Computer Generated Music Composition

by

Chong (John) Yu

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degrees of

Bachelor of Science in Computer Science and Engineering

and Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May 28, 1996

Author ____
Department of Electrical Engineering and Computer Science
May 28, 1996

Certified by ____
Professor Tod Machover
Thesis Supervisor

Accepted by ____
F. R. Morgenthaler
Chairman, Department Committee on Graduate Theses

Computer Generated Music Composition

by
Chong (John) Yu

Submitted to the
Department of Electrical Engineering and Computer Science

May 28, 1996

In Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science in Computer Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

A computer composition engine has been designed in an attempt to capture basic music composition and improvisation knowledge in a variety of styles. The output is based solely on user-controlled parameters and low level rules embedded within the generation engine. Although the generator itself is platform independent, current versions exist for both Windows and Java, using MIDI and sound file output respectively.

Thesis Supervisor: Tod Machover
Title: Professor, Media Arts and Sciences

Acknowledgments

I am indebted to Professor Randall Davis for teaching the class that inspired this project. To Professor Tod Machover, for giving me the chance to turn the idea into reality, whose support made the entire project possible. To Ben Denckla and Patrick Pelletier, for Rogus McBogus on which the early versions of this project ran, and for their knowledge of all things low level. To Charles Tang, whose versions of two pentatonic scales are now sitting in the generation engine. To Eric Metois, for his Java and DSP expertise that made the Java version and the ''Singing Tree'' possible. To Will Oliver, for invaluable help in the design and testing of the ''Singing Tree.'' To the rest of the Brain Opera team, for being a great bunch to work with, and for not stringing me up in a tree. And lastly to my agent, my producer, the direc -- oh wait, wrong speech.

1. Purpose

Imagine a person who needs a short piece of music. She is trying to put together a video game, a graphics demonstration, or a television advertisement that calls for a tune of some sort to fill in the background. However, being a programmer, theorist, or ad executive, composing music is not her strongest point. That is the motivation behind this project -- to provide a short tune for someone who has only limited musical knowledge. Although the music currently provided may only be worthy of prototypes and is neither a substitute for a professional ‘‘tune writer’’ nor grand works of art, with further extensions, such performance may not be completely beyond its scope.

2. Background

Originally an expert systems class project written in Lisp, Sharle[1] has been greatly transformed and reshaped, both in language (C++) and structure, since I joined Professor Tod Machover's *Brain Opera* project. The *Brain Opera* was envisioned to be an ambitious exhibition and performance of computer based music and graphics, scheduled to premiere in July of 1996. As a result, a MIDI representation of the music has become critical. Sharle was at first attached on the lower end to a Macintosh-based MIDI system written by Ben Denckla and Patrick Pelletier, with input taken from a MIDI keyboard. However, care has been taken to keep the various interface and decision-making sections as independent as possible. The current version runs under Window NT; input is taken from a Windows-based interface, and output (in MIDI) can be sent either to the local synth or an external device.

[1] For lack of a better name, I chose a shortened version of "Sharon Apple", the name of a fictional "character" (from *Macross Plus*) -- a computer program / entity / singer / performer that became sentient.

The immediate precursor of this work at the Media Lab was done by Alexander Rigopulos and was finished in 1994. His system was based on the notion of seeds -- music was defined in terms of a set of parameters such as activity and coloration. The user was able to affect the outcome of the system by controlling these parameters. The current system is similar to his work in that respect. However, his system was limited to one or two layers (ie. one or two simultaneous, but independent voices) and made extensive use of looping pre-composed sequences.

Fumiaki Matsumoto's ''Drum-Boy'' system attempted to do with rhythms what Sharle attempts to do with pitches as well. That is, it gave the user a set of high-level controls that even non-musicians should be able to understand (descriptions like ''graceful'' and ''mechanical''). The user would then be able to manipulate these controls to produce a rhythm to her own liking. The difficulty lies in deciding which high-level controls are appropriate and how to relate them to lower level descriptions.

Both David Cope and Robert Rowe designed systems that created music by analysis of other music. While Cope's system took input from classical compositions, Rowe looked at live input from a performer. Rowe drew a distinction between transformative and generative music composition. While his ''Cypher'' system contained elements of both, it was primarily a transformative one -- listening to the input from the user, pushing the input through a series of transformations, and then outputting something derivative, although not necessarily reminiscent.

I do not believe that there is a clear distinction between transformation and generation. Enough transformation can produce something as new and as unrecognizable as generated music. However, generative systems are usually characterized as systems which do not involve any pre-composed material, using only the most basic elements, such as scales, as the building blocks. This purely generative method is what Sharle is based on; I did not want possible deficiencies in the system to be masked by well composed sequences or by thinly transformed live input.

One of the challenges faced by many expert system designers is that they are often met with opposition and skepticism from existing human experts. The designers find it important to emphasize that they are not trying to replace human beings, but rather create a tool that is meant to either make the job of the experts less cumbersome or to help educate would-be experts. Rather than seeking to replace composers, Sharle relies on them for its knowledge.

It can be seen as a new medium for composition -- just as tapes and CDs became a new medium for performers -- or Tcl and Visual Basic creating new directions for language and library design, instead of replacing old programmers. Because this project does not espouse any particular music theory, no attempt is made to judge the relative merits of various compositional approaches. Instead, the design attempts to include any and all compositional knowledge that can be readily articulated into a concrete rule-like form (starting with my own, and extendible to that of others, if they have the time and desire).

3. Models of Interaction

A ''real'' composer knows what music she likes and how to produce it. In a sense, a radio listener is also composing in his own way. He knows what radio stations he likes and how to tune into those stations. This project attempts to cover part of the middle ground between the professional composer and the radio listener. Just as the piano attempted to facilitate musical performance by restricting the tones that can be played, Sharle attempts to do the same for composition / improvisation.

3.1 The Instrument Model

In one sense, any interactive system that produces music in real-time can be called an instrument. Various systems however require different levels of interaction. Systems that perform live transformations -- relying on constant musical input from the user -- are true instruments in the sense that a normal instrument requires constant interaction in order to produce anything interesting. On this criterion, Sharle was not designed to be a pure instrument. In fact, if the user is changing too many parameters every second, the results may well resemble a jumble of half-started sentences with little evidence of continuity. Instead, Sharle operates somewhere between the radio model and the instrument model.

3.2 The Radio Model

A radio is on the other end of the spectrum. It operates largely independently from the user; the most important interaction (tuning to different stations) is fixed on a number of discrete choices. Interaction is measured in terms of minutes and hours -- the length of songs and radio programs. All output is pre-composed. However, a radio does allow a certain degree of transformative interaction with controls like volume and treble / bass.

The radio's primary parameter is of course frequency. Frequency is a sort of high level control that allows the listener to choose between various styles of music (and non-music) -- alternative, hip-hop, top 40, classical, jazz, techno, country, talk-radio, etc. This project attempts to design an analogous high level interface, but one that also allows the user to have more direct say in what music is produced.

While Sharle allows more direct control than the radio, it is also abstracted to a higher level than the instrument. If the user does not provide any input, it does not stop and wait. Instead, notes continue to be played, reflecting the current parameters in the system, and yet the music does not fall into an obvious loop. Typical interaction rates are geared toward anywhere from once every 10 seconds to once every 10 minutes (or whenever the user becomes bored). As a compositional tool, this time frame allows the user to listen for a few moments, judge the desirability of the output, and decide on additional parameter changes. As entertainment, the user is allowed to engage in other activity while listening, without the pressure of having to constantly ''babysit'' the controls.

3.3 Autonomy

The input to Sharle allows the user to guide the music being composed without having to deal with actually selecting specific notes and rhythms for the entire composition. The interface is little more than a way to change various parameters in the system. Some parameters are continuous (for example, consonance and tempo), while others discrete (for example, scale).

The primary way the user can influence the music composed is by choosing higher level characteristics such as scale and tempo. However, an even higher level of control is possible. By putting together various combinations of scale, rhythm, and other traits, individual ''songs'' analogous to those on the radio can be produced. The parameters that make up these songs can then be mixed together in various degrees (see section 9.2). There is no notion of one set piece of music that embodies all of the generator's output; instead, the extremes in Sharle's parameters are designed to enable manipulation of one ''piece'' into another entirely different one.

Sharle provides the user with a choice between varying levels of computer independence. At the highest level, the music generation behaves much like a radio, allowing the user to engage in other activities while listening, interacting only occasionally in response to particularly desirable or undesirable music. At the lowest level, the user is only a few steps up from true composition, deciding everything from pitch direction to rhythmic structure.

4. Knowledge Base

There are many tools available to researchers in artificial intelligence, from grammars to neural nets, Markov models and more. However, the most successful commercial applications have tended to come from the expert systems field. Instead of attempting to model low-level thought and gather knowledge through pattern recognition, Sharle was designed with expert system methods (that is, explicit rules that attempt to summarize higher level thought).

Although originally constructed with TMYCIN, the rules, however, are no longer written in the traditional "if-then" form. Instead they are various combinations of equations, random number generators, and algorithmic functions that try to define each piece of knowledge. C++ has made representation of algorithmic knowledge easier, however it is at the cost of the simplicity that would have it easier for non-programmers to extend the knowledge base. The design of a knowledge acquisition meta-language easily usable by professional composers to articulate their knowledge is a direction for possible future work.

The decision making process is based entirely on the rules in the system and the parameters that the user has given. These parameters are fed into various equations as variables or cause the system to branch to differing sets of rules.

5. Main Design Issues

This section describes some of the principles that guide the structural and algorithmic design that will be mentioned in later sections. Many of these involve tradeoffs and sacrifices that had to be made in order to realize one specification at the expense of another. In general, decisions have been made to err on the side of prudence rather than on the side of novelty.

5.1 What is Acceptable?

Accessible is the word often used to describe music that is generally accepted by the average member of an audience. Though it may not be well loved (''muzak'', ''elevator music'', etc.), ''accessible'' music is generally not disputed as being *music*, while other forms -- minimalism, serialism, certain genres of popular music -- are often labeled by various sections of the general public as questionable music or even non-music. Because the target audience in this case is not music aficionados but the general masses -- to bridge the gap between composers and those with no compositional experience -- accessibility won out in deciding whether to focus on accessibility or innovation. As a result, the default tends toward regularity and tonality. However, Sharle's parameters are not limited so much as to prevent all deviations from general ''accessibility.''

5.2 Unity vs. Variation

The balance between unity and variation is a constant source of push-and-pull tradeoffs in the generation algorithm -- on the one hand, repetition, predictability, and expectation play a large role in distinguishing music from the chaos of noise; on the other hand, variation, change, and the unexpected are vital in preventing boredom and maintaining interest.

Repetition is the primary tool Sharle uses to unify successive parts of the music. It establishes something familiar, something a listener will come to expect. When that expectation is met, the tension is satisfied. However, if it is always met, monotony sets in. Therefore changes are made continuously, though not too quickly. A number of parameters allow the user to adjust the tradeoff between unity and variation including rhythm consistency, change, cohesion, consonance (sections 7.1.5, 7.1.8, 7.1.1, and 7.2.1 respectively).

Another of the goals of Sharle's generation engine is to avoid producing the overly mechanical or predictable music that automatic computer algorithms can easily lead to. Much of this is accomplished largely through randomizations specifically designed to avoid over-repetition in places such as rhythm generation (7.1.1).

5.3 Randomness in Decision Making

Many of Sharle's decisions rely heavily on randomization. However, the decisions are not completely random -- in the sense that all possible outcomes have the same probability. Instead, randomness is one of the methods used in turning continuous parameters into discrete choices. For example, if choice A has a 90% weight and choice B has a 10% weight, the decision is made by choosing a random number between 1 and

100. Choice B is taken only if the random number was over 90. Although choice A is heavily favored, it is not guaranteed. Interactions are usually much more involved than in this example however, often involving multiple random variables.

The use of randomness means that Sharle's output is not deterministic. The same situation at a different time will produce something similar, but not an exact duplicate -- somewhat analogous to the creative process -- had I been writing these words the day before, I would probably have used a different word here or changed a term there.

5.4 Moderation vs. Generality

One of the major tradeoffs in the generation algorithm results from preventing the generation from ''going wild'' so that acceptable music is the norm rather than the exception. Limitations are enforced to prevent extremes; for example, consecutive notes from a layer (see section 6.4) are never more than an octave apart (8.1.4). One can easily argue that a great many composers break these limits and still produce ''acceptable'' music. Although it may exclude great music, enforced moderation is invaluable in excluding noise as well.

5.5 Simplicity vs. Specificity

Another design issue deals with how much specific control should be given to the user. Nearly every variable in the generation algorithm can be made into a parameter. However, presenting the user with too many controls -- both broad and narrow -- can do more to obscure the effects of the parameters than to help. As a result, parameters like cohesion (7.1.1) play many different roles in the generation algorithm.

5.6 Melody vs. Accompaniment

Although the distinction can be easily blurred, Sharle does attempt to distinguish between melody and harmony in the interest of unity. One layer and its corresponding line is distinguished from the others as the melody. In order to keep them related to the melodic layer, the other harmonic layers play a subordinate role with respect to pitch (see 8.1.7) and line repetition (see 6.6).

6. Data Hierarchy

Sharle's generation engine has just two main parts: (1) deciding what (if any) notes to *start* playing at a given moment, and (2) deciding what (if any) notes to *stop* playing at a given moment -- corresponding to the MIDI messages of note-on and note-off.

Inside the generator is a hierarchy of data objects. Each level reflects common characteristics (such as rhythm or scale) shared by the music in that grouping. At the low level, the user would be working with the characteristics of individual lines and notes, while at a higher level, only the characteristics of sections (or ''movements'') could be directly affected.

6.1 Note

On the lowest level is the basic note object. It contains all the information needed to describe and output a note in MIDI:

1. The pitch - a MIDI note number.
2. The volume - represented by one of five values, from very soft to very loud. This is not a continuous range of values, because, while a difference of 1 in MIDI note

number is very important, a slight difference in MIDI note velocities is much less noticeabie.

3. The duration -- a measure of time that depends on the tempo (see section 7.1.4).
4. The instrument - a MIDI program number.

In addition, the time at which the note was started is also stored, so that its relation in time with respect to other events can be determined.

6.2 Line

The second lowest level in Sharle's hierarchy is the line object -- simply a list of notes, stored in the order they were played. Although each line can only generate one note at a time, there can be many lines active at once. The length of a line is the same as that of its basic rhythm (see 6.3). In order to see what other lines are doing and to check higher level parameters, each line object contains access to its parent compound line (6.5). Each line also contains has access to its layer (6.4) characteristics, to guide note generation.

6.3 Rhythm

Tangential to the hierarchy, a rhythm object is associated with each line object. A rhythm is a series of attacks (MIDI note-on events) that are played once for a given line, then repeated for the next line. While the attacks do not have to be periodic within the rhythm object itself, periodicity is created by repetition of the rhythm. Anything related to time and volume in a line's note generation process is determined by the line's rhythm object. Besides a list of when attacks should take place and how loud they should be, a rhythm object also contains:

1. Timing information that keeps track of *which* attack was last played and *when* it was last played.
2. Tempo information (see 7.1.4).
3. Three parameters used in rhythm generation: density, length, consistency (see 7.1.5-7.1.7).

6.4 Layer

Another data object tangential to the hierarchy, a layer is analogous to one violin in an ensemble. The layer object contains the characteristics of the violin player. A line (6.2) is analogous to the notes that the violin plays for some given segment of time. As time passes, the violin may be playing new lines, but it's still the same violin player (ie. the same layer). The data in the layer object determines how each successive line of that layer will behave. A layer stores the rhythm information its lines will use as well as the following parameters: consonance (7.2.1), direction (7.2.2), pitch center (7.2.3), and instrument (7.2.4). In addition, a layer can also be turned off, causing its lines to produce no notes.

6.5 Compound Line

For lack of a better term, the list of all the lines (6.2) that are playing simultaneously is called a compound line. This is the hierarchical level one step above that of the line. While a layer (6.4) describes lines horizontally (ie. successively, one line after another), a compound line groups lines vertically (ie. a set of concurrent lines). Because all its lines are being played simultaneously, the temporal length of a compound line is the same as that of the lines in its list. Compound lines also have access to higher level parameters through their parent stanzas (6.6). The compound line object also keeps track of whether a cadence (see 8.1.1) is to be played, affecting all the individual lines in its list.

6.6 Stanza

A stanza is the next level up in the hierarchy. As in poetry, where a stanza is a small collection of lines, Sharle's stanza objects are a small collection of compound lines. Stanzas group compound lines that are closely related. To promote unity, accompaniment lines are repeated within a stanza; they are only re-generated at the start of a new stanza. The number of successive compound lines within each stanza depends on the cohesion parameter (see 7.1.1). In addition, cadences (8.1.1) can only occur on the last compound line of a stanza -- marking its end. Stanzas also maintain layer data that is copied into each successive stanza.

6.7 Section

The section is the highest level in Sharle's current hierarchy. In fact, there is only one section into which stanzas are stored. Ideally, however, more levels above the section object would be created, each new level representing more abstract similarities in the music it encompasses. In a finite project however, the abstraction had to stop somewhere. As a result of having only one section object, changes to the section parameters are not saved in a history (see section 8.1.6) of sections, but instead the changes only replace the current state of the section.

Besides maintaining a list of stanzas, the section object also keeps track of high level parameters that have impact over all current generation in all layers: the scale

(7.1.3), key (7.1.2), cohesion (7.1.1), and change (7.1.8) parameters.

7. Parameters

Parameters are the controls on which all of Sharle's generation engine is based. However, even without user input to set parameters, the generator can operate well enough on its own based on default parameter values. The parameters described in this section are the atomic ones -- they do not affect the values of other parameters (with the exception of the change parameter, see 7.1.8) and are used to determine lower level variables not available for control by the user. Sections 9.1 and 9.2 describe the non-atomic controls.

7.1 General Parameters

General parameters are not specific to any one layer. Instead, when they change, all layers are affected. Although it is possible to make these parameters specific to individual layers, the added specificity does not offset the additional complexity (both in implementation and control) to warrant the move.

7.1.1 Cohesion

Cohesion is the parameter primarily responsible for governing the unity vs. variety tradeoff on the line level. When cohesion is high, unity is favored, and when it is low, variation is favored. This one parameter plays many different roles in Sharle's generator. Of course, these various roles can be broken down into more specific parameters; however, in the interest of simplicity, the roles were united.

1. Line repetition: When a stanza creates a new melodic line object, cohesion is consulted. The higher the cohesion, the more likely that a previously played line will be copied (as opposed to generating a new set of pitches).
2. Line choice: The higher the cohesion, the more likely a recently played line will be copied (as opposed to one further in the past).
3. Offset: If a previous line is being copied, a low cohesion increases the chance and amount of a pitch offset. The offset varies the previous line by transposing it up or down some number of half-steps (however, the pitch is still matched to the scale according to the consonance parameter, see section 7.2.1).
4. Amount: If a line is being copied, the percentage that is actually copied also increases with cohesion. Low cohesions cause more of the previous line to be ignored, re-generating new notes to replace the ignored ones.
5. Rhythm tweaking: To avoid sounding too mechanical (see 5.2), low cohesions also cause more occasional attacks to be randomly inserted into Sharle's rhythm generator output.
6. Stanza length: High cohesions also cause stanzas to be longer, thereby increasing the number of compound lines which share similar characteristics.

7.1.2 Key

The key parameter is not quite so interesting; it is merely a simple transposition of pitches. The same tune in different keys adds a little more variety when used in contrast to other recently played sections. However, this parameter was added mainly for completeness-of-control's sake. Key (an absolute parameter) and scale (a relative parameter) are used together in pitch determination.

7.1.3 Scale

The scale is the most ''pre-composed'' of the parameters. Sharle currently has a total of six scales -- two versions of a major scale, two versions of a minor scale, and two versions of pentatonic scales. While these six scales are sufficient to produce a variety of music, there is no reason why more can not be added. A scale is represented by an object that contains 12 values corresponding to the 12 half-steps in an octave. Each value represents a priority: the higher the priority, the more often that tone is played. So, when ''composing'' a scale, the root / tonic is given highest priority. Sharle uses five different priority levels. The highest for the root, the lowest for chromatic tones. The middle three priorities differentiate between frequent, common, and occasional tones. For example, one version of a major scale looks like this: 042412414243, 12 rankings of tones from 0 to 4. 0 represents the root and has a unique priority ranking. 1 is given to the rest of the major triad. 2 to most of the other non-chromatic notes, etc. Tones are thus tied to probabilities that will be used later in combination with the consonance parameter (section 7.2.1).

7.1.4 Tempo

The tempo parameter is simply a number that determines the length of the atomic unit of time on which the rhythm generator is based. All events occur only on these atomic units of time, where there are slots into which potential notes can be placed. Tempo changes nothing else other than the speed at which notes are produced. Pitch and relative durations are not affected.

7.1.5 Rhythm Consistency

The first rhythm generation parameter is consistency. The higher the consistency is, the more likely successive rhythm attacks will fall at regular intervals. The lower it is, the time interval between successive notes becomes more irregular. The upper bound on consistency is (of course) beats that occur exactly after a fixed period. The lower bound allows intervals between attacks (determined randomly) to range from one atomic unit of time (determined by tempo, 7.1.4) up to twice the average interval. Consistency also affects attack volumes, ranging from all medium volumes at top consistency to allowing an equal chance of light, medium, or heavy volumes for minimum consistency.

7.1.6 Rhythm Density

Density determines the likelihood that an attack will be placed in any given slot. At maximum density, every slot will be filled with an attack. At minimum density, the vast majority of available slots for attacks will be left blank. Because the period between slots is determined by tempo, a fast tempo and low density can have a similar number of notes per unit of real time as a slow tempo with high density. The difference however is that density affects number of notes per line, while the tempo parameter affects how quickly lines are played -- if density is the amount of traffic, tempo is how quickly the traffic is moving. A high tempo will speed up the generator's progression through lines and stanzas, while a high density will not.

7.1.7 Rhythm Length

The length parameter determines the number of available slots per line in which attacks could be placed. While fast tempos hurry line progression, longer lines slow down progression -- increasing the length of the road on which the traffic travels.

Rhythm length and density combine to determine how many actual notes will be played for a given line.

7.1.8 Change

The change parameter is the rate of self-mutation. Sharle attempts to prevent boredom by periodically changing existing parameters. As each stanza is finished, Sharle checks the change parameter to see if it is time for a change. The lower this parameter, the more likely the generator will just continue on as if nothing happened. The higher it is, the more likely the current stanza will be ended with a cadence (8.1.1); then a random set of parameters are changed randomly. The range from which mutated values are chosen is subject to limits -- determined empirically -- in to prevent extremes in behavior (5.4). There are a few parameters that are never mutated however: the change parameter itself (to avoid recursive behavior) and the instrument parameter (because of hardware differences, see 7.2.4). The number of layers that are actively generating notes is also not changed, in order to limit the extent of self-mutation.

7.2 Layer Specific Parameters

These parameters affect only one layer each. Each layer carries its own set and is able to act relatively independently from the other layers based on the layer specific parameters. Their purpose is to allow each layer to behave as one among many members of an ensemble -- able to work independently, but in the end, still tied to the same purpose.

7.2.1 Consonance

The consonance parameter is closely related to the current scale (7.1.3). If consonance is high, notes generated tend to be non-chromatic -- bounded above by the limit of producing only the root note (at various octaves). At low consonance, equality among notes appears and root notes begin to appear with frequencies similar to chromatic ones. After each tentative pitch is generated, it is ''rounded'' to nearest pitch which has a given scale priority or higher. A high consonance parameter forces that priority to be high. At very low consonances, priorities are ignored.

7.2.2 Direction

The direction parameter is one of the two primary ways the user can directly control pitch values. Direction is used as part of the randomized function that determines the position of a new note relative to the previous note. For a medium direction value, the new note is just as likely to be higher as lower than the previous one.

7.2.3 Pitch Center

As the second major way the user can change pitch values, the pitch center parameter affects the overall pitch range from which notes will be generated. While direction is a relative (and temporal) pitch parameter, pitch center is absolute. As each line ends, pitch center is consulted to determine the starting point of the next line. However, to avoid the discontinuity of sudden jumps in pitch, the starting point is chosen to be a pitch between that of the center and the last note on the previous line. From that starting point, direction takes over pitch movement until the start of the next line is reached. This parameter is a MIDI note number -- all else being equal, pitches will tend to be around this number.

7.2.4 Instrument

The instrument parameter corresponds merely to a MIDI instrument number. Thus different layers can be distinguished by different voices. Since the sound produced by each instrument number depends heavily on the MIDI hardware, Sharle does not attempt to regulate instrument choice. Like overall volume, control over instrument choice is entirely up to the user; they are both very straightforward. Each unique number represents a different instrument, and it is up to the MIDI equipment (or whatever other interface) to make use of it.

7.2.5 Rhythm Modification

Sharle was designed with the notion of a primary rhythm (which is determined by the three non-layer rhythm parameters -- consistency, density, and length; see 7.1.5, 7.1.6, and 7.1.7). All other accompanying rhythms are modifications on the primary rhythm using the rhythm style parameter and the rhythm style argument parameter.

1. The first style, of course, is playing the unmodified primary rhythm. The argument parameter is ignored.
2. A second style, places attacks between the attacks of the main rhythm. The argument determines exactly when -- whether to place attacks at 10%, 50%, or 90%, etc. of the gap between the successive attacks of the main rhythm.
3. Another purely mathematical style, produces a regular beat. The argument determines the length of its period.
4. Finally, a fourth modification produces a deterministic pattern that varies with the main rhythm's parameters. The additional argument affects its periodicity.

All these variations share the characteristic that they have the same temporal length as the primary rhythm, so they can all be cycled without going out of phase. To avoid overemphasis and to avoid drowning out the main rhythm, the volume of the variations is made to be lower.

8. Execution

The basic interaction with ɔharle's generation engine is described below:

1. What time is it?
2. Ask the generator if there is a new note to play at this time.
    a. If so, play the note and repeat step 2.
    b. If not, continue on.
3. Ask the generator if there is a new note to stop playing at this time.
    a. If so, stop the note and repeat step 3.
    b. If not, continue on.
4. Wait for a short unit of time.
5. Return to step 1.

Other than controls to change the generator's parameters, the only input to the generator is a measure of thc passage of time. The only outputs are notes to start and stop. This simplicity helps in portability (section 10) -- steps 2a and 3a (playing and stopping a note) are purely interface dependent. If instead of playing MIDI, we just want to print out a score, steps 2a and 3a are all that need to be changed... while the generator can be left alone.

8.1 Behavior

This section describes some of the specifics of the engine's behavior -- involving practical concerns (like keeping a history and turning notes off), creation and generation (of rhythm and pitch), and holding the music together (through copying and pitch matching).

8.1.1 Cadence

Whenever Sharle is about to mutate its parameters (see 7.1.8), a cadence is used to signify the end of the previous set of parameters -- to give the sense of separation between sections and to help prepare the listener for parameter change. Cadences only occur on the last compound line of a stanza. Sharle uses four characteristics to mark a cadence, all intended to increase the sense of finality.

1. Volume of each attack is increased.
2. The tempo slows, from normal to half the normal tempo by the end of the compound line.
3. The compound line ends on the root note.
4. An artificial pause in note generation is added after the end of the stanza. The duration of this delay is dependent upon the current tempo (7.1.4), plus a randomized factor used to avoid becoming too mechanical (5.2).

8.1.2 Rhythm Generation

Because rhythm is one one of the primary sources of regularity, rhythms are not constantly being regenerated. Generation is done only when one of the consistency, density, or length (7.1.5-7.1.7) parameters change. When this happens, a new fixed rhythm is randomly generated from the new set of parameters and is reused until the next time one of these parameters changes.

8.1.3 Durations and Note-off Timing

Durations are less regulated due to the effects of different instrument sounds (7.2.4). While some instruments have a naturally quick decay time, others have unlimited sustain. The result is that actual durations heard and technical durations based on note-on and note-off timing do not always match. The main concern in setting durations and sending the resultant note-offs has been to prevent sustained instruments from playing notes that blend together too much, preventing listeners from making out the attacks of new notes. Durations were therefore chosen so that notes usually end when the next attack on that layer occurs. Durations can be long enough such that one note from a previous line lasts into the next line. As a result, looking only at the notes of the current lines will miss the notes that have crossed line boundaries. Therefore, a list of notes that are currently on is kept separate from the history of notes that have been played in the past (see also 8.1.6).

8.1.4 Pitch Generation

After the rhythm generator has determined that an attack must occur (returning a duration and volume), a pitch must then be found for this note. There are two methods Sharle uses in creating a new pitch -- pure generation and line copying.

In pure generation, a random step size is picked, favoring small steps and in the direction determined by the direction parameter (7.2.2). This step size is added to the previous note and then rounded according to consonance (7.2.1).

8.1.5 Line Copying

During the copying process, the line being copied is asked for the corresponding notes. In order to add variety, a possible offset, up or down, is added to each of the notes (see cohesion, 7.1.1). However, in order to prevent dissonance (which can occur, for example, after an offset or when copying a line that was generated under a different scale or key), copied pitches are also rounded based on consonance (7.2.1).

8.1.6 History

In order to make line repetition and variation possible, Sharle records a history of the music generated so far. However, because computers are not possessed of infinite memory, Sharle must avoid crashing after executing for extended periods of time (due to lack of memory). As a result, Sharle periodically ``packs'' its history of stanzas by deleting half of them. However, in order to lessen the lack of long term memory (which would result if the earlier half is deleted) and to prevent the absence of short term memory (which results if the recent half is deleted), every other stanza is deleted -- thus helping to preserve older music a little longer, while still keeping a good deal of younger music around.

User input always directly replaces the current state of the parameters, but (at least for stanzas and lower levels in the hierarchy), parameters of previous stanzas, lines, etc, are safely saved in the history (at least until the next packing). Only the current state can be ``corrupted'' by user input. When Sharle's self-mutation occurs however (as opposed to asynchronous user input), parameters are only changed when new stanza objects are created, so no characteristics of the previous stanza object and its components are lost.

8.1.7 Priority Matching

Another source of unity (see 5.2) is the matching of scale priorities (7.1.3). When an accompaniment line is being played, Sharle looks for the last melodic note that was played (the last note from the melodic layer). The accompaniment pitch is then rounded to one whose priority is similar (although not necessarily identical) to that of the melodic pitch. However, in order to maintain the dominance of higher priorities, there is a chance that accompaniment pitches with high priorities will not be rounded to match a melodic pitch with lower priority.

8.2 Walk-Through

In this section, an example of the generation of a single typical note is described. First the current time is given to the section object. The section object checks to see if a new stanza needs to be created and queries the current stanza object. The current stanza has not reached the last of its allotted compound lines, so it replies in the negative. Having no stanza creation to do, the section passes on the time to the current stanza object.

Similar to the section object, the stanza object starts by checking to see if a new compound line needs to be created and queries the current compound line. The compound line object in turn queries each of its line objects. The line objects determine from their rhythm objects that they are not yet finished. As a result or having no creation to do either, the stanza passes on the time to the current compound line object.

The compound line asks each of its line objects for a note, starting with the melodic line. Line copying information (8.1.5) was determined at the start of this compound line. This line copying information and the time is then passed on into the line object.

The line object determines that it has not yet played a note at this time marker and that the layer that it belongs to is active. The line then obtains rhythm modification data (7.2.5) from its layer object; the parent compound line informs the line object that the generator is not in the midst of playing a cadence (8.1.1). This information and the time are then passed into the layer's rhythm object -- asking if an attack is to be played.

Because a cadence is not being played, the rhythm object does not modify the current tempo. The rhythm modification data indicate that the primary rhythm is being played. The time and the previous attack's timing data are compared to the sequence of relative attack times and attack volumes that has been stored in the rhythm object since it was generated. This indicates that the time for the next attack has been reached. The previous attack's timing data is updated. A duration is calculated that will keep this note playing until the next attack in the sequence (see 8.1.3). The duration data and volume data are then returned to the line object.

Because the line copying information do not indicate any repetition, the line object generates a new pitch -- a scale priority is calculated from a random number and the layer object's consonance value. A pitch distance that favors low numbers is then calculated from a random value. Another random number and the layer object's direction parameter determine that the new pitch should be higher than the last. The distance is then added to the previous pitch, and the result is rounded to the nearest one which has the given scale priority or higher.

The rhythm object determines that this is not the last note of the line; therefore, the pitch is not made more consonant. In addition, since the current line belongs to the melodic layer, the pitch priority is not matched (8.1.7) to the previous melodic pitch.

Finally, the note is added to the list of currently active notes and to the history of notes kept in the line object.

This note is then returned back up the hierarchy, to the compound line, to the stanza, to the section, and to the top level, where it is sent to the MIDI interface to be played as audio.

9. Interface Hierarchy

Sharle was designed to operate with varying levels of independence from the user. At the highest level, the interaction required should be minimal. At the lowest level, control over a large set of parameters should be given to the user so that generation can be directly tailored to the user's preferences.

The set of controls at the highest level is also the simplest, in order to avoid intimidation. The goal is to allow any first time user to use it. Like a non-pilot sitting down in front of a cockpit, a first time user that is presented with too many controls may not know where to start. Timidly changing one low level parameter -- that produces only a subtle change in the music generation -- may result in the first time user not noticing a difference at all.

Given only a few controls (such a joystick and buttons for a flight simulation video game), the user knows exactly what to experiment with. After the user is comfortable with the high level controls, the next lower level opens up more possibilities, and by then, the user will hopefully know what to listen for and how the new controls relate to the old ones. Even just seeing higher level controls without using them can provide valuable information about which controls have the most effect.

Sharle's interface is separated into four levels. The parameters in the top two levels directly affect the parameters on the bottom two levels. Thus, by looking at just how the lower level parameters are affected, the user can learn to recreate the effects of the high level controls.

9.1 Level I -- The Colors

At the highest level, there are four ''radio stations'' from which the user may choose, based loosely on four moods -- festive, calm, angry, and sad -- and named after the colors red, green, violet, and blue respectively. These are four different pre-''composed'' settings of the lower level parameters. Although they were chosen to demonstrate a wide range of what is compositionally possible, these four pre-settings are in no way exhaustive. There is nothing unique to these controls; they are fully recreatable using only the lower level controls of levels III and IV -- these controls are just easier. They are an example of the type of higher level composition that the lower levels enable.

A fifth control at the top level is the rate of change parameter (7.1.8), which allows Sharle to become largely autonomous. While the four pre-settings above may represent the typical sound of four radio stations, listeners would quickly become bored if the station is always playing the same song over and over. So when listener is bored, the change parameter can be increased, causing the music to change -- even to styles similar to other ''radio stations.'' When the listener hears a ''song'' that is particularly interesting, this control can be turned down to keep the current music playing a little longer. The rate of change can also be left at a particular value that suits the length of the listener's attention span. This frees the listener to do other things without leaving the music generator in stagnation.

9.2 Level II -- Interpolation

At the second highest level, just two sliders were added. However they add much more control. Each of the four pre-settings in level I represents a point in a space of many dimensions, each dimension representing one parameter. These sliders interpolate between the four points, producing a smooth range of intermediate music between the four discrete "songs." It is not completely continuous however, because scale is represented as a discrete parameter. Scale instead varies among a list of predefined scales.

Among the various parameters, the most conspicuous changes resulting from the interpolation are probably in scale and tempo -- near red and green are variations of the major scale, while violet and blue are minor; red and violet are faster than green and blue. Instruments are also a conspicuous difference between the colors -- whether they are harsh or smooth, drawn out or terse. Drawn out instruments (winds & strings) were used for the slower green and blue to fill up the gaps, but they tend to be too unresponsive to faster speeds. Because each of the four pre-settings has a different number of layers, Sharle does not interpolate between instruments or the other layer specific parameters.

9.3 Level III -- General Parameters

The third level adds many more degrees of freedom. All the non-layer specific parameters are now controllable (see section 7.1). Although these controls are all affected by the level II interpolation sliders, now a change in one of these parameters no longer has to be tied to changes in the other parameters. Although these parameters are more general than those specific to individual layers, the parameters on this level -- like those on level IV -- are atomic; that is, they are not defined in terms of other parameters in the way the controls in the top two levels are.

9.4 Level IV -- Layers

The lowest level of the interface effectively adds an infinite number of degrees of freedom by introducing layers, each layer representing a different independent instrument. The number of layers is unlimited, although MIDI limits the number of instruments to 16 in actual practice. Still we can have 100 separate layers of MIDI piano. Of course, the practical use of too many layers can be limited by both performance speed and the fact that too many independent and unsynchronized layer rhythms may easily lead to a messy jumble of notes.

It is only here that the user can choose new instruments. In retrospect, this is not very user friendly, but nonetheless, keeping all the layer specific characteristics together keeps organization simple.

10. Ports

Sharle was designed with portability and ease-of-integration into other programs in mind. The generation engine itself is platform independent. Platform specific information are limited to the graphical interface, the output of audio, and user input. Two other instances of Sharle's generation algorithm exist -- in Java and in the *Brain Opera*'s ''Singing Trees.''

10.1 Java

Because of the similarity between Java and C++, the language conversion took surprisingly little time. Input is still similar to the Windows version -- by mouse click. However, output is limited by Java's (current) lack of MIDI capability. As a result, MIDI output is imitated through the use of a large number of sound files, each file representing a single note. The primary disadvantage of this method is that output is completely limited by the number and quality of the sound files. Given the entire MIDI range of 128 pitches, multiplied by the different volumes desired, and again multiplied by the number of instrument sounds desired, approximation of MIDI through audio files becomes extremely difficult.

A separate Java version of Sharle also exists that makes use of a Java MIDI package written by Michael St. Hippolyte (for Windows). This version allows the Java applet to have the same controls over MIDI as a Windows application. Currently the two Java versions of Sharle are located at:

http://theremin.media.mit.edu/cyu/Sharle.html

http://theremin.media.mit.edu/cyu/javamidi/Sharle.html

10.2 Singing Trees

Also written for Windows, the ''Singing Trees'' makes use of voice input. Instead of mouse input, users interact by singing into a microphone. A voice analysis package written by Eric Metois extracts voice characteristics from the user's input, and these characteristics are used to recalculate Sharle's parameters. In this version, not all of Sharle's parameters are controllable by the user. Many are set to fixed values in order to present a more regulated and consistent experience to the user.

11. Looking Back

If I were an outsider looking in for the first time, I would probably ask myself, ''So why does this work? Is there a gimmick?'' The gimmick is the expert system gimmick. The computer is expected to learn from the user only in as much as what it can determine from the set of ''questions'' posed by the input parameters -- ''Do you prefer a 0 or a 127?'' It does not start from scratch the way, for example, a neural net might. Specific knowledge is explicitly written in, and the outcome hinges on the interaction between the knowledge embodied in the many rules and the user's input. Decisions are made randomly, but the rules impose limits on the choices to prevent them from going too far. Because perfect imitation only leads to an exact replica, imitation is not the ultimate goal. In a way, creation of something new makes it easier because there is no set pattern that must be adhered to.

Because the boundary between right and wrong is not absolute in matters of taste, there have been many ''good enough''s in the design of the generation engine that I will criticize. In preventing the rhythms from sounding too mechanical, the random additional attacks inserted into the rhythm could have been matched by random deletions as well. Instead, insertion alone was judged to be sufficient in preventing over-predictability. In playing a cadence at the end of a stanza, other cadence styles could have been added that give the same sense of an ending -- for example, having the music trail off at the same tempo until the notes are too soft to be heard. In allowing different layers to play different rhythms, each layer could have been given its own full set of rhythm generation parameters to allow for maximum flexibility in combining rhythms. Instead, the various modifications were considered flexible enough in creating rhythmic contrast. Finally, there is more room for improvement in that instruments can be classified into groups based on characteristics such as amount of sustain and harshness. Again, differences in MIDI equipment will pose an obstacle, but it is not insurmountable.

12. Looking Forward

Much of the knowledge contained in the rule base will be simplistic to musicians, but still the average person often either cannot get easy access to it, or does not care enough to try. Yet they do listen to the radio, to tapes, and to CDs of performers that otherwise cannot perform in their living rooms. If made available, a system like Sharle can help give everyone a bit of compositional experience without a great deal of effort and without looking over the shoulder of a composer.

I was asked by a businessman about the application and practicality of computer music. "Is there a market?" was the basic question, "Do people feel a need for compositional experience? There are so many people who go through life without ever feeling such an urge." My response to him was that new technology creates demand. Before the web, use of the internet was mostly limited to the academic community, but the better interface and ease of use provided by the web browser suddenly put everyone, expert and novice alike on the internet. A simulation of flying a jet cannot compare to the real thing, but for people who may lack either the time or ability to learn to fly... or to compose music, we can give them a chance to taste some of the creative experience.

That computer music will be available in everyone's living rooms is not a thing for the future. It is a thing of the present. Just recently (May 1996), Brian Eno announced the release of *Generative Music 1* for Windows (with SSEYO's Koan software) -- an interactive music program with nearly 150 parameters. Although the work is still limited to envelopes around twelve distinct pieces and up to nine hours of continuous play, the wide availability of fully generative musical software cannot be too much further down the road.

References

David Cope. ''An Expert System for Computer-assisted Composition.'' *Computer Music Journal*, 11:4, 1987.

David Cope. *Computers and Musical Style*. A-R Editions, Inc., 1991.

Randall Davis. ''Expert Systems: How Far Can They Go?'' *Artificial Intelligence Magazine*, 10:1 p. 61-67, 10:2 p. 65-77.

Randall Davis. ''Production Rules as a Representation for a Knowledge-based Consultation Program.'' *Artificial Intelligence Journal*, 8:15-45, 1977.

Damon Horowitz. ''Representing Musical Knowledge.'' Master's thesis, MIT Media Lab, 1993.

Fumiaki Matsumoto. ''Using Simple Controls to Manipulate Complex Objects: Application to the Drum-Boy Interactive Percussion System.'' Master's thesis, MIT Media Lab, 1993.

Marvin Minsky and Otto Laske. ''A Conversation with Marvin Minsky.'' *AI Magazine*, 14:3, Fall 1992.

A. Newell. The Knowledge Level. *Artificial Intelligence Magazine*, Summer 1981, p. 1-20.

Alexander Rigopulos. ''Growing Music from Seeds: Parametric Generation and Control of Seed-Based Music for Interactive Composition and Performance.'' Master's thesis, MIT Media Lab, 1994.

C. Roads. ''Artificial Intelligence and Music.'' *Computer Music Journal*, 2:2, 1980.

Robert Rowe. ''Machine Learning and Composing: Making Sense of Music with Cooperating Real-Time Agents.'' Ph. D. thesis, MIT Media Lab, 1991.

Appendix - User's Guide

This is a short summary of the Windows version of Sharle.

Startup

On startup, the user is asked if MIDI output should be sent only to the sound card, or sent via the sound card to an external synth that is assumed to be be attached to the sound card by a MIDI cable. The generation engine will already be on, using default parameters to produce the notes for a single layer, using instrument 0.

Level I

The controls described in section 9.1 will be immediately visible. The change parameter's slider control defaults to 0 (no change) on the left and can be increased by moving it to the right.

Three other buttons also appear. *Debug* can be ignored. *Quit* (of course) terminates the program. *Next Level* will bring us to the next level, which is...

Level II

Two sliders, one vertical and one horizontal, appear. Moving the horizontal slider to the right interpolates toward blue and green, to the left toward red and violet. Moving the vertical slider up interpolates toward red and green, down toward violet and blue.

Level III

All the parameters are increased by moving their sliders to the right, with two exceptions. Moving the *Tempo* slider to the right increases the period length (decreasing the tempo). The *Scale* slider has 6 distinct values from left to right -- a major scale, a more consonant major scale, a more consonant minor scale, a minor scale, a pentatonic scale, and a more minor sounding pentatonic scale. Although the *Key* slider ranges from 0 to 127, only the value modulo 12 is significant.

Level IV

The *LAYER* control shows which layer is being modified by the other controls. When placed on the far left, all layers are targeted for modification. One step to the right targets the first layer, two steps to the right targets the second layer, etc. When this slider control is placed on the far right, a new layer is added.

New layers default to being inactive. Move the *Active* control to the right to turn a layer on. The rest of the controls all increase their respective parameters when moved to the right, except the *R. Style* slider which has four values from left to right -- a regular beat, attacks between those of the main rhythm, a periodic pattern, and the main rhythm itself (see 7.2.5).

Table of Contents