DRUM CIRCLE: INTELLIGENT AGENTS IN MAX/MSP

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

ABSTRACT

The author describes a recent work created in Max/MSP, which explores the use of multi-agents over a local area network. Individual agents emulate improvising percussionists in a drum ensemble to create intelligent, evolving rhythms.

1. INTRODUCTION

Drum Circle is an installation/performance using the composer's software *Kinetic Engine* [1], in which intelligent agents created in Max/MSP emulate improvising percussionists in a drum ensemble. Intelligent agents are elements of code (programs/patches) that operate without direct user interaction (autonomous), interact with one another (social), interact with their environment (reactive), and make decisions as to when they should operate, and what they should do (proactive)[6]. Since these are also attributes required of musicians in improvisational settings, the use of agents to emulate human-performer interaction has proven to be a fertile field of research [7, 5].

Kinetic Engine arose out of a desire to move away from constrained random choices and utilize more musically intelligent decision-making within real-time interactive software. In order to retain high-level performance control, the principle control parameter is limited to *density*: how many notes are being played by all agents. All other decisions - when to play, what rhythms to play in response to the global density, how to interact with other agents – is left to the individual agents.

This paper will describe the design and operation of *Kinetic Engine* within the work *Drum Circle*. Section 2 gives an overview of *Kinetic Engine*, describing agent types and aspects of fuzzy logic used in the system. Section 3 describes how agents generate rhythmic patterns. Section 4 describes how agents interact socially to adjust their patterns in response to other agents. Section 5 describes how messaging between agents operates. Section 6 contains conclusions and future directions. A more detailed comparison of *Kinetic Engine* and other systems is given in [3].

2. OVERVIEW

Drum Circle was premiered with one central computer operating as a conductor agent, and nine networked computers operating as player agents: there can be an indefinite number of player agents. The author used a data-glove to control density (see figure 14) and other parameters (see next paragraph).

Kinetic Engine is, essentially, generative, rather than interactive; the only *effective* user control is density: the cumulative number of notes all agents play. An earlier version of the software was premiered as an installation at ICMC New Orleans. The potential to interact with a performer is currently being investigated (see Section 6).

2.1. The Conductor Agent

The conductor sets global variables, such as tempo, metre, and subdivision. The conductor also handles incoming sensor information, which is mapped to high level parameters, specifically density, system responsiveness, global controls for influencing personality attributes, as well as triggering new compositions. The conductor agent also sends a global pulse (Max's "bang" messages) using the java object net.maxhole, to which all player agents synchronize.

2.2. Player Agents

Player agents are instances of a single patcher running on separate machines. Upon initialization, agents "report in" to the conductor with their instance number, and are assigned a unique ID, which is stored in the agent as a local value (see figure 1).

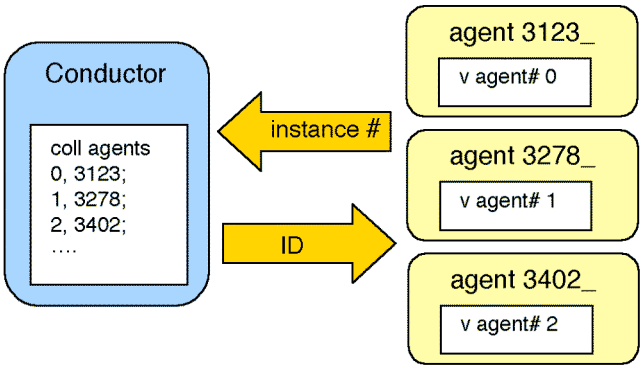


Figure 1. Initialization of agents.

The conductor also counts the number of agents online, and sends this information out: agents adjust their internal data arrays accordingly.

Upon initialization, player agents read from disk their *type,* which determines several aspects of the agent's behaviour (see Table 1).

Also from disk, agents read their personality traits which further determines how they will react in certain contexts (see Figure 2 for the ten personality parameters). For example, an agent with high *Confidence* will enter with more notes; an agent with high *Responsiveness* will react faster to global changes.

	Type 1	Type 2	Type 3
Timbre	low frequency: • bass drums	midrange frequency: • most drums	high frequency: • rattles, • shakers, • cymbals
Density	lower than average		higher than average
Variation	less often		more often

Table 1. Agent *types* and how they influence agent behaviour.

Figure 2. Example personality parameters for a player agent.

2.3. Fuzzy Counters

Kinetic Engine makes liberal use of *fuzzy counters*[4], which allow for the concept of "wait a bit". For example, although a performance begins once the conductor starts "beating time" by sending out bangs, agents do not respond immediately, nor synchronously. Agents react every few beats – a *checkbeat* - using such a fuzzy counter: each beat is tested[1], and agents wait on average between 3 and 6 beats before passing a checkbeat. This amount is scaled by the agent's responsiveness parameter, as well as the overall system responsiveness; less responsive agents will take longer to react to the conductor's demands (see Figure 3 to see how probabilities increase with each test).

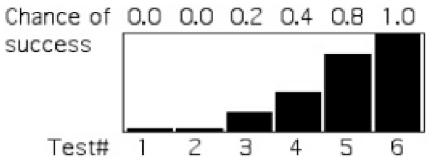


Figure 3. Using fuzzy logic to "wait a bit" by controlling chance of success for each test.

The system responsiveness parameter scales the test number, whereas the agent responsiveness parameter scales the chance of success. The former thus controls how quickly success is possible - allowing for all agents to react immediately - whereas the latter controls how soon success is reached after the initial wait of three beats.

Once a checkbeat is passed, an agent decides whether to activate itself by testing its responsiveness parameter. When an agent becomes active, it determines its density.

[1] A random value between 0.0 and 1.0 is generated, and compared to the parameter in a Boolean test.

2.4. Fuzzy Logic Ratings

Kinetic Engine attempts to model human approximation through the use of fuzzy logic to judge success. In the case of density, agents are unaware of the exact global density required. Instead, the conductor rates the global 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: this value is scaled by the agent's type and type-scaling parameter. Agents generate individual densities after applying a Gaussian-type curve to this number (see Figure 4 for the Gaussian curve in Max's **table** object), and broadcast this density.

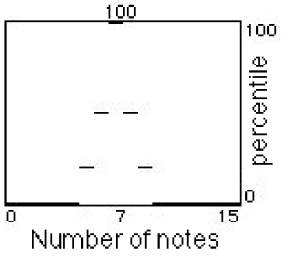


Figure 4. A Gaussian curve, allowing for a random variation around a given value.

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 global density, and broadcasts this success rating.

- if the accumulated density is "way too low", non-active agents can activate themselves and generate new densities (or conversely, active agents can deactivate if the density is "way to high").
- if the accumulated density is "too low", active agents can add notes (or subtract them if the density is "too high").
- if the accumulated density is judged to be "close enough", agent densities are considered stable.

3. PATTERN GENERATION

3.1. 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 (see Figure 5).

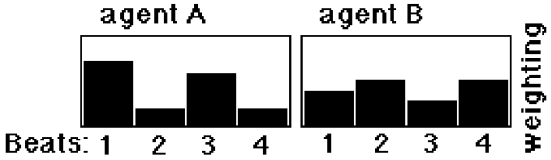


Figure 5. Example density spread weightings for two agents, 4/4 time with different downbeat and offbeat parameter values.

Once the number of notes per beat has been decided, agents determine the placement of the notes within the beat using a similar technique, but influenced by the agent's syncopation parameter.

Notes are represented using simple binary values over a grid of potential notes within a measure, given the time signature and subdivision. Thus, in 2/4 time with a subdivision of 4, (0 0 0 0 0 0 0 0) represents an empty measure, whereas (1 0 0 0 1 0 0 0) represents a quarter note on each beat (durations are not represented).

3.2. Pattern Checking

After an initial placement of notes within a pattern has been accomplished, *pattern checking* commences. Each beat is evaluated against its predecessor and compared to a set of rules in order to avoid certain patterns and encourage others (see Figure 6).

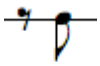
Previous beat	Pattern A	Pattern B
		
Probability	30%	90%

Figure 6. Example pattern check: given a previous beat's rhythm, with one note required for the current beat, two "preferred" patterns for the current beat.

In the above example, pattern A is tested first: there is a .3 percentile chance that this pattern will result. Failing that, pattern B is tested, and there is then a .9 percentile chance that this pattern will result. If this last test fails, the original rhythm is allowed to remain.

4. SOCIAL BEHAVIOUR

Once the density is stable among active agents, they begin social interactions, based upon their social parameter. Interactions involve potentially endless alteriation of agent patterns in relation to other agents; these interactions continue as long as the agents have a *social bond*, which is broken when testing an agent's social commitment parameter fails[2].

Agents evaluate[3] other agent's density spreads, testing for similarity and dissimilarity (see Table 2).

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

Table 2. Example density spreads: comparing agent 1 with agents 2 and 3. Similarity to agent 2 is highest.

Agents message the agent with the highest evaluation; the type of evaluation (similar or dissimilar) will determine the type of interaction (see Section 5.1). See Figure 7 for messaging protocol.

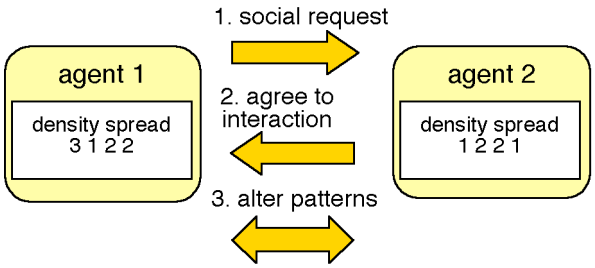


Figure 7. Social messaging between agents

4.1. Interaction types

Two types of interaction are possible between agent patterns: polyphonic or heterophonic.

4.1.1. Polyphony (interlocking)

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[4] (see Figure 8).

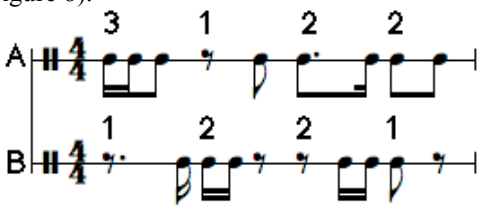


Figure 8. Example polyphonic interaction between agents A and B, with density spreads of (3 1 2 2) and (1 2 2 1). Note that not all notes need to successfully avoid one another (beats 3 and 4).

4.1.2. 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 (see Figure 9).

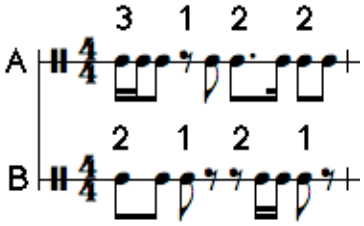


Figure 9. Example heterophonic interaction between agents A and B, with density spreads of (3 1 2 2) and (2 1 2 1). Agent B had an initial spread of (1 2 2 1).

[2] This test is done every "once in a while", another fuzzy counter.

[3] Evaluation methods include comparing density spread averages and weighted means, both of which are fuzzy tests.

[4] Because both agents are continually adjusting their patterns, stability is actually difficult to achieve.

5. MESSAGING

Through the use of instances, agents hide their data within local values and colls, two of Max's data objects (see Figure 10).

Figure 10. The value object as a local variable.

Certain variables are required to be global: they can be accessed by any agent, but are only altered by the conductor agent (see Figure 11). When the conductor alters a global variable, it broadcasts this to the network, and agents update their internal values.

Figure 11. A global variable.

Data that is shared between agents – i.e. an agent's note density – is stored as an array within every agent. Each time an agent alters its own internal value, it broadcasts it to the network (see Figure 12).

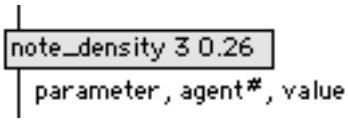


Figure 12. Broadcasting new values to the network.

Agents receive the new value(s), and store them in their own arrays, using the agent number as an index.

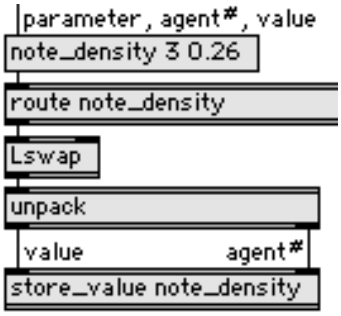


Figure 13. Storing data from other agents.

Example data-handling abstractions, such as "store_value" in Figure 13, are given in [2].

6. CONCLUSION AND FUTURE WORK

This paper presented methods of creating networked multi-agents within Max/MSP, specifically the autonomous, proactive, reactive, and social nature of these agents. *Drum Circle* demonstrates the potential for Max/MSP to create music that explores "groove-based" rhythm through such complex methods, music that can be described as displaying emergent properties.

There are several planned strategies for improving the machine musicianship of *Kinetic Engine*, including the ability for agents to become soloists, the notion of an "attraction" parameter to make certain agent groupings possible, the ability to incorporate predefined ideas (i.e. scored elements) through such attracter agents, and the ability to interact with human performers.

Example music created by *Kinetic Engine* is available at www.sfu.ca/~eigenfel/research.html.

Figure 14. The premiere of *Drum Circle* in Vancouver, April 2007, with the author controlling the system using a P5 Dataglove (photo R. Bader).

7. REFERENCES

[1] Eigenfeldt, A. "Kinetic Engine: Toward an Intelligent Improvising Instrument" *Proceedings of the 2006 Sound and Music Computing Conference*, Marseilles, France, 2006.

[2] Eigenfeldt, A. "Managing Complex Patches in Max" www.cycling74.com/story/2007/2/5/142639/8843, 2007.

[3] Eigenfeldt, A. "The Creation of Evolutionary Rhythms Within a Multi-Agent Networked Drum Ensemble" *Proceedings of the 2007 International Computer Music Conference,* Copenhagen, Denmark, 2007.

[4] Elsea, P. "Fuzzy Logic and Musical Decisions" University of California, Santa Cruz. http://arts.ucsc.edu/EMS/Music/research/FuzzyLogicTutor/FuzzyTut.html, 1995.

[5] Murray-Rust, D., Smaill, A. "MAMA: An architecture for interactive musical agents" *Frontiers in Artificial Intelligence and Applications* Volume 141, 2006 ECAI 2006 - 17th European Conference on Artificial Intelligence, 2006.

[6] Woolridge, M., Jennings, N. R., "Intelligent agents: theory and practice" *Knowledge Engineering Review*, 10, 2, 115-152, 1995.

[7] Wulfhorst, R.D., Flores, L.V., Flores, L.N., Alvares, L.O., Vicari, R.M. "A multi-agent approach for musical interactive systems" *Proceedings of the international conference on Autonomous agents and multiagent systems*, pp. 584–591. ACM Press, 2003.