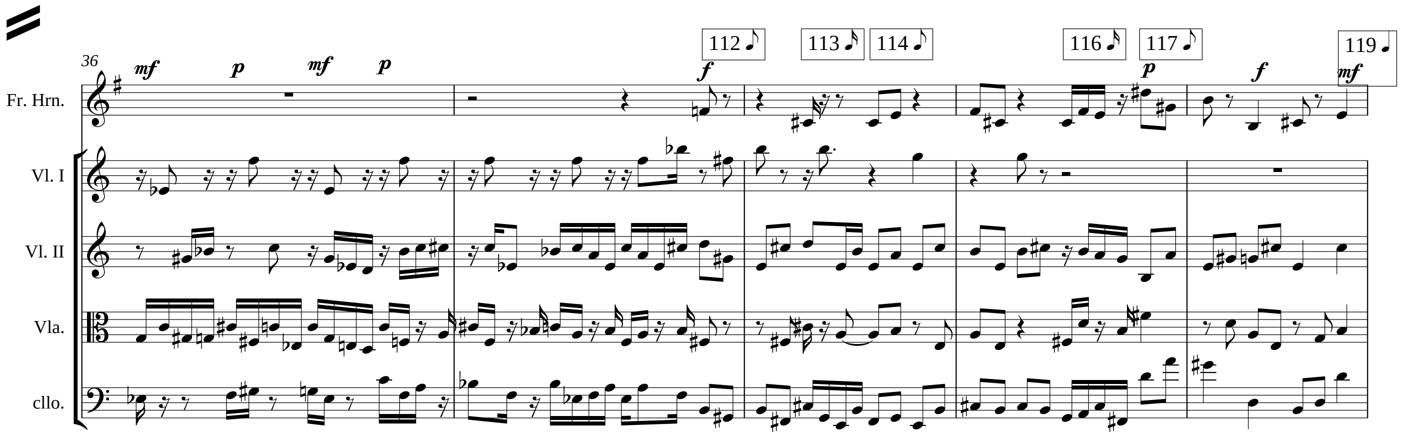
5
18
19
22
p
f
Fl.
Ob.
Bssn.
Hr.
Vl. I
Vl. II
Vla.

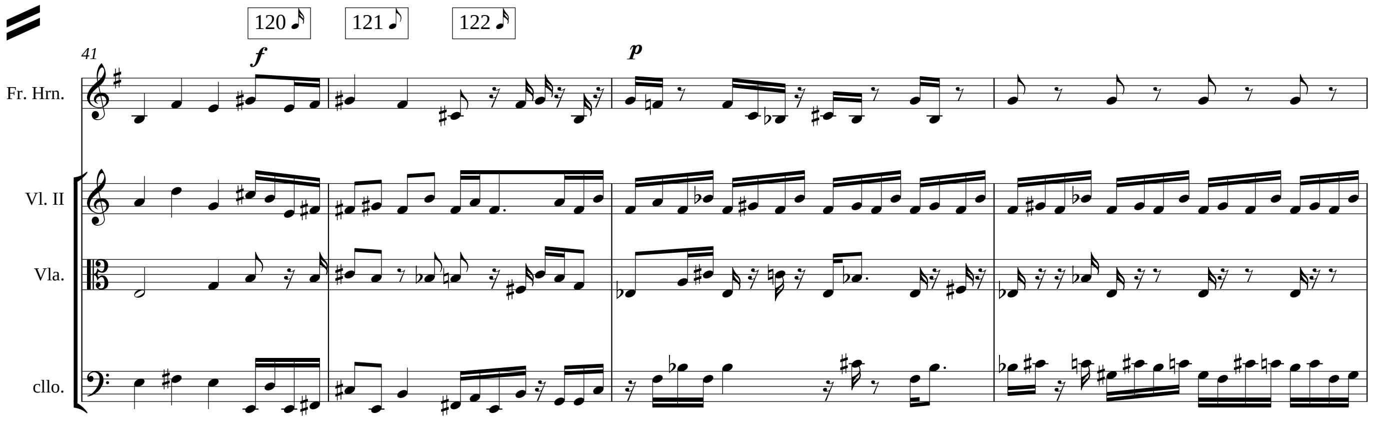

9
23
33
34
Ob.
Cl.
Bssn.
Fr. Hrn.
Vl. I
Vl. II
Vla.
f
p

13
37
38
48
Cl.
Fr. Hrn.
Vl. I
Vl. II
Vla.
cllo.
f
p

17
49
52
53
Cl.
Vl. I
Vl. II
Vla.
cllo.
f
p

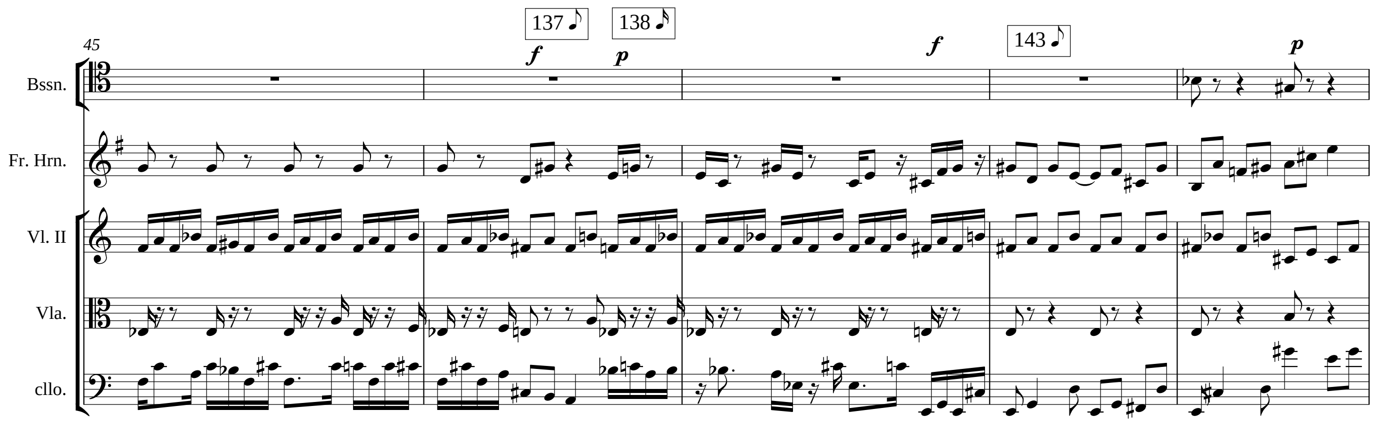

63
64
67
68
76
77
21
f
p
f
p
f
p
f
p
Vl. I
Vl. II
Vla.

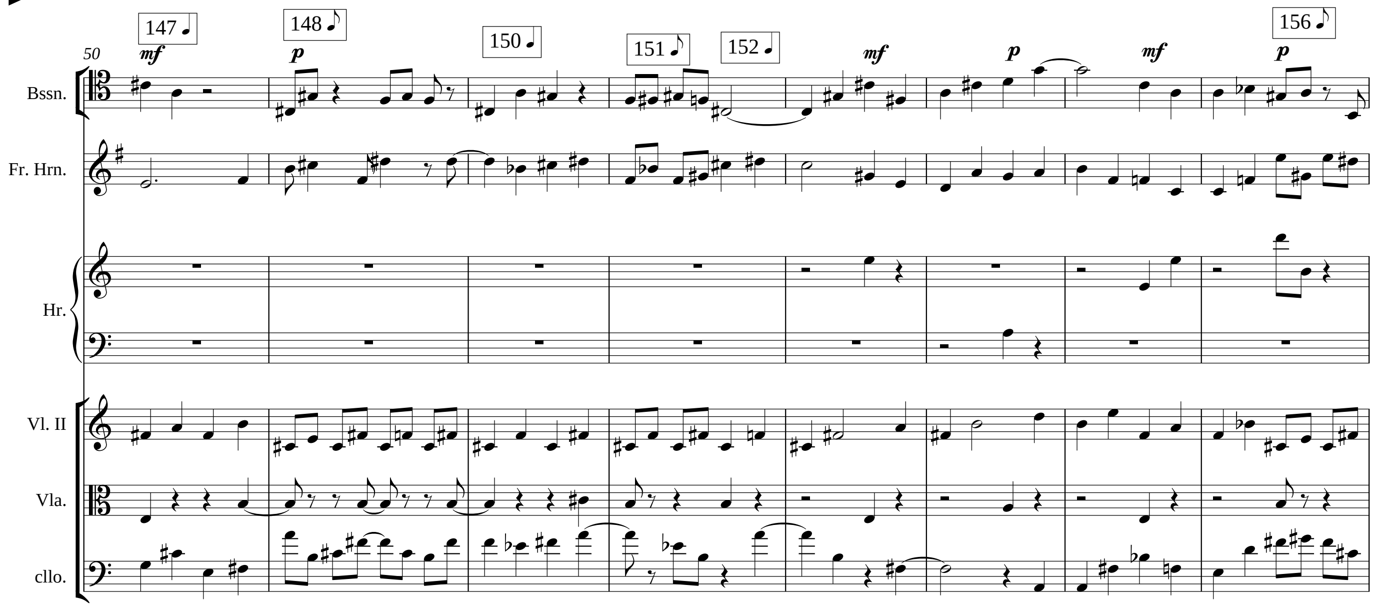

78
79
85
26
f
p
f
p
f
p
Vl. I
Vl. II
Vla.

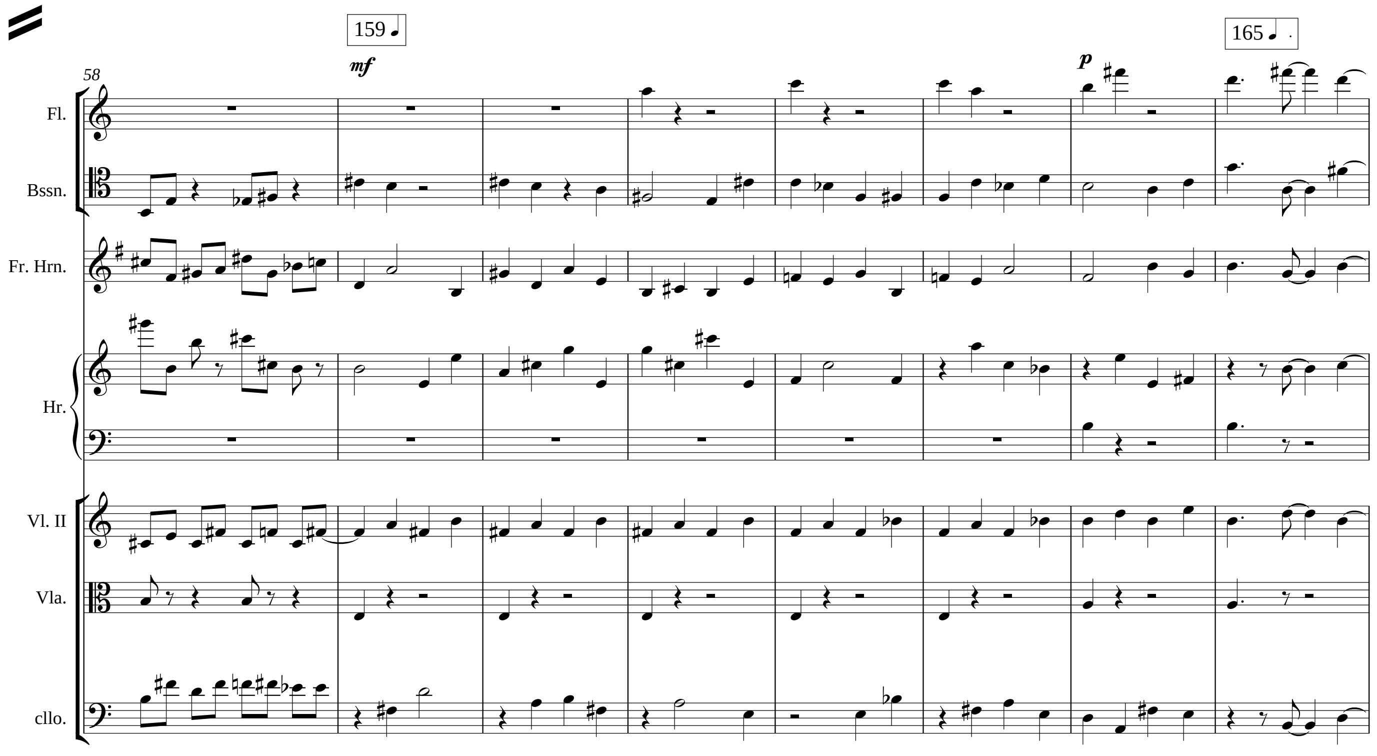

95
32
f
p
mf
p
mf
p
Vl. I
Vl. II
Vla.
cllo.

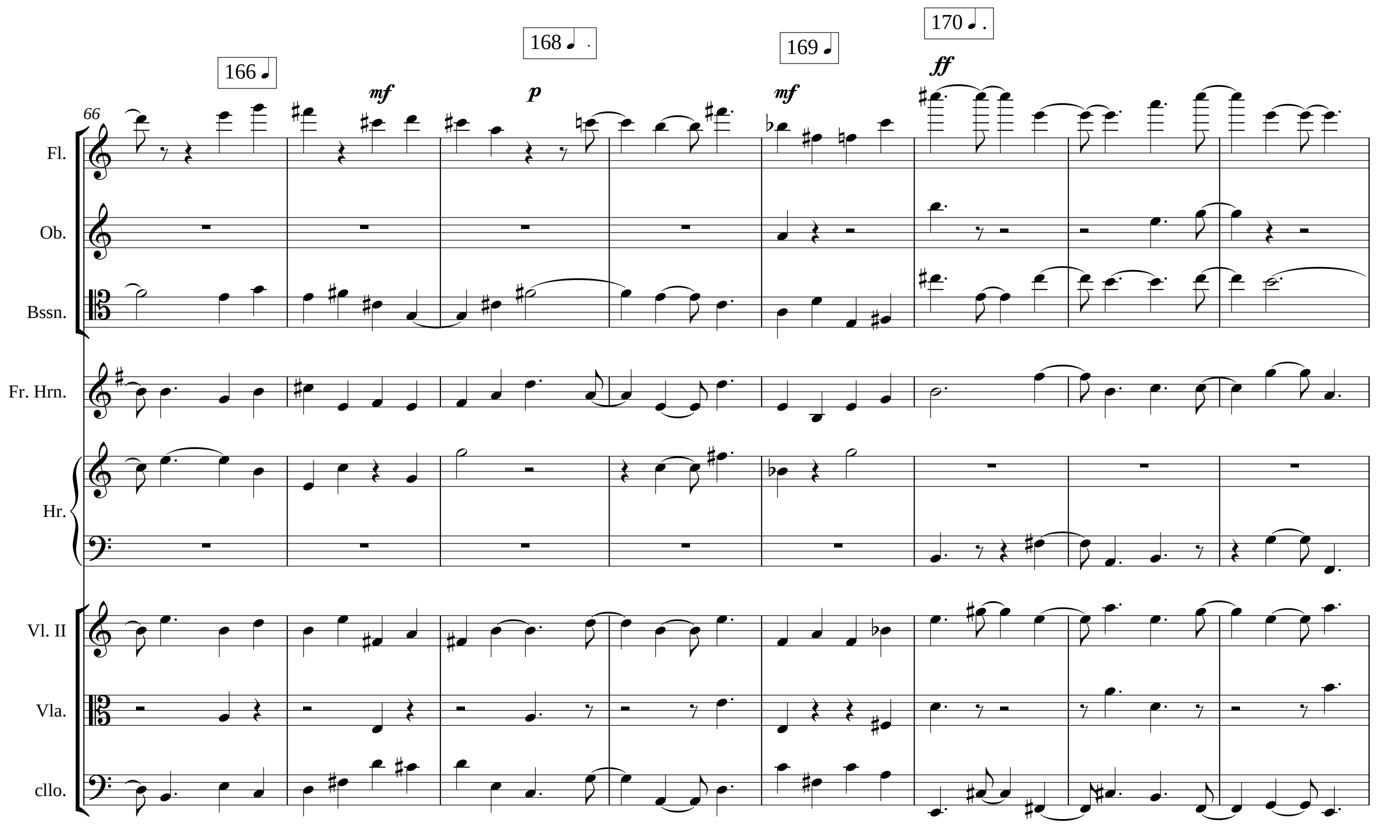

112
113
114
116
117
119
36
mf
p
mf
p
f
p
f
mf
Fr. Hrn.
Vl. I
Vl. II
Vla.
cllo.


120
121
122
41
f
p
Fr. Hrn.
Vl. II
Vla.
cllo.

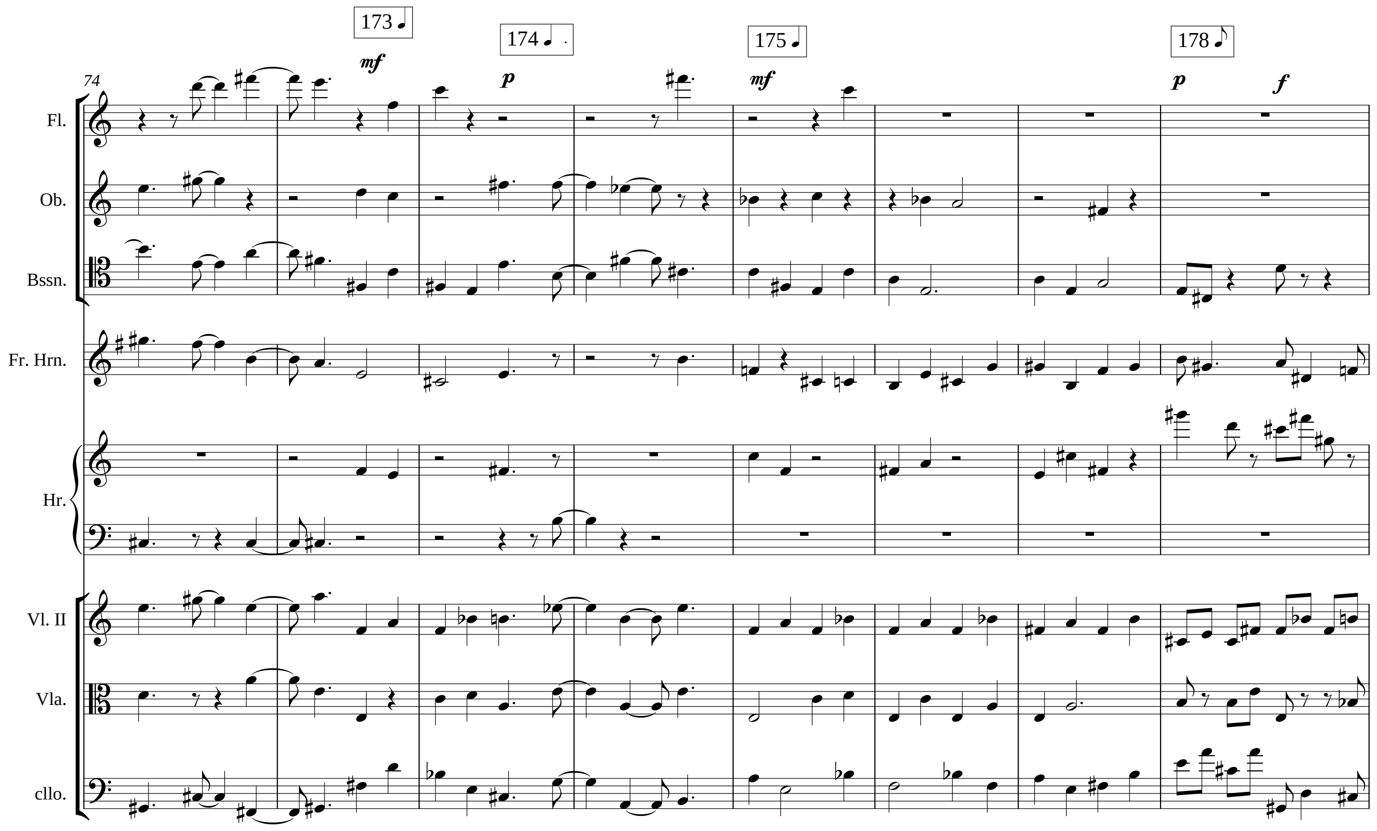

45
137 ♪
138 ♬
143 ♪
f
p
f
p
Bssn.
Fr. Hrn.
Vl. II
Vla.
cllo.


50
147 ♩
148 ♪
150 ♩
151 ♪
152 ♩
156 ♪
mf
p
mf
p
mf
p
Bssn.
Fr. Hrn.
Hr.
Vl. II
Vla.
cllo.

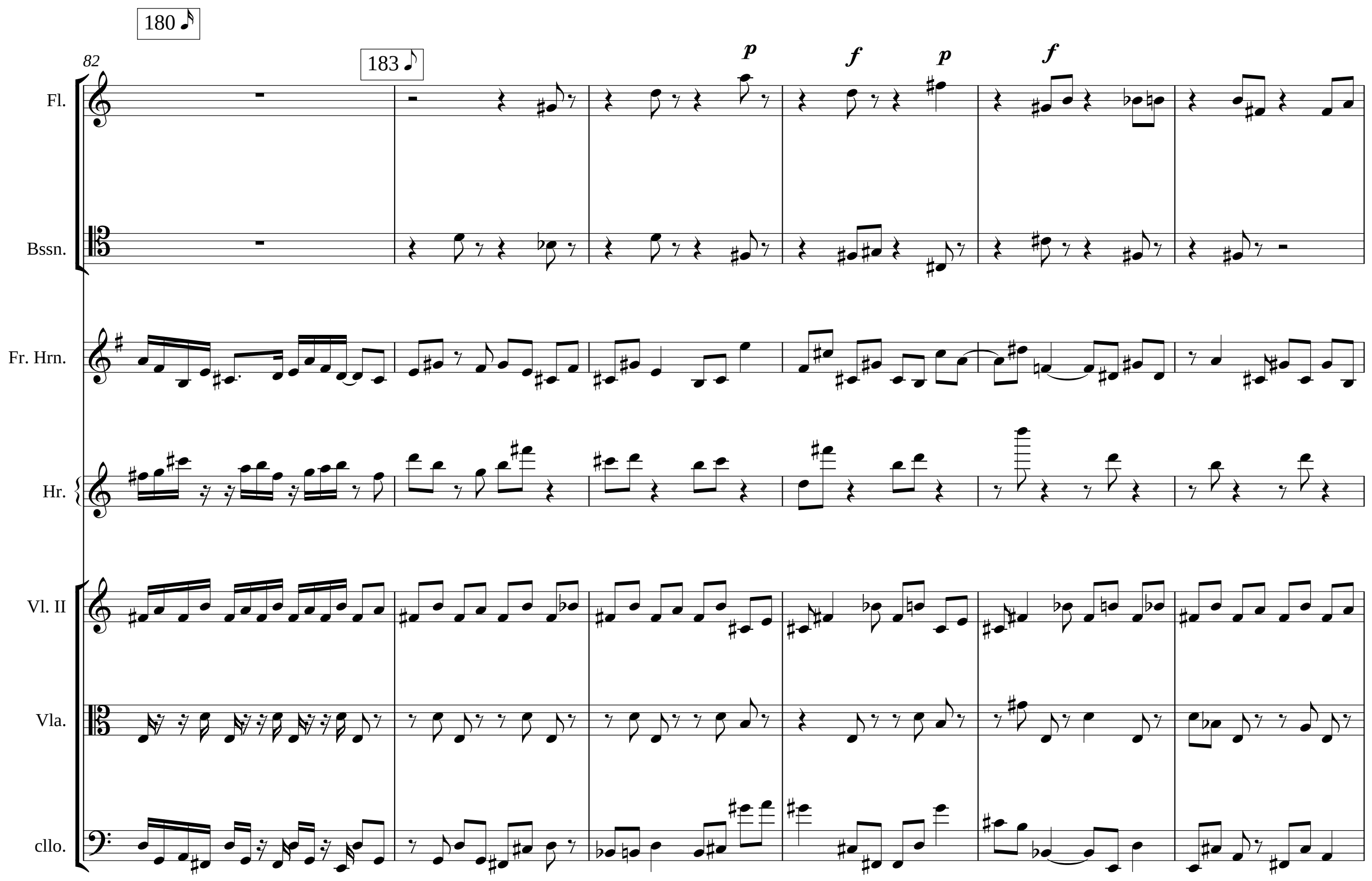

58
159 ♩
165 ♩.
mf
p
Fl.
Bssn.
Fr. Hrn.
Hr.
Vl. II
Vla.
cllo.



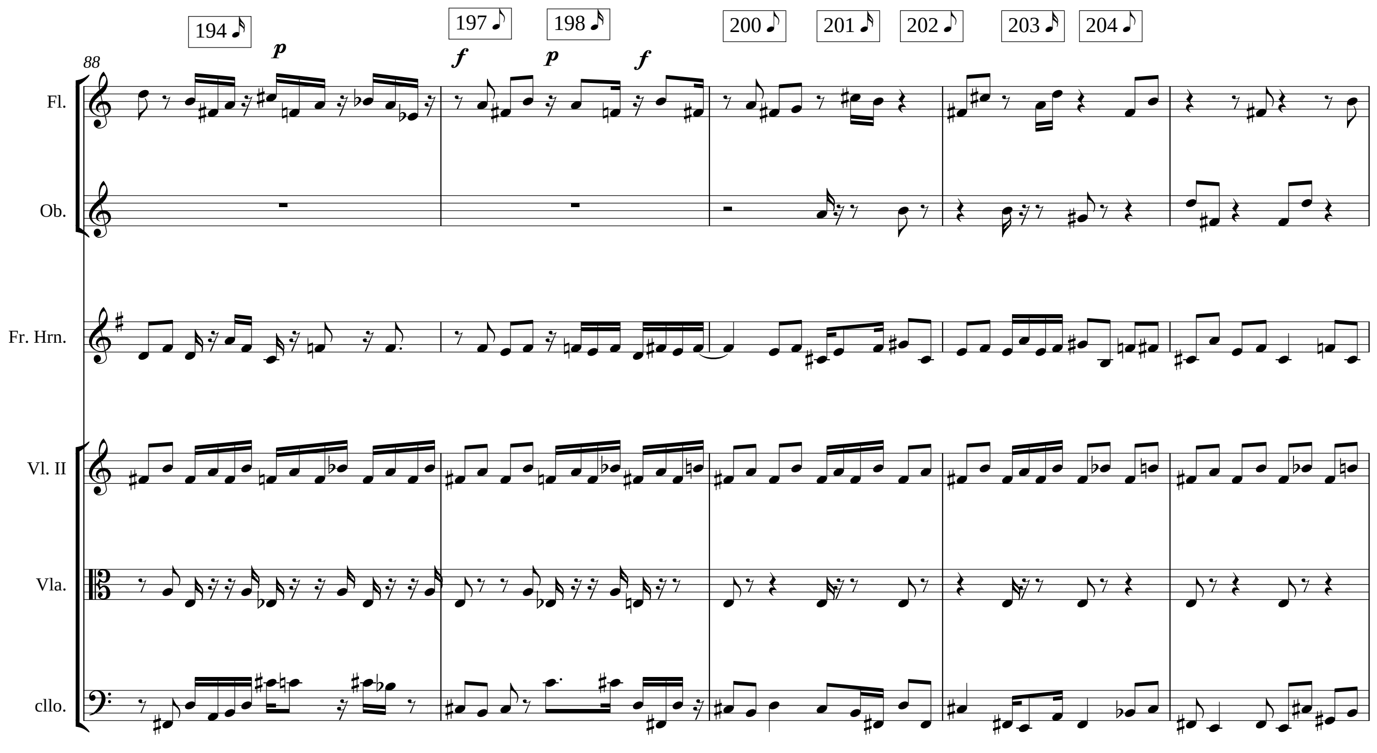
166
168
169
170
mf
p
mf
ff
66
Fl.
Ob.
Bssn.
Fr. Hrn.
Hr.
Vl. II
Vla.
cllo.



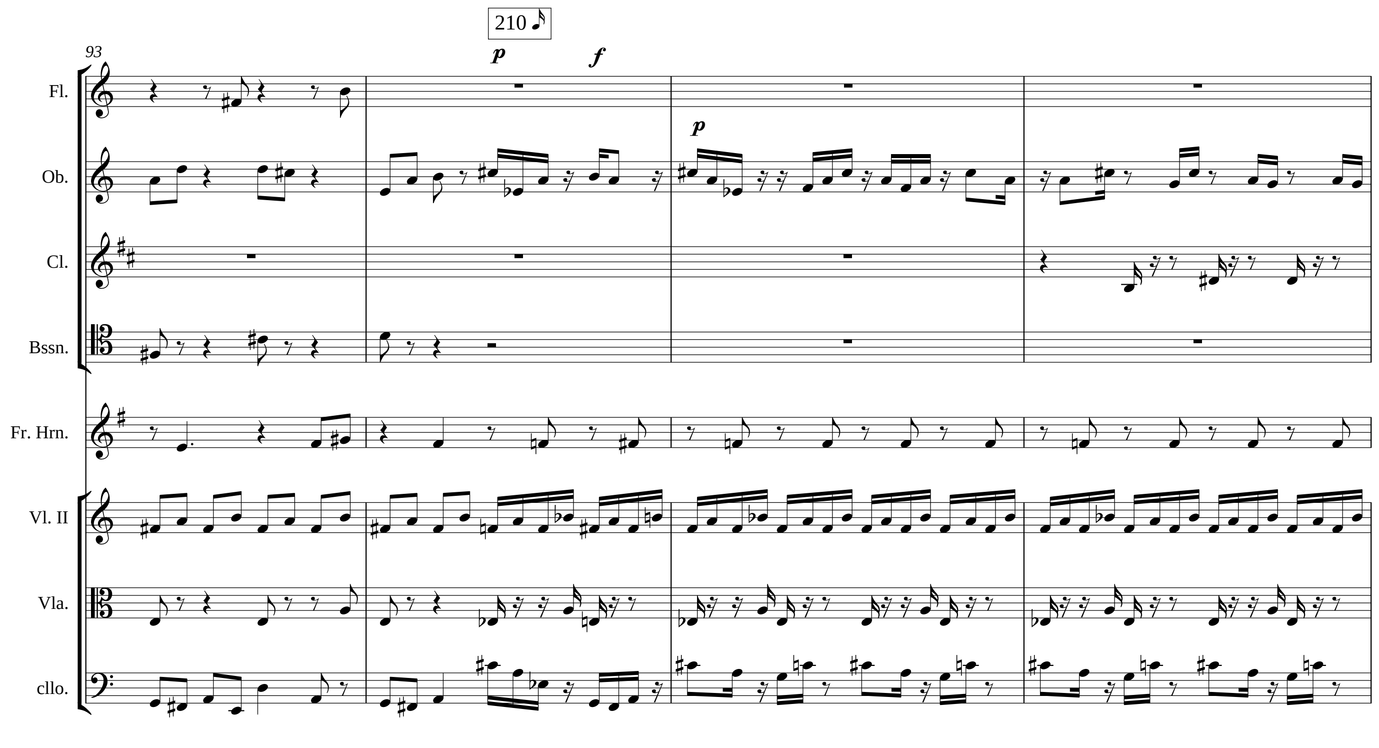
173
174
175
178
mf
p
mf
p
f
74
Fl.
Ob.
Bssn.
Fr. Hrn.
Hr.
Vl. II
Vla.
cllo.



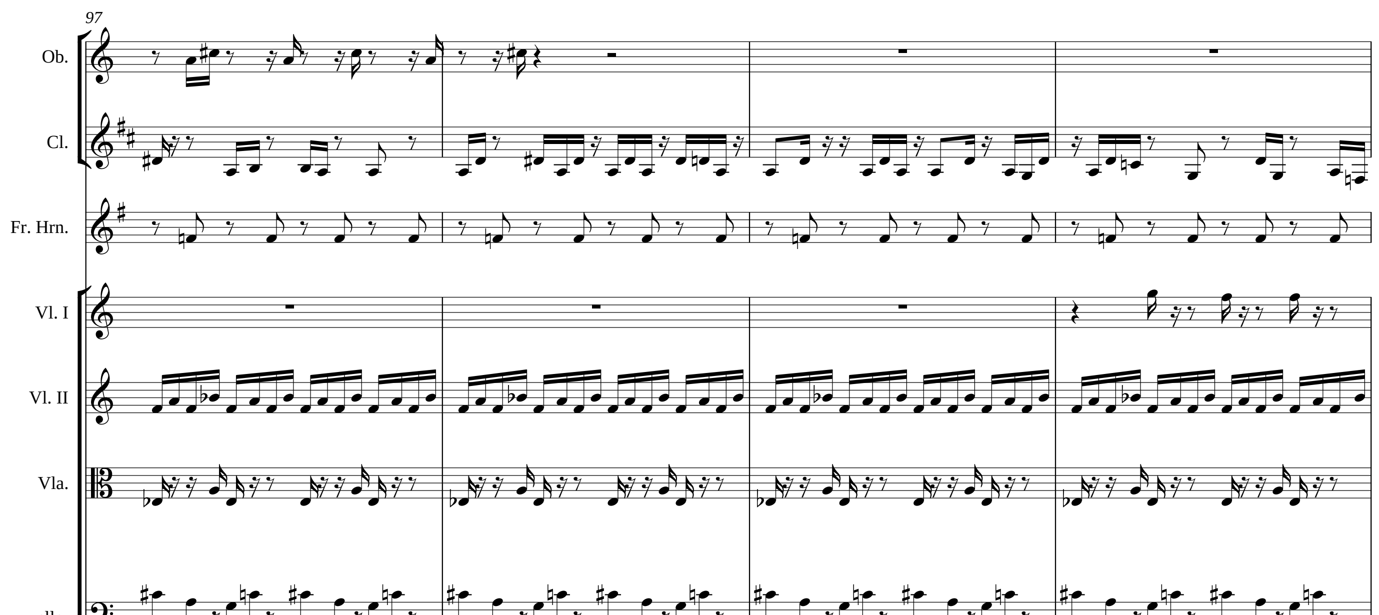
180
82
183
p
f
p
f
Fl.
Bssn.
Fr. Hrn.
Hr.
Vl. II
Vla.
cllo.



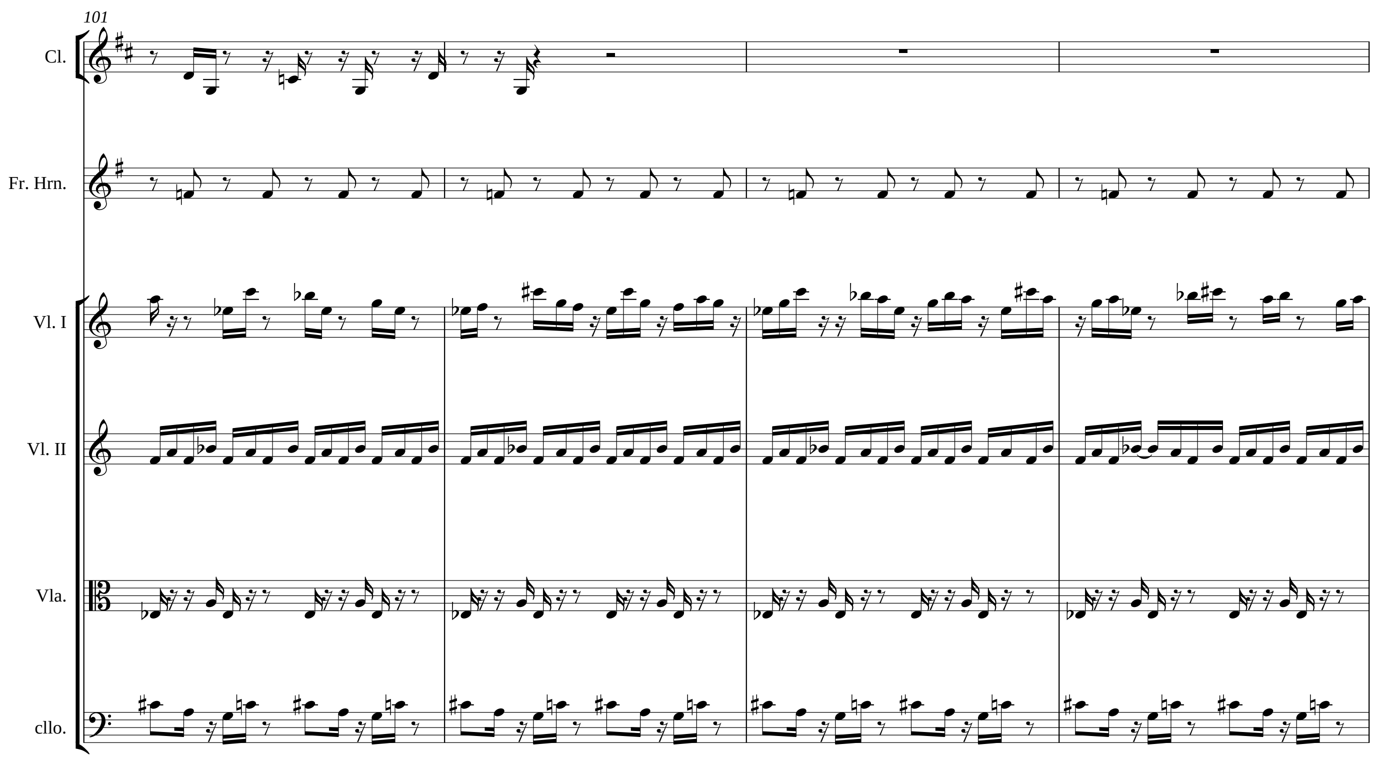
194
197
198
200
201
202
203
204
88
p
f
p
f
Fl.
Ob.
Fr. Hrn.
Vl. II
Vla.
cllo.

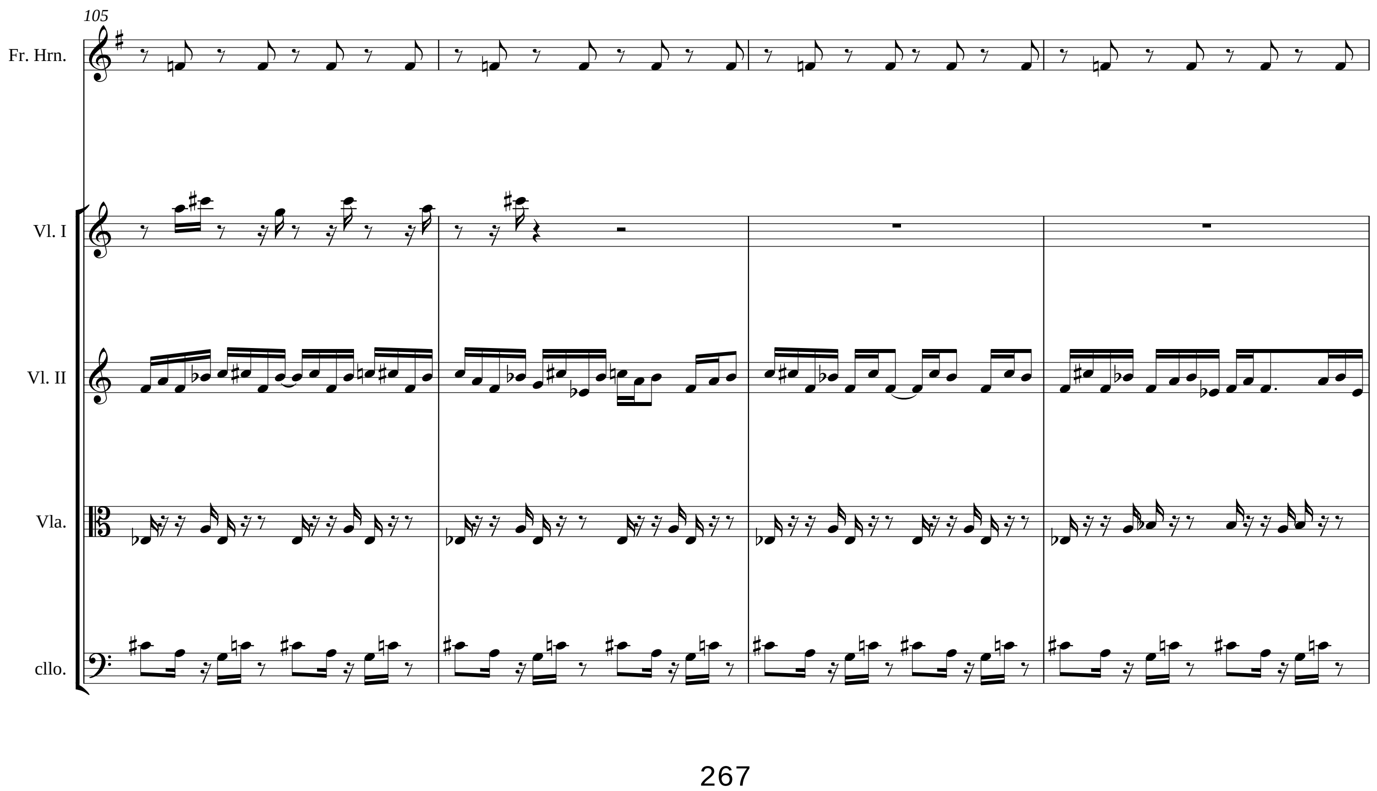


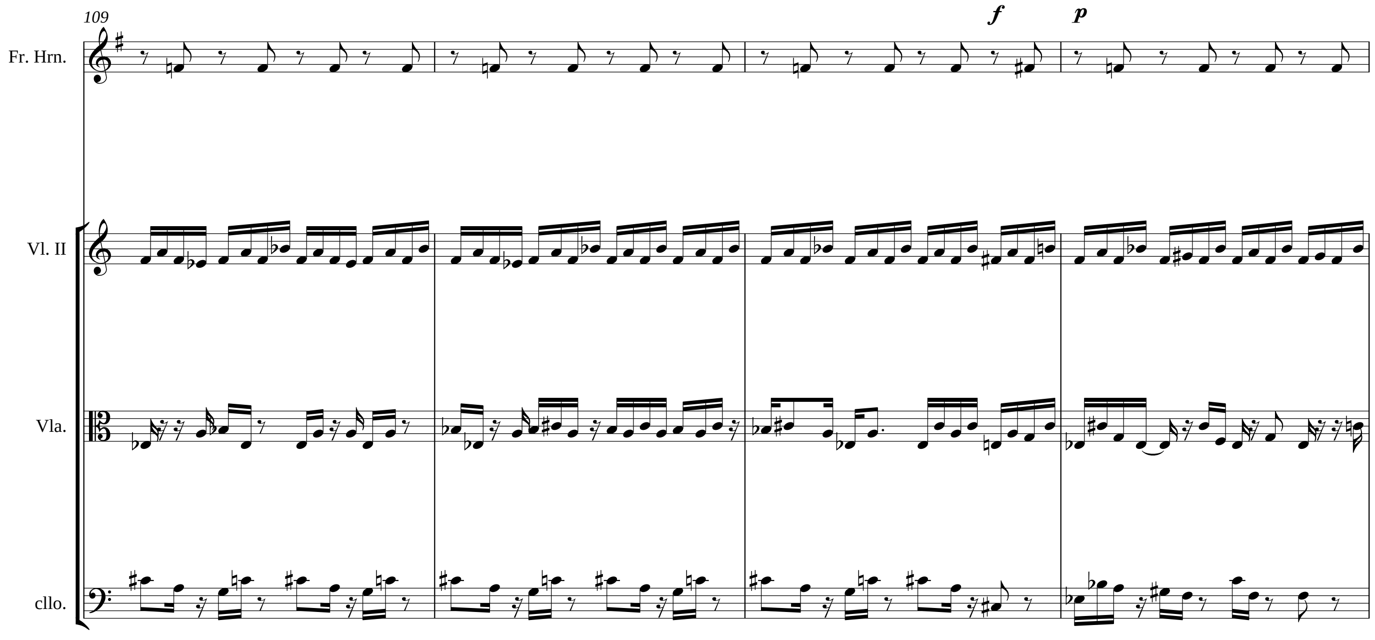
210
93
p
f
p
Fl.
Ob.
Cl.
Bssn.
Fr. Hrn.
Vl. II
Vla.
cllo.

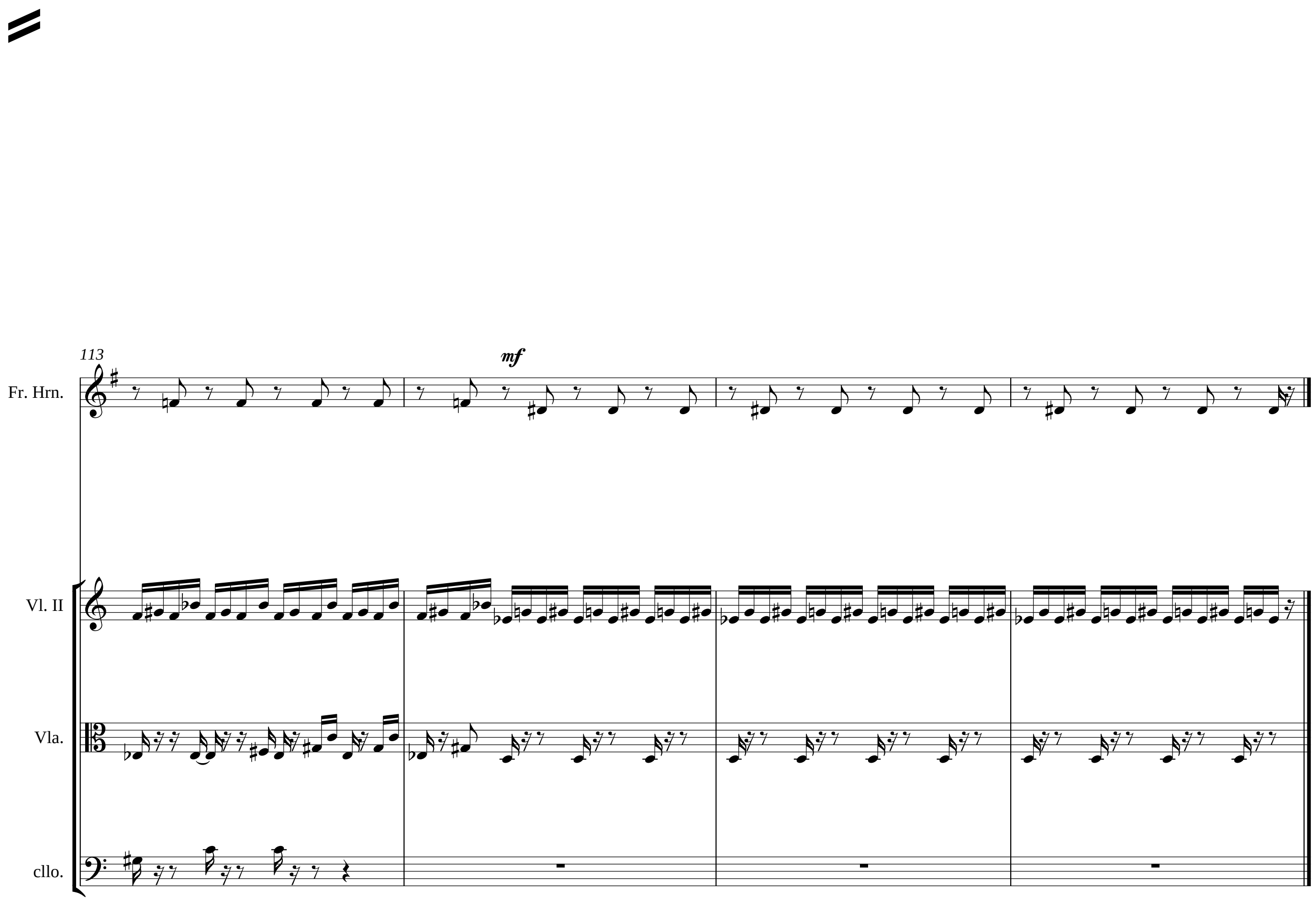


97
Ob.
Cl.
Fr. Hrn.
Vl. I
Vl. II
Vla.
cllo.


101
Cl.
Fr. Hrn.
Vl. I
Vl. II
Vla.
cllo.


105
Fr. Hrn.
Vl. I
Vl. II
Vla.
cllo.


109
f
p
Fr. Hrn.
Vl. II
Vla.
cllo.


113
mf
Fr. Hrn.
Vl. II
Vla.
cllo.

Duration and dynamic changes are marked at the top of systems in the score. Duration changes are boxed along with the generation number were they occur. Since each generation produces four identical rhythmic values, these markers can be used to locate any particular generation by simply counting groups of four values from the marked generation.

The generated scales in each generation are as follows:

Generation	Scale	Generation	Scale	Generation	Scale	Generation	Scale
0	scale3	70	dorian	122-123	dorian	179	acoustic
1	dorian	71	acoustic	124	acoustic	180-184	dorian
2-9	acoustic	72	dorian	125-131	dorian	185	acoustic
10	dorian	73-74	acoustic	132	acoustic	186-187	dorian
11	acoustic	75	dorian	133	dorian	188	acoustic
12	dorian	76-79	acoustic	134-136	acoustic	189	dorian
13-14	acoustic	80	dorian	137	dorian	190-191	acoustic
15-16	dorian	81	acoustic	138-141	acoustic	192-194	dorian
17-24	acoustic	82-83	dorian	142-144	dorian	195-196	acoustic
25	dorian	84-87	acoustic	145	acoustic	197	dorian
26	acoustic	88-91	dorian	146-148	dorian	198	acoustic
27	dorian	92	acoustic	149-152	acoustic	199-203	dorian
28-29	acoustic	93-94	dorian	153-154	dorian	204	acoustic
30-31	dorian	95	acoustic	155	scale3	205	dorian
32-39	acoustic	96-97	dorian	156-157	dorian	206	acoustic
40	dorian	98-100	scale3	158	acoustic	207-209	dorian
41	acoustic	101	dorian	159-161	dorian	210	acoustic
42	dorian	102	scale3	162-163	scale3	211	dorian
43-44	acoustic	103-104	dorian	164-168	dorian	212-278	acoustic
45-46	dorian	105	scale3	169	scale3	279	dorian
47-54	acoustic	106	dorian	**170**	**lydian**	280	acoustic
55	dorian	107	scale3	171	dorian	281	dorian
56	acoustic	108	acoustic	172	lydian	282	acoustic
57	dorian	109	dorian	173	scale3	283-288	dorian
58-59	acoustic	110-112	acoustic	174	acoustic	289-299	scale3
60-61	dorian	113-120	dorian	175-176	scale3		
62-69	acoustic	121	acoustic	177-178	dorian		

What follows is an analysis of the generated score and an explanation of how it relates to the structure of the cellular automaton.

The first 23 measures in the score correspond to the first 70 generations of the automaton. The two gliders at generation 0 start to move diagonally in opposite direction toward each other until they coalesce at about generation 70. Since they propagate through the matrix, their structure is mapped to several instruments. The glider at the top of the matrix in generation 0 moves through the rows assigned to the flute (mm. 1-5,) oboe (mm. 3-10,) clarinet (mm. 9-15,) and violin I (mm. 13 and following.) Simultaneously, the second glider at the bottom of the matrix moves through the rows assigned to the harp (mm. 1-5,) bassoon (mm. 4-10,) horn (mm. 9-15,) and cello (mm. 14 and following.) At the same time, the oscillating pentadecathlon affects the rows assigned to violin I, violin II, and viola. The pentadecathlon is a non-propagating, period-15 oscillator. Its cell structure is musically transcribed as a repeating passage between violin I, violin II, and viola (mm. 1-6, 6-11, 11-15, 16-18.) From m. 18 (generation 52,) although the pentadecathlon is still undisrupted, the two gliders have already entered the rows assigned to violin I, violin II and viola, thus generating different pitches. From mm. 16-37 (generations 47-111,) the automaton evolves in the rows assigned to the string quartet. There is an increase in cell density from generation 85 to 95, thus the longer values in mm. 28 to 33. From generation 97 to 111 a block stays undisrupted in the two middle rows assigned to the violin I (see figure 6.) This transcribes musically to the pitches of mm. 34-37. Although the block is a stable structure, the pitches generated from it change because the scales and scale start-note, which are mapped to the overall cell density, change within this passage as well.

From m. 37 (generation 112,) cells start to occupy the rows assigned to other instruments. The horn enters at m. 37, then the bassoon (m. 49, generation 145,) harp (m. 54, generation 153,) flute (m. 61, generation 161,) oboe (m. 70, generation 169,) etc.

From generation 123, a block and loaf are created and remain undisrupted until the end (see Fig. 6, middle right of the matrix.) These configurations occupy all four rows assigned to the violin II and the first of the viola (see Fig. 6, generations 125 and following.) This creates a four-note pattern that repeats in the violin II from m. 43 to the end of the piece (see score.) The pitches change because the scales (as well as the scale start-note) from which they are drawn, are mapped to the overall cell density and change as well; however, the intervalic contour of the pattern is always the same. From generation 143 the cell density of the automaton starts to increase, reaching a maximum at generation 170. This transcribes musically as longer note values, just as decided in the mapping scheme (mm. 48-71.) From generation 170, cell density decreases until the end, reaching a minimum at generation 289, the point were the automaton achieves an unchanging state. Musically, note values start to get shorter (mm. 81 and following.)

At generation 157 a glider is created in the lower right part of the matrix. This glider starts to travel through it until it gets destroyed at generation 279, occupying in its journey the rows assigned to the horn, bassoon, harp, flute, oboe, clarinet, violin I, violin II, and viola (see Fig. 6, generations 160-280.) The effect of the glider is more evident in the music from generation 173 (m. 75,) since the cell density has started to decrease. At this point the glider passes through the rows assigned to the harp, then the flute (generation 185, m. 83,) oboe (generation 201, m. 90,) clarinet (generation 217, m. 96,) violin I (generation 233, m. 100,) violin II (generation 249, m. 104,) and viola (generation 265, m. 109.) From generation 212 (m. 95) the automaton contains only four types of objects: the moving glider, the block and loaf——which remain since generation 123——, in the rows of violin I and viola a toad (a period-2 oscillator,) in the rows of the cello, and two more blocks in the rows of the horn (see Fig. 6, generations 210 and following.) These structures yield orderly patterns in the above instruments (see mm. 95 and following,) only to be disrupted when the glider passes through their assigned rows. At generation 279 the glider hits the toad and both get destroyed, leaving the system unchanged from generation 289 to the end, creating a repeating texture between the horn, violin II, and viola (mm. 114-116)

The overall dynamical behavior of the automaton is also transcribed to the music through the mapping process. It yields a recognizable "form" in the music.

This dynamical behavior may be broadly structured into four sections. The first section, which goes from generation 0 to 75 (mm. 1-24,) corresponds to the evolution of the two gliders and the pentadecathlon. These two gliders move throughout the matrix, creating "duos" between the instruments as they pass through their assigned rows. At the same time, a patterned texture between violin I, violin II, and viola is created by the evolution of the pentadecathlon.

The second section, more unpredictable and chaotic, corresponds to generations 75-210 (mm. 24-82.) Within this section, cell density starts to increases, and hence instrumental texture. We may think of this as a kind of developmental section, with instruments slowly being incorporated to the texture, and reaching a *fortissimo* climax at generation 170 (m. 71.) Incidentally, this climax occurs at the golden section of the piece! (116 mm. x 0.618033 = m. 71.) However, this is only an unexpected (and unprepared) coincidence derived from the particular initial cell configuration, matrix size, and number of mapped generations.

From generation 170 cell density starts to decrease, yielding a thinner instrumental texture. The third section includes generations 212 to 279 (mm. 95-113.) Here, the system is stabilized with a moving glider, three blocks, a loaf, and a toad. The moving glider yields "solos" from the instruments as it passes through their assigned rows, against a patterned texture between the horn, violin II, viola and cello, derived from the blocks, loaf, and toad.

Finally, from generation 279 to the end (mm. 114-116,) the music settles to a repeating pattern between horn, violin II, and viola, determined by the remaining three blocks and loaf.

The particular scale, duration, and dynamic event sets chosen for the mapping process dictate the modal "feeling" as well as the relatively simple rhythmic complexity of the generated music. These event sets were chosen only for the sake of simplicity and for demonstration purposes. Different, more elaborate event sets, such as increasing the number of scales, perhaps including microtonality or even absolute frequency; continuous duration and dynamic values (not quantized,) etc., would undoubtedly affect the character of the generated music. However, the underlying musical structure would still be the same, since it is yielded by the structure and evolution of the automaton.

Furthermore, different starting cell configurations will generate a different evolution of the automaton and hence, different musical structures. Other configurations may lead to static and unchanging structures, highly dynamical and orderly, or even chaotic and unpredictable. This incorporates in the generated music elements of pattern, variation, and development, elements which are, this writer believes, an essential part of music.

BIBLIOGRAPHY

Abelson, H. and A. diSessa. *Turtle Geometry. The Computer as a Medium for Exploring Mathematics*. Cambridge, MA: The MIT Press, 1986.

Anderson, J. T. and C. S. Ogilvy. *Excursions in Number Theory*. New York: Dover, 1988. 139-140.

Barnsley, M. *Fractals Everywhere*. New York: Academic Press, 1988.

Beauchamp, J., A. Horner, and L. Haken. "Genetic Algorithms and Their Application to FM Matching Synthesis." *Computer Music Journal* 17(4) (1993): 17-29.

Bidlack, R. "Chaotic Systems as Simple (but Complex) Compositional Algorithms." *Computer Music Journal* 16(3) (1992): 33-47.

Biles, J. "GenJam: A Genetic Algorithm for Generating Jazz Solos." In *Proceedings of the 1994 International Computer Music Conference.* San Francisco: International Computer Music Association, 1994. 131-137.

Boethius, A. "Fundamentals of Music." In *Strunk's Source Readings in Music History*. Ed. O. Strunk. New York: Norton, 1998. 137-143.

Bolognesi, T. "Automatic Composition: Experiments with Self-similar Music." *Computer Music Journal* 7(1) (1983):25-36.

Boon, J. P. and O. Decroly. "Dynamical Systems Theory for Music Dynamics." *Chaos* 5(3) (1995): 501-508.

Boulanger, R., ed. *The Csound Book: Perspectives in Software Synthesis, Sound Design, Signal Processing, and Programming*. Cambridge: The MIT press, 2000.

Boulez, P. *Boulez on Music Today.* S. Bradshaw and R. Benett, trans. Cambridge: Harvard Univ. Press, 1971.

Briggs, J. *Fractals: the Patterns of Chaos: a new Aesthetic of Art, Science and Nature.* New York : Simon and Schuster, 1992.

Briggs, J. and F. D. Peat. *Turbulent Mirror*. New York: Harper and Row, 1989.

Cage, John. *Silence*. Middletown, Conn.: Wesleyan Univ. Press, 1961.

Cassiodorus, F. "Fundamentals of Scared and Secular Learning." In *Strunk's Source Readings in Music History*. Ed. O. Strunk. New York: Norton, 1998. 143-148.

Casti, J. L. *Complexification*. New York: Harper Collins, 1994.

Clawson, C. C. *Mathematical Mysteries: the Beauty and Magic of Numbers*. New York: Plenum Press, 1996.

Crick, F.H.C. and J.D. Watson. "Molecular Structure of Nucleic Acids. A Structure for Deoxyribose Nucleic Acid." *Nature* 171 (1953): 737-738.

Darwin C. *On the Origin of Species.* Cambridge, MA: Harvard University Press, 1964.

Dewdney, A. K. "Computer Recreations." *Scientific American* (September 1986): 78-80.

Diaz-Jerez, Gustavo. "Fractals and Music." *Electronic Musician* (October 1999): 108-113.

Di Scipio, A. "Composition by Exploration of Non-Linear Dynamical Systems." In *Proceedings of the 1990 International Computer Music Conference*. San Francisco: International Computer Music Association, 1990.

Dodge, C. "Musical Fractals." *Byte* 11(6) (1986):185-196.

__________. "Profile: A Musical Fractal." *Computer Music Journal* 12(3) (1988):10-14.

Erickson, R. *Sound Structure in Music.* Berkeley: Univ. of California Press, 1975.

Fichet, L. *Les theories scientifiques de la musique.* Paris: Librairie philosophique J. Vrin, 1996.

Fitch, J. and J. Leach. "Nature, Music, and Algorithmic Composition." *Computer Music Journal* 19(2) (1995): 23-33.

Gardner, M. "White and Brown Music, Fractal Curves and 1/f Fluctuations." *Scientific American* 238(4) (1978): 16-31.

__________. "Mathematical Games: the Fantastic Combinations of John Conway's new Solitary Game 'Life'." *Scientific American* 223 (April 1970): 120-123.

Glass, L. and D. Kaplan. *Understanding Nonlinear Dynamics*. New York: Springer, 1995.

Gleick, J. *Chaos*. New York: Penguin Books, 1987.

Gogins, M. "Iterated Function Systems Music." *Computer Music Journal* 15(1) (1991): 40-48.

Goldberg, D. and A. Horner. "Genetic Algorithms and Computer-Assisted Music Composition." In *Proceedings of the 1991 International Computer Music Conference*. San Francisco: International Computer Music Association, 1991. 476-482.

Hénon, M. "A two-dimensional Mapping with a Strange Attractor." *Communications in Mathematical Physics* 50 (1976): 69-77.

Hiller, L. and L. M. Isaacson, eds. *Experimental Music; Composition with an Electronic Computer.* New York: McGraw-Hill, 1959.

Horowitz, D. "Generating Rhythms with Genetic Algorithms." In *Proceedings of the 1994 International Computer Music Conference*. San Francisco: International Computer Music Association, 1994. 142-143.

Howat, R. *Debussy in Proportion: a Musical Analysis*. New York: Cambridge Univ. Press, 1983.

Hunt, A., R. Kirk, and R. Orton. "Musical Applications of a Cellular Automata Workstation." In *Proceedings of the 1991 International Computer Music Conference*. San Francisco: International Computer Music Association, 1991. 165-168.

Inokuchi, S., H. Katayose, and Y. Nagashima. "Deterministic Chaos, Iterative Models, Dynamical Systems and Their Application in Algorithmic Composition." In *Proceedings of the 1993 International Computer Music Conference.* San Francisco: International Computer Music Association, 1993. 194-197.

Jones, K. "Compositional Applications of Stochastic Processes." *Computer Music Journal* 5(2) (1981): 45-61.

Kendall, G. "Composition from a Geometrical Model: Five-leaf Rose." *Computer Music Journal* 5(4) (1981): 66-73.

Koza, J.R. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* Cambridge, MA: The MIT Press, 1992.

Kurepa, A. and R. Waschka, "Using Fractals in Timbre Construction: an Exploratory Study." *Proceedings of the 1989 International Computer Music Conference*. Eds. David Butler and Thomas Wells. San Francisco: International Computer Music Association, 1989. 332-335.

Ladd, S. R. *C++ Simulations and Cellular Automata.* New York: M&T Books, 1995.

Lagarias, J. "The 3x+1 Problem and its Generalizations." *American Mathematical Monthly* 92 (1985): 3-23.

Langton, C. "Self-Reproduction in Cellular Automata." *Physica D* 10 (1984): 135-144.

Lincoln, Harry B., ed. *The Computer and Music.* Ithaca: Cornell University Press, 1970.

Lindenmayer, A. "Mathematical Models for Cellular Interactions in Development, Parts I-II." *Journal of Theoretical Biology* 18 (1968): 280-315.

Lindenmayer, A. and P. Prusinkiewicz. *The Algorithmic Beauty of Plants.* New York: Springer, 1990.

Lindgren, K. and M. G. Nordahl. "Universal Computation in Simple One-dimensional Cellular Automata." *Complex Systems* 4 (1990): 299-318.

Little, D. C. "Composing With Chaos; Applications of a New Science for Music." *Interface* 22(1) (1993): 23-51.

Lorenz, E. N. "Deterministic Nonperiodic Flow." *Journal of the Atmospheric Sciences* 20 (1963): 130-141.

Loy, G. "Composing with computers – a Survey of some Compositional Formalisms and Music Programming Languages." In *Current Directions in Computer Music Research*. Eds. M. Mathews and J. R. Pierce. Cambridge: MIT Press, 1989. 292– 396.

Ludwig, F. "Die 50 Beispiele Coussemakers aus der Handschrift von Montpellier." *SIMG* 5 (1903-4): 177-224.

Mandelbrot, B. *The Fractal Geometry of Nature*. San Francisco: W.H. Freeman, 1982.

Mason, S. and M. Saffle. "L-Systems, Melodies and Musical Structure." *Leonardo Music Journal* 4 (1994): 53-58.

May, R. "Simple Mathematical Models with very Complicated Dynamics." *Nature* 261 (1976): 459-467.

McCauley, J. L. *Chaos, Dynamics and Fractals*. New York: Cambridge University Press, 1993.

Millen, D. "Cellular Automata Music." In *Proceedings of the 1990 International Computer Music Conference*. San Francisco: International Computer Music Association, 1990. 314-316.

Miranda, E. "Cellular Automata Music: An Interdisciplinary Project." *Interface* 22 (1993): 3-21.

__________. "Music Composition Using Cellular Automata." *Languages of Design* 2 (1994): 105-117.

Morse, M. "Recurrent Geodesics on a Surface of Negative Curvature." *Trans. Amer. Math. Soc.* 22 (1921): 84-110.

New Harvard Dictionary of Music. Ed. Don Randel. S.v. "Arithmetic and harmonic mean." Cambridge, MA: Harvard Univ. Press, 1986.

__________. Ed. Don Randel. S.v. "Pythagorean hammers." Cambridge, MA: Harvard Univ. Press, 1986.

__________. Ed. Don Randel. S.v. "Pythagorean scale." Cambridge, MA: Harvard Univ. Press, 1986.

Ott, E. *Chaos in Dynamical Systems.* New York: Cambridge University Press, 1993.

Parshall, K. H. "The Art of Algebra from Al-khwarizmi to Viète: a Study in the Natural Selection of Ideas." *History of Science* (June 1988): 129-164.

Peak, D. *Chaos Under Control*. New York: Freeman and Co, 1994.

Peitgen, Heintz-Otto, ed. *Chaos and Fractals: New Frontiers of Science*. New York: Springer, 1992.

Peitgen, H-O. and D. Saupe, eds. *The Science of Fractal Images*. New York: Springer, 1988.

Penrose, R. *The* Emperor's *New Mind: Concerning Computers, Minds, and the Laws of Physics*. New York: Oxford Univ. Press, 1989.

Pickover, C. A. *Computers, Pattern, Chaos, and Beauty: Graphics from an Unseen World.* New York: St. Martin's Press, 1990.

__________. "Million-Point Sculptures." *Computer Graphics Forum* 10(4) (1991): 333-336.

Poincaré, H. *Science and Hypothesis.* New York: Dover, 1952.

Pressing, J. "Nonlinear Maps as Generators of Musical Design." *Computer Music Journal* 12(2) (1988): 35-46.

Roads, C., ed. *Composers and the Computer.* Los Altos, CA: William Kaufmann, Inc., 1985.

Schillinger, J. *The Schillinger System of Musical Composition.* New York: Da Capo Press, 1978.

__________. *The Mathematical Basis of the Arts.* New York: Philosophical Library, 1948.

Schroeder, M. *Fractals, Chaos, Power Laws: Minutes from an Infinite Paradise.* New York: Freeman and Co, 1991.

Schwanauer, S, and D. A. Levitt, eds. *Machine Models of Music.* Cambridge, Massachusetts: The MIT Press, 1993.

Smith, E. Liner notes to W. A. Mozart, *A musical dice game.* Phillips Complete Mozart Edition—Rarities and Surprises 422 545-2, 1991.

Truax, B. "Chaotic Non-Linear Systems and Digital Synthesis: An Exploratory Study." In *Proceedings of the 1990 International Computer Music Conference*. Ed. Stephen Arnold. San Francisco: International Computer Music Association, 1990. 100-103.

Varèse, E. "The Liberation of Sound." In *Strunk's Source Readings in Music History*. Ed. O. Strunk. New York: Norton, 1998. 1339-1346.

Vetterling, W. T., ed. *Numerical Recipes in C.* New York: Cambridge Univ. Press, 1997.

Von Neumann, J. *Theory of Self-Reproducing Automata.* Champain, IL: University of Illinois Press, 1966.

Voss, R.F. and J. Clarke. "1/f noise in music and speech." *Nature* 258 (1975): 317-318.

__________. "'1/f noise' in Music: Music from 1/f Noise" *Journal of the Acoustic Society of America* 63 (1978): 258-263.

Williams, P. G. *Chaos Theory Tamed*. Washington DC: Joseph Henry Press, 1997.

Waldrop, M. *Complexity: the Emerging Science at the Edge of Order and Chaos*. London: Penguin Books, 1993.

Weaver, W. *Lady Luck: The theory of Probability.* New York: Dover, 1982.

Wolfram, S., ed. *Theory and Applications of Cellular Automata.* Singapore: World Scientific, 1986.

__________. "Universality and Complexity in Cellular Automata." *Physica D* 10 (1984): 1-35.

Xenakis, Iannis. *Formalized Music; Thought and Mathematics in Music*. New York: Pendragon Press, 1992.

Zarlino, G. *Institutioni harmoniche*. Trans. G. Tomlinson. Ridgewood, N.J.: Gregg Press, 1966.

Gustavo Diaz-Jerez

1970	Born 27 February in Tenerife, Spain.
1984-88	Attended High School in Tenerife, Spain.
1988-92	Attended Manhattan School of Music, major in piano.
1992-94	Attended Manhattan School of Music, major in piano.
1993	Winner of the Manhattan School of Music Concerto Competition.
1993	2nd Prize winner in the Maria Canals International Piano Competition (Barcelona.)
1994	B.M. and M.M., Manhattan School of Music
1994	Manhattan School of Music's "Harold Bauer" Award.
1994-98	Attended Manhattan School of Music, major in piano.
1996	2nd Prize winner in the Viña del Mar International Piano Competition (Chile.)
1997	1st Prize winner in the Palm Beach International Piano Competition.
1998	Laureate Prize in the XIII Santander International Piano Competition (Spain.)
1999	Article: "Fractals and Music," *Electronic Musician* (October 1999): 108-113.
2001	D.M.A., Manhattan School of Music.