KINETIC ENGINE: TOWARD AN INTELLIGENT IMPROVISING INSTRUMENT

Arne Eigenfeldt
Simon Fraser University
School for the
Contemporary Arts
Burnaby, BC, CANADA
arne_e@sfu.ca

ABSTRACT

Kinetic Engine is an improvising instrument that generates complex rhythms. It is an interactive performance system that models a drum ensemble, and is comprised of four multi-agents (Players) that collaborate to create complex rhythms. Each agent performs a role, and decisions are made according to this understanding. Furthermore, the four Players are under the coordination of a software conductor in a hierarchical structure.

1. INTRODUCTION

Winkler notes the relationship between interactive computer music and existing models of musical improvisation, and the fact that many processes in interactive music tend to model activities found in improvisation, namely listening, interpreting, composing, and performing [7]. My own investigation into computer improvisation has disregarded machine listening and interpretation, and instead focused upon real-time composition and performance.

Such performance systems can be considered as complex instruments, in that multiple gestures are generated that proceed and interact in complex ways, yet under the direction of a single performer/operator. In many cases, this performer provides the high-level compositional decisions, allowing the computer to use various methodologies to generate lower level, event-to-event decisions. As the number of possible gestures has increased – partially as the result of technological advances in hardware and software – higher-level decisions are required of the computer in order to control complexity. This suggests the possibility of the computer rising to the level of an intelligent compositional partner, rather than merely a tool. One such design for an intelligent software instrument is outlined in this paper.

1.1. The Use Of Constrained Random Procedures

My personal interactive systems involve generative processes, and are informed by the aesthetics of John Cage, who stated: "I would assume that relations would exist between sounds as they would exist between people and these relationships are more complex than any I would be able to prescribe. So by simply dropping that responsibility of making relationships I don't lose the relationship. I keep the situation in what you might call a natural complexity that can be observed in one way or another" [3]. A good deal of live algorithmic systems employ this model through the use of constrained random procedures, which can be adjusted or altered by performer (the computer operator). As Winkler suggests, this allows for the shaping and influencing of musical output [7]. The use of such random procedures allows music to be unpredictable, yet remain within a defined range that determines a coherent musical style [7]. Truax states that complexity in musical systems is a direct correlation to its unpredictability; in this respect, music that is randomly generated (or influenced) is, by definition, complex [5].

2. THE LIMITS OF RANDOMNESS

The use of constrained random procedures in interactive systems produces music that has a surface complexity that can be argued as being modeled after certain forms of improvised music; however, in such systems, event-to-event decisions are being left to random procedures rather than human "inspiration". As is the case with most improvised music, the notion of form requires large-scale structural change that balances this immediacy. Algorithmic music can only appear intelligent when it exhibits such direction and macro-level control. As Lewis suggests, within his *Voyager* program, "tendencies over a long period of time exhibit consistency, (while) moment-to-moment choices can shift unpredictably" [2].

Unfortunately, constrained randomness does not offer a viable model for mid-level control strategies. In event-to-event decisions, a less-than-satisfying decision will be quickly forgotten due to the amount of events that will be occurring. Given a sixteen note melody, for example, if the melodic shape is interesting, and 75% of the notes are deemed "good" by the listener, the remaining 25%, if only "adequate", will be acceptable, or at least "forgiven". Poor midlevel decisions, such as sectional changes, will have greater, and at least longer lasting, consequences.

3. THE NEED FOR NEW MODELS

Rowe suggests "interactive software simulates intelligent behavior by modeling human hearing, understanding, and response" [4]. In this respect, more consideration must be given to understanding of the totality of the music produced at any given time by the system.

The approach outlined in this paper is based in *Intelligent System Engineering*, without reference to cognitive models. As Widmer and Smaill suggest, "the concern is not to match up closely with human capabilities, but to provide a system capable of

performing a given task, usually inspired by some ideas of how it is done by humans" [6]. This research has been undertaken by a composer searching for new methods of software control in interactive systems. As such, the pursuit of intelligent performance is not abstract, but rather firmly centered within behavior-based AI.

In order to simplify the problem, the initial research described has been limited to rhythmic systems and the interaction of rhythmic layers. Music that relies upon such qualities, particularly in repetitive structures, has been used as loose models, and observations about their behaviors have been made. At this point, these models have been limited to African drum ensembles (specifically the music of the Ewe in Ghana) and Techno, as well as certain elements of pre-1960s jazz.

3.1. Modelling Intelligent Behaviour

Winkler suggests that types of limitations placed upon performance choices determine the style of music [7]. Furthermore, within structured improvising ensembles, players assume certain roles that are often delineated by such limitations. For example, in bebop or related jazz styles, each part (drums, bass, piano, soloist) has a degree of freedom, but must fulfill its role; players are required to work towards a communal goal while listening and responding to one another. The exact pattern played by the drummer, for example, may be unpredictable, but it must be appropriate for the particular moment. Individual choices are limited in varying degrees depending upon the exact style (note choices are more limited in swing music than in bebop, for example), whereas larger formal designs are determined by the song's form.

Techno displays a similar stratification within its layers based upon its parts. The bass drum is expected to provide the simplest pulse, while the snare interacts with the bass creating a homogeneous rhythmic layer. Other parts, often given to other percussion sounds and hihat, exhibit a greater density and complexity. These parts vary more often, and essentially providing foreground material with the greatest degree of change.

Similarly, in the Ewe drum ensemble, specific instruments fulfill specific functions: the support drums of *sogo* and *kidi* provide active, but relatively unvarying, patterns; the double bell *gankogui* and rattle *axatse* provide the unvarying time line; and the master drummer provides the most foreground change through improvisation.

In all three models, different levels of consistency are required within each part. Small variations will occur in the background parts, but a large degree of repetition is required as well. Foreground parts display the greatest degree of variation. In the case of the Ewe music, the master drummer can continually change his pattern, dropping out entirely; it is not until every drum changes its pattern that we perceive a large scale formal variation.

In using these models to derive intelligent behavior, several observations and assumptions can be made:

- Each performer understands the directed goal, and their role in achieving it;
- The parts are not progressing independently[1], but are fully aware of their role in relation to others. A certain degree of interaction is expected and required;
- Different parts react differently to the need for dynamic variation.

4. KINETIC ENGINE

In designing Kinetic Engine, the goal was to create a system that involved minimal performer (user) input - fewer actual controls, each of which controlled extremely high level processes. At the same time, the system had to be evolutionary and generative, one that would not rely upon performer input to afford dynamic change. If left alone, it would begin to generate its own musical variations, both at the lower and higher levels. At any point, however, a performer could nudge the system, or push it dramatically.

Kinetic Engine was written in Max/MSP [1].

4.1. Players

The basis of Kinetic Engine is a *Player*, a software agent defined in terms of its behavior. Each of the four Players perform one rhythmical pattern, and has the ability to vary its performance details (panning, dynamics, processing) in a constrained random fashion within predetermined ranges.

On its own, the Player has no intelligence, nor does it have an ability to listen or react to the other Players – interaction is controlled in a top-down fashion through the use of a virtual conductor. The Player's performance variations are only meant to maintain interest for several measures at a time; as such, the Player can be viewed as a one-voice drum machine with random parameter variation.

4.2. Type

Each Player is given a role within the overall music, which establishes its *Type*. The Type determines how the Player reacts to control information (i.e. when to vary its parameters, when to vary its pattern), its density, and its pattern construction[2]. Type loosely parallels the division of roles in African music (the support drums of *sogo*, *kidi*; the double bell *gankogui*; the rattle *axatse*) and/or in techno (bass drum, snare, other percussion, cymbal/hihat).

4.3. Pattern Generation

Patterns are constructed using knowledge based algorithms, and are correlated between Players – specifically, Players One and Two form a pair, as do

[1] A comparison can be made to collective goals of heterophonic music as opposed to the independence of polyphony.

[2] Since there are four Players, and four Types, Player numbers also correspond to Type; i.e. Player One is Type One, Player Two is Type Two, etc. The Player number is simply a number, a unique identifier: it is its Type that defines how it operates. For purposes of clarity in this paper, when I refer to "Player One", for example, I am also referring to "Player One of Type One".

Players Three and Four. Players One and Three are freely generated, and Players Two and Four are subsequently correlated to their partner patterns.

The number of notes in a given Player's pattern is dependent upon the overall *Density* parameter (see 4.4.1). From this value, the number of notes per Player is determined. These notes are then distributed amongst the number of beats in the measure (variable between two to eight, and not necessarily consistent between Players). Finally, given the number of notes per beat, a specific pattern is chosen from available patterns. For example, given a subdivision of four (sixteenths) and two notes required in a beat, only six possible permutations are available. Certain permutations are weighted more heavily than others for specific Players.

In correlated generation, once the number of notes per beat has been determined, possible permutations are tested against the partner pattern. Those permutations whose notes fall on the reference pattern's rests are scored higher, resulting in a rhythmic hocketing. If multiple patterns score equally, a random choice from these results is made. If no exact match results (for example, if three notes are required and two notes exist in the partner's reference beat), fuzzy logic is used to determine the best match.

The complete methodology involved in pattern generation is complex, and cannot be discussed in detail in this paper.

4.4. Performance Controls: Density and Variation

The two performance controls are limited to overall *Density* and *Variation Amount*.

4.4.1. Density

The Density parameter determines the cumulative number of notes of all Players. The distribution of notes across Players is weighted – lower Player's notes are weighted more heavily than higher Player's notes. Therefore, the same number of notes dispersed across the different Players (i.e. 3 4 5 6 versus 6 5 4 3) does not constitute equality in distribution. The musical result of this weighting is that lower Players have fewer notes than higher Players.

Density change is fundamental to Kinetic Engine. An initial Density is set, and patterns are generated for each Player. When Density is changed (as a parameter of Variation), each Player's pattern is also varied (by adding or subtracting notes), or regenerated. Higher numbered Players will react first to density change requirements, resulting in variations in density occurring mainly in the more foreground layers.

Changes in density do not occur instantaneously; instead, an initial attempt is made to match the new Density at the beginning of each measure. The resulting Density is then compared to the requested Density on each subsequent beat, and an adjustment is made if required - this process can go on for several beats. Due to the independence of each Player's adjustments, density changes are unpredictable, paralleling Lewis' goal of avoiding "the kind of uniformity where the same kind of input routinely leads to the same result" [2].

Density change is considered a mid-level variation (see 4.4.2) and can occur in reaction to the need for variation. In such cases, a much lower Density is set, and a new target density evolves over several measures (the duration of which is a constrained random variable within a given section), and the Density slowly increases to this new level. Since the Density level is dynamic, each Player's pattern is adjusted to match this changing variable, resulting in a complex interaction between Players.

4.4.2. Variation

The need for variation is assumed to maintain interest. Small variations may hold the listeners attention for a few measures, but then greater and greater variations are required in order to maintain this interest. Thus, a *Variation Meter*, which determines how much variation is to be applied, gradually increases over time, literally demanding more and more change. At recurring time intervals – such as every one to eight measures, determined by a variable for the section – this meter is tested, and assorted variations are initiated. Each variation has a scaled value, depending upon its perceived interest. For example, tuning range (a variable range from which a Player randomly chooses a detuning), processing range (a variable range in which the cutoff frequency drifts for a Player's filter), and delay line range (a variable range for a Player's delay amount or feedback) are considered small variations, while pattern variation through Density change is considered a medium variation, and timbre/instrument changes are considered large variations. Because the amount of variation is dependent upon the Variation Meter, only small variations are called for when the meter is low, whereas larger variations are invoked the higher the meter gets.

At the test interval, the total amount of *actual* variation is summed. Because of the use of random procedures and a complex weighting system for variation initiation, the amount "requested" by the Variation Meter might not actually occur. The sum total of actual variations is then subtracted from the Variation Meter, causing it to be lowered. Although this sum rarely equals the required amount, the Variation Meter should return to zero or relatively close to it. If the Variation Meter has not returned to zero, the cumulative result is a gradual rise in the meter, which is offset by larger variations being initiated, which, in theory, should bring the meter down again.

Small variations are not considered to maintain interest for very long. Thus, the summed variation amount begins to be scaled over time; in other words, it loses its effect, and the Variation Meter begins to increase. This increase then requires larger variations to occur in order to keep it down; however, they, too, will lose their effect, and the meter will once again rise. Finally, a major variation (Density change coupled with new instrument selection) will start the process all over again.

4.5. Sections: large scale changes

The musical result of increasing variations is a gradual increase in complexity, culminating in a major change. The above-described outline generates complex interactions that maintain interest for long periods of time; however, large scale formal change does not occur since there is an overall consistency within a given section.

For this reason, the notion of Sections were created. Within a given Section, a range of probabilities is predetermined: the group of instrumental sounds; the probability of particular processing; the range of densities and the range of time it takes to move between them; how often variation occurs; how quickly the Variation Meter progresses, and the effectiveness of the variations on keeping it down; and the maximum number of Players active at once. Of note is the fact that each of these probabilities is dynamic throughout the Section; for example, the Section may begin with only two maximum Players, but progress to four during the course of the section.

4.6. Performance versus Autonomous Mode

Kinetic Engine can operate in Performance mode, in which a performer can control the Variation and Density Meters to initiate immediate large-scale changes. Once the performance input ceases, the program returns to Autonomous mode. Performance mode also allows for section changes to be initiated, or blocked.

Lastly, Performance mode allows for a "freeze" command, essentially a temporary off switch for the Variation Meter. This allows the performer to pause the evolution of the system if a particularly interesting rhythmic moment has evolved.

5. FUTURE DIRECTIONS

At present, Kinetic Engine is a top-down hierarchical system. This is more of an aesthetic decision on my part, rather than a functional one, since I am a composer/conductor first, rather than a performer. Perhaps this system represents my ideal musical situation, in which creative performers slavishly follow a composer/conductor's every whim.

Kinetic Engine could be rewritten so that each Player is an autonomous agent, and decisions are made at the Player level. This would no doubt lead to greater independence, and perhaps more closely model a human improvising ensemble. However, it would also create very different music; for example, it would then fall into the improvised music trap of an inability to quickly and dramatically change musical direction.

One limitation of Kinetic Engine is a lack of understanding of its timbral spectrum. The choice of instruments for each Player is currently based upon predetermined groupings. This guarantees a certain consistency, but also limits potential combinations. Greater freedom of instrumental choice could involve an analysis of currently playing instruments using FFTs, thereby determining the existing spectrum, and deciding which available instruments could best complement this spectrum.

Lastly, Kinetic Engine lacks any ability to learn from the music that it creates. In comparing Kinetic Engine to other behavior-based AI programs – such as those involving populations of robots – the difficulty of evaluating its own success is apparent. Most learning-based systems require this evaluation and reinforcement for future applications; in other words, if the system did something well, it should continue to do that in the future. Success, in the case of Kinetic Engine, is the creation of interesting music. Since there is no algorithm that can make such a determination, human evaluation is required. Such training is possible, potentially involving the use of neural nets and/or genetic algorithms.

Music generated by Kinetic Engine can be found here:

http://www.sfu.ca/~eigenfel/research.html

6. ACKNOWLEDGEMENTS

I would like to thank Dr. Richard Vaughan of Simon Fraser University's Autonomy Lab and Computing Science Department, for help with concepts related to Artificial Intelligence, and advise on system design in both current and future implementations.

7. REFERENCES

[1] Dobrian, C. *MSP, The Documentation*, Cycling74, 1998.

[2] Lewis, George E. "Too Many Notes: Computers, Complexity and Culture in Voyager", *Leonardo Music Journal*, 10, (2000), p. 33-39.

[3] Nyman, Michael *Experimental Music: Cage and Beyond*, New York: Schirmer Books, 1974, p. 25.

[4] Rowe, Robert. *Interactive Music Systems*, Cambridge, Mass., MIT Press, 1992.

[5] Truax, B. "The inner and outer complexity of music", *Perspectives of New Music*, 32 1, (1994), p. 176-193.

[6] Widmer, G. and Smaill, A "Musical Knowledge: What can Artificial Intelligence bring to the Musician?", *Readings in Music and Artificial Intelligence*, Eduardo Reck Miranda, ed., 2000, p.29-46.

[7] Winkler, Todd. "Strategies for Interaction: Computer Music, Performance, and Multimedia" *Proceedings of the 1995 Connecticut College Symposium on Arts and Technology*, 1995.