The Virtual Primitive Drum: a Multi-agent Real-time System to Create Complex Ensemble Rhythms

Arne Eigenfeldt
Simon Fraser University
School for the Contemporary Arts
Burnaby, CANADA

Introduction

I have always found inspiration in the music of other cultures; however, not as a performer playing within them, nor as a musicologist trying to understand them, but as a composer looking for new models.

Steve Reich's comment on how non-Western music can influence a composer has resonated within me:

> "one can create a music...that is constructed in the light of one's knowledge of non-Western structures...This brings about the interesting situation of the non-Western influence that is there in the thinking, but not in the sound...Instead of imitation, the influence of some non-Western musical structures on the thinking of a Western composer is likely to produce something genuinely new." (Reich, Postscript to a Brief Study of Balinese and African Music, (1973).

Improvisation vs. composition

It is probably not a coincidence that I have also been drawn to the mercurial element of improvisation and real-time composition. Unlike the EuroAmerican modernist tradition of a fixed-time object, in which the unvarying score is the goal, non-Western music tends to be more improvisatory, allowing for significant performer input into the final product. Some of my own most rewarding musical experiences resulted when performers were involved in the realization of my compositions in very active ways: each musician contributed their own musicality to create a collective whole whose details were not directly specified, nor predicted, by me.

However, my training as a composer has suggested that organization and complexity are musical aspects that I also want to pursue, a desire seemingly at odds with the spontaneity of improvisation.

Interactive computer music

Consequently, I have been drawn to interactive computer music, in which the computer actively plays a part in the compositional process, making decisions during the performance. This type of music has been called *interactive composing* by Joel Chadabe, in which the performer (myself) is influencing the system, and the outcome of the system is influencing the decisions that I make in turn. Some contemporary laptop music can be considered interactive composing, but only those in which the computer makes active decisions towards the musical results.

If the decisions the computer makes could be made to resemble my own, ones that I *might* have made, but not necessarily *would* have made, then there is the potential for a compositional partner that creates music consistent with my goals at a high level, but spontaneous and unpredictable in their low-level detail.

Constrained Randomness

A traditional method of restricting the computer's choices within interactive music is through the use of constrained randomness: the composer limits choices to a range of values for the various musical parameters, and the computer can freely choose from those values.

For example, limiting pitch choices to a certain scale, and rhythms to durations that guarantee a perceivable pulse. Todd Winkler noted that using such procedures allow for the shaping and influencing of musical output, resulting in music that is unpredictable, yet remaining within a defined range that determines a coherent musical style. This is an example of such a program, created in Max/MSP, in which rhythms are randomly generated and varied. One can consider this an algorithmic 2 voice drum machine, in which rules determine how patterns are made, and varied.

EXAMPLE: Beat Gestures movie

In this example, what becomes apparent after 30-40 seconds is the need for higher level variation (as opposed to simple re-ordering): although the music can be argued to be *complex*, in that the relationship between the two parts cannot be predicted, it also becomes monotonous due to this consistent unpredictability. There is no musical direction, no evolution within the music.

Typically, in music such as this, at this stage one can do the following:

1. add more parts (which was already done)
2. add more variations to the existing parts (i.e. volume, timbral processing, etc.)
3. develop the parts musically

In most cases, the first choice is the easiest (and the most often heard), while the last is the most difficult, and rarely heard using algorithmic means.

Natural Rhythm

Compare that computer generated example with these recordings of rhythmic invention from around the world:

EXAMPLES

- **Caribbean Meringue**
- **Cuban Inle**
- **Ewe Atsiagbeko**
- **Kenyan Hunting Song**

All of these examples have several elements in common:

- the use of repetitive structures;
- the use of rhythmic polyphony, as opposed to more soloistic rhythm found in India, for example;
- complexity achieved through the integration of simple parts rather than any one virtuoso part;
- differences between fore- and background parts primarily dependent upon the level of variation: the greater and more frequent the variation, the more foreground the part;
- the elusive element of "groove", seemingly a combination of predictable rhythm (emphasis of certain key beats) with unpredictable elements (slight variations in timing, amplitude, pattern).

Is it possible to create a system that can generate this type of music, without necessarily imitating its details (as Reich would suggest)? Can the nature of this music be recreated without reference to its specific patterns (or more precisely, without the use of a stored database of rhythms)?

Certainly not by methods of constrained randomness, since there is nothing random here; instead, the software will need to act more human, to make musical decisions that resemble that of a musician.

Kinetic Engine

Kinetic Engine is an interactive real-time software program which has been under development since 2005; it is an initial effort (in a very long term project) to encode musical knowledge within software.

In this first stage, an effort was made to limit the intricacy of this problem, and it has been limited to polyphonic rhythm generation. It addresses what I find are the shortcomings of much laptop music, in that individual parts within the music are aware of one another, and interact and develop appropriately and musically.

My interest in process music has led to a reluctance to specify copious amounts of data beforehand. Perhaps this goes back to Reich's notion of imitation vs. structural models, but I would much rather specify how to generate a rhythm, and have the software do that, rather then enter any specific rhythms, for example. For this reason, Kinetic Engine is a rule-based, rather than database driven, system; while I look to the music of Africa and South/Central America for inspiration, I make no attempt to recreate these styles accurately.

Similarly, none of this research is based upon cognitive models: I make no claims that humans actually think or act in this manner. I am interested in producing software whose *outcome* resembles intelligent decision making. As such, my research can be considered behaviour-based AI.

High level control

One of the first design factors was to create software that could make high level decisions on its own - if I stopped controlling it, the software would take over and begin making these same decisions (thus the system's name - it should potentially generate endless variations). Any control I did exhibit would be at a very high level - similar to a conductor controlling an ensemble using only hand gestures. The only control I have is *density* - how many notes are playing at any given time. Although I can *influence* the system, all other details are left to the software.

Intelligent Agents

I suggested earlier the idea of a conductor and ensemble - this is, in fact, the model in *Kinetic Engine* : independent virtual players that contain musical knowledge, who interact with one another in musically intelligent ways. The software is thus based upon methods of *agency*.

The promise of agent-based composition in musical real-time interactive systems has already been suggested by Murray-Rust, Wulfhorst et.al, Miranda, and others. One reason for this interest is the potential of agent-based systems to emulate human-performer interactions. Agents have been defined as autonomous, social, reactive, and proactive, similar attributes required of performers in improvisation ensembles.

Overview of agents in *Kinetic Engine*

In the case of *Kinetic Engine,* agents determine independently how and when to react to the current context, determining their own actions in relation not only to their internal states, but to those of other agents. Their actions (and reactions) are dependent upon "personalities" of the agent.

For example, whether the agent will tend to syncopate; how responsive it is to global change; how willing it is to upset a stable system.
Agents communicate portions of their current state to the other agents, which can inform the other agent's decisions. This reflects the notion that musicians may not know each others personalities, but they can *hear* what each other are playing - similarly, agents become aware of the *results* of an agent's decisions, but not the *circumstances* (i.e. parameters) that led to those decisions. The agents exhibit "social" behaviors by forming connections with other agents via messaging, similar to how musicians interact, taking visual cues from one another. When agents interact "socially", they alter their patterns to form rhythmic polyphony (interlocking patterns) or heterophony (similar rhythms).

Agents classes

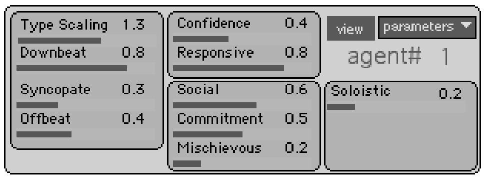
Kinetic Engine is comprised of two agent classes: a conductor and an indefinite number of players.

The Conductor Agent

The conductor agent handles incoming sensor information (for example, performer control over density), as well as managing high-level organization: tempo, metre, subdivision, and probabilities for timbral processing. The conductor also sends a global pulse, to which all player agents synchronize. In our model, imagine the conductor as the person standing in front of our ensemble, snapping off the tempo.

The Player Agents

Upon initialization, agents read their stored personality traits from disk, which determine how they will react to their environment.



Type Scaling 1.3
Downbeat 0.8
Syncopate 0.3
Offbeat 0.4
Confidence 0.4
Responsive 0.8
agent# 1
Social 0.6
Commitment 0.5
Mischievous 0.2
Soloistic 0.2

Some of these parameters include
• *Confidence* (number of notes with which to enter),
• *Responsiveness* (how responsive an agent is to global parameter changes),
• *Social* (how willing an agent is to interact with other agents),
• *Mischievous* (how willing an agent is to upset a stable system).

Agents assume one of three *types*, which determines certain aspects of their behaviours, including timbre, density, and variation:

Type	Timbre	Density	Variation
low	low frequency (bass drums)	below average	less often
mid	mid-range frequency (most drums)	average	average
high	high frequency (cymbals, shakers, bells)	above average	more often

An agent's *type* is made somewhat less restrictive through its personality trait of *Type-scaling*. For example, *low* agents will tend to have lower densities than other types, but a *low* agent with a high type-scaling will have higher than usual density for its type.

Agent activation

A performance begins once the conductor starts “beating time” by sending out pulses on each beat. Agents, however, do not react immediately, nor synchronously; instead, they react “every few beats” - a variable number called a *check-beat* - which based upon an independent fuzzy logic counter that makes them “wait a bit”, taking into account the agent’s *responsiveness*. Once a check-beat occurs, the agent tests its *responsiveness* parameter (by generating a random number and comparing it to this parameter): less responsive agents will take longer to react to the conductor's demands.

Fuzzy Logic Ratings

Kinetic Engine attempts to model human approximation through the extensive use of fuzzy logic to judge success. For example, in the case of density, agents are unaware of the exact global density required. Instead, the conductor rates the required density as “very low”, “low”, “medium”, or “high”, and broadcasts this rating.

Agents know the average number of notes in a pattern based upon this rating, and apply various scaling properties based upon personality parameters to generate an actual density.

	Density
Potential notes: *16* (metre = 4, subdivision = 4)	
Fuzzy density: *medium*	7*
Type : *low*	4.91**
Type scaling: *1.1*	5.16
Confidence: 0.52	2.68
Gaussian curve	4***

*This value is an example of “hand-optimized” data that is based upon *listening* to the resulting music, rather than an analyzed database, for example.

**floating point is maintained until the last calculation.

***rounded to nearest integer

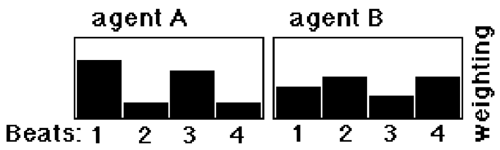
The conductor collects all agent densities, and determines whether the accumulated densities are “way too low/high”, “too low/high”, or “close enough” in comparison to the required global density, and broadcasts this success rating. Agents then react to the updated requirements.

For example, if the accumulated density is “way too low”, a non-active agent can activate itself. If the accumulated density is simply “too low”, an active agent can increase its density and add more notes. If the accumulated density is judged to be “close enough”, the overall density is considered *stable*, and social interactions can begin.

Generating Rhythms

Density Spread

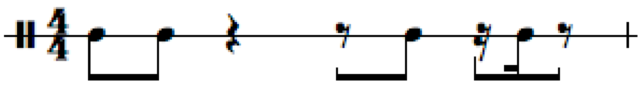
An agent’s density is spread across the available beats using fuzzy logic to determine probabilities, influenced by the agent’s *downbeat* and *offbeat* parameters.



Continuing our example above, agent A’s *density* of 4 might end up with a *density spread* of 2 0 1 1, whereas agent B’s *density* of 4 might end up with a *density spread* of 1 2 0 1.

Agents determine the placement of the notes within the beat using a similar technique, but influenced by the agent’s *syncopation* parameter. Thus, a density spread of 2 0 1 1 requires 2 notes in beat one, and 1 note in beats three and four.

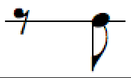
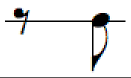
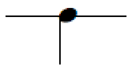
Given a *syncopation* parameter of .25, the resulting rhythm might result:



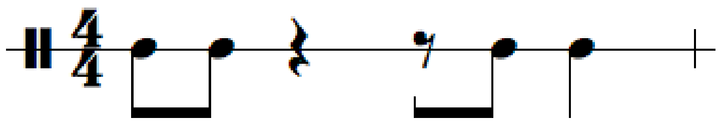
(The syncopation at the subdivision level has only occurred in beat four)

Pattern Checking

After an initial placement of notes within a pattern has been accomplished, *pattern checking* commences within each agent. Each beat is evaluated against its predecessor and compared to a set of rules in order to *avoid* certain patterns and *encourage* others.

Previous beat	Pattern A	Pattern B
		
Percentile chance of alteration	.3	.9

Thus, in our ongoing example, the third beat *did* contain the rhythm in fig.3, and the fourth beat did require a single note, therefore this rule would be initiated for the fourth beat. The test is made for pattern A, it fails (a random value above .3 was generated). The test for pattern B is made, it passes (a random value below .9 was generated), therefore the rhythm in beat four is changed.



Note that if the second test failed as well, the original rhythm would be allowed to remain.

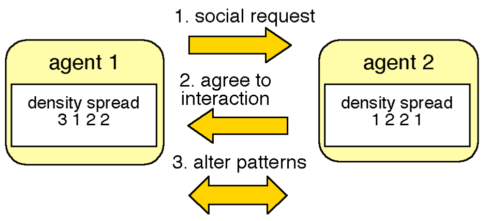
Social Behaviour

Once the density is stable among active agents and all patterns are generated, agents begin social interactions. Agents evaluate other agent's *density spreads*, testing for similarity and dissimilarity. Evaluation methods include comparing density spread averages and weighted means, both of which are fuzzy logic tests.

Agent #	**1**	**2**	**3**
Density Spread	3 1 2 2	1 2 2 1	2 3 3 3
Similarity rating		**0.53**	0.48
Dissimilarity rating		0.42	0.33

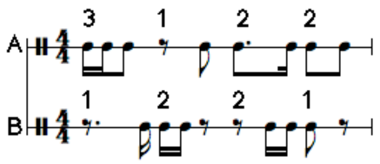
Agents message the agent with the highest evaluation; the *type* of evaluation (similar or dissimilar) will determine the type of interaction. Thus, in the above example, Agent 1 has evaluated two other agent's density spreads; the *similarity* to agent 2 is highest. Therefore, Agent 1 will message Agent 2 in an attempt to begin social interaction.

Before this can begin, however, Agent 2 must acknowledge the message. This is an attempt to model eye contact between musicians: if "eye contact" is not made, no relationship can be formed, and the agent will begin to look elsewhere for social interaction



Interaction types - Polyphony (interlocking)

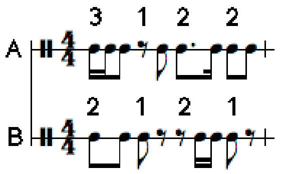
Two types of interaction are possible: polyphonic (interlocking) and heterophonic (similar). In polyphonic interaction, agents attempt to “avoid” partner notes, both at the beat and pattern level. For example, given a density spread of (3 1 2 2) and a partner spread of (1 2 2 1), both agents would attempt to move their notes to where their partner’s rests occur.



Note that not all notes need to successfully avoid one another (beats 3 and 4). Also note that this is a continual and mutual process: both agents have the potential to alter their patterns; for example, agent B may also alter its pattern based upon that of agent A - on the next check-beat, this could result in *another* alteration of patterns to match the new patterns.

Interaction types - Heterophony (expansion)

In heterophonic interaction, agents alter their own density spread to more closely resemble that of their partner, but no attempt to made to match the actual note patterns.



In this example, Agent B had an initial spread of (1 2 2 1) - note that it rearranged this spread to more closely resemble that of Agent A.

Learning and evolution of agent personality

I have recently begun to explore how agents can learn, adapt and evolve their personalities over several performances, and within the performances themselves. When *Kinetic Engine* begins to play, this is considered a new *composition*; because the tempo, metre, and subdivision remain constant within the composition, there is an obvious consistency within this formal unit. New compositions can be triggered by the user, causing new musical parameters to be generated; several such compositions would result in a *performance*.

When an agent is initialized at the beginning of a performance, it reads its individual personality parameters from disk. Between compositions, agents temporarily alter these personality parameters, based upon an evaluation of previous compositions within the current performance. For example, if an agent were particularly active in one composition, it would likely decide to "take a rest" for the next composition by lowering its *Responsiveness* parameter. Similarly, if its average density were below its preferred density (based upon its *type*, *type scaling*, and *confidence* parameters), it would raise its

Confidence parameter for the subsequent composition(s).

After each composition and over the course of the performance, the agent keeps track of its actual behaviour in relation to its preferred behaviour, and continually makes adjustments in an attempt to match the latter; in other words, it tries to “average out” to its stored personality parameters. In certain cases, agents also adjust their behaviours in relation to other agents.

For example, the *Downbeat* parameter determines how likely an agent will place a note on the downbeat: this is particularly important for type *low* agents (since bass drummers tend to emphasize the downbeat). All *low* agents will compare their own patterns to those of the other *low* agents: if their own downbeat percentile (the number of times their pattern contained a downbeat) is above the average of other *low* agents, that agent would feel the responsibility to maintain a higher percentile of downbeats, and thus keep a high *Downbeat* parameter.

At the end of a performance, the user can decide whether the agents should "learn" from this experience, and store their evolved personality parameters to disk, to be subsequently used for the next performance. In fact, what is stored is not the most recent parameter, but a comparison of the difference between the original parameter and the accumulated history of the past performance. An exponential probability curve is created between these two values, and a new personality parameter is generated - this new value will be close to the original, but altered by the previous performance. The result is that agents begin to evolve not only in relation to how they performed, but also in relation to the *ensemble* that they performed within. The particular group of *type low* agents, for example, will begin to evolve a certain working relationship as they develop certain roles amongst themselves.

Thus, agents do not begin each performance *tabla rasa*, nor through randomized parameters, nor via user specification. Instead, they act as human performers might act, evolving performance tendencies based upon interactions learned from previous performances. What has not yet been tested is how agents react when taken out of their "rehearsed" ensemble, and thrown together in a "pick-up" group comprised of agents from other ensembles.

(Immediate) Future Directions

Kinetic Engine is an ongoing research project: it is literally changing every week; therefore, as the software evolves, many new potential directions emerge. One initial limitation to the software was its restriction to rhythm, and my goal was to move to pitch and timbral organization as soon as possible. However, I have become more and more interested in the evolution of the independent agents and their musical decision-making potential: the placement of "melodic/harmonic agents" on the "To Do" list seems to always get pushed down near the bottom.

Here are some issues that will be explored in the next few months:

Soloists vs. ensemble

At the moment, every agent is part of the ensemble, fulfilling a background role. Agents do evaluate their own success: agents that are particularly active in terms of pattern variation will increase their own volume in an effort to “feature” themselves. The next step would be for the agents to decide to "take a solo", and break apart from the group dynamic through greatly increased variation and decreased repetition. All the agents would need to acknowledge the soloist, by "voting" upon which agent is deemed to be the most interesting. Once an agent receives enough votes, it can become a soloist. This also raises the potential of multiple soloists, or a "quarrel mode", in which two or more agents compete for soloist rights.

Attractors, Scored Data, and Human Interaction

When agents become social, they determine which agents to interact with based upon similarity of density and type: *high* agents will more likely interact with other *high* agents, for example. The next step would be to incorporate an *Attraction* parameter, in which agents have an evolving parameter that determines how likely other agents will interact with it. Agents with high Attraction parameters will have the potential to "lead" the ensemble (perhaps as soloists).

This would allow the concept of predefined data, or scored patterns, to be used. Giving an agent a particular pattern to play, and making that agent highly attractive, would cause other agents to interact with scored material in evolutionary ways. This particular "scored" agent may, or may not, actually play: thus, a predefined structural *tala* could exist within such an agent, without the actual performance of such a defining rhythm.

Lastly, the attractor agent could receive its data from a live performer - the predefined data with which other agents interact could be continually changing.