Distributed Musical Decision-making in an Ensemble of Musebots: Dramatic Changes and Endings

Arne Eigenfeldt
School for the Contemporary Arts
Simon Fraser University
Vancouver, Canada
arne_e@sfu.ca

Oliver Bown
Art and Design
University of New South Wales
Sydney, Australia
o.bown@unsw.edu.au

Andrew R. Brown
Queensland College of Art
Griffith University
Brisbane, Australia
andrew.r.brown@griffith.edu.au

Toby Gifford
Sensilab
Monash University
Melbourne, Australia
toby.gifford@monash.edu

Abstract

A musebot is defined as a piece of software that autonomously creates music and collaborates in real time with other musebots. The specification was released early in 2015, and several developers have contributed musebots to ensembles that have been presented in North America, Australia, and Europe. This paper describes a recent code jam between the authors that resulted in four musebots co-creating a musical structure that included negotiated dynamic changes and a negotiated ending. Outcomes reported here include a demonstration of the protocol's effectiveness across different programming environments, the establishment of a parsimonious set of parameters for effective musical interaction between the musebots, and strategies for coordination of episodic structure and conclusion.

Introduction

Musebots are pieces of software that autonomously create music collaboratively with other musebots. A defining goal of the musebot project (Bown et al. 2015) is to establish a creative platform for experimenting with musical autonomy, open to people developing cutting-edge music intelligence, or simply exploring the creative potential of generative processes in music.

A larger and longer-term goal for the project has been a sharing of ideas about musebot programming, as well as the sharing of code. There already exists substantial research into Musical Metacreation (MuMe) systems, with some impressive results. However, much of the creative work in this field is idiosyncratic, comprising ad hoc standalone systems, and as a result the outcomes can be opaque. In such a diverse environment, it is difficult for artistic researchers to share their ideas or their code, or work out ways that their systems might be incorporated into other's creative workflows. Musebots, responding to these challenges, are small modular units designed to be shared and studied by others. By making collaboration central, and agreeing on communications protocols between computational agents, the musebot project encourages developers to make transparent their system's operation, while still allowing each musebot to employ different algorithmic strategies.

This paper presents our initial research examining the affordances of a multi-agent decision-making process, developed in a collaboration between four coder-artists. The authors set out to explore strategies by which the musebot ensemble could collectively make decisions about dramatic structure of the music, including planning of more or less major changes, the biggest of which being *when to end*. This is in the context of an initial strategy to work with a distributed decision-making process where each author's musebot agent makes music, and also contributes to the decision-making process. Questions that needed to be addressed, then, included:

- how each musebot should relate its decision-making to its music generation strategy;
- how it should respond to the collective, updating both its future decisions and musical plan;
- what decision messages should be used;
- whether decision-making agents should share common code;
- and whether decision-making agents should strictly conform to given agreements about the decision-making process.

We describe these explorations and their outcomes below, but first provide a brief overview of the musebot research context to date.

A Brief History of Musebot Ensembles

The premiere musebots ensemble occurred in July 2015 as an installation at the International Conference on Computational Creativity (ICCC) in Park City, and was followed in August 2015 at the International Symposium of Electronic Art (ISEA) in Vancouver. Since then, it has been presented at the Generative Art Festival in Venice in December 2015, the New Interfaces for Musical Expression (NIME) conference in Brisbane in July 2016, and the Sound and Music Computing (SMC) conference in Hamburg in August 2016. The first musebot ensembles are more fully de-

scribed elsewhere (Eigenfeldt et al. 2015), along with issues and questions raised by these performances.

The Chill-out Sessions – so named due to an initial desire to provide musebots as an alternative listening space to the dance rhythms of Algoraves (Collins and McLean 2014) – have consisted of ensembles curated by the first author from a growing pool of publicly shared musebots. The musebot test suite[1] currently has a repository of over sixty shared musebots – including source code – by nine different developers.

Ensembles

Curation of ensembles for installation has consisted of combining musebots based upon their musical function. For example, several ensembles consist of one or more of the following: a beat musebot, a bass musebot, a pad musebot, a melody musebot, and a harmony generating musebot. A contrasting ensemble might involve combining several noise musebots, or a grouping of only beat musebots (see Table 1). The diversity of musebots is highlighted in their presentation: if listeners don't find the current ensemble particularly interesting, they can wait five minutes and be presented with something completely different.

Table 1. Musebot types available online

Type	Number available
Bass Generators	5
Beat Generators	13
Harmony Generators	5
Keys/Pads Generators	6
Melody Generators	19
Noise/Texture Generators	16

Musebot ensembles are launched by a master Conductor, whose function is also to provide a centralised timing source by broadcasting a running clock, as well as serving as a central network hub through which all messages are broadcast (see below) via OSC (Wright 1997). A desire for the musebot project is to be platform agnostic, so musebots are standalone applications (Mac and PC) that do not need to run in the development environment. As a result, launching musebot ensembles on a single computer does take some time, as individual audio applications are started up.

Ensembles are organized as text files, in the following format:

```
tempo      [BPM]      duration      [seconds]
musebot_name              message       value…
musebot_name              message       value…
…
```

Messages currently used in the ensembles include:

- gain (0.0 - 1.0)
- delay (in seconds)
- kill (in seconds).

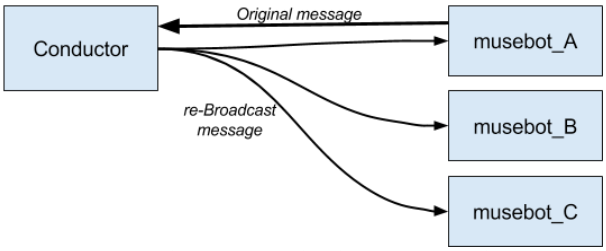
[1] http://musicalmetacreation.org/musebot-test-suite/

The **gain** value will be sent to the musebot after all musebots are loaded. This allows for rough mixing of musebot levels. A **delay** value, in seconds, will delay the launching of that musebot by that length of time. Launching begins after the delay, and may take several seconds, so it cannot be assumed that the musebot will begin playing at that specific time. A **kill** value, in seconds, will cause the Conductor to send a kill message to the musebot after that time once the musebot has been launched, taking into account any delay. Thus, "delay 20 kill 20" will cause the musebot to launch with a delay of twenty seconds, then be killed 20 seconds later. Combining delay and kill messages allow for the ensemble to dynamically vary during presentation time.

Broadcasting Messages

As mentioned, the Conductor also serves as a central hub through which messages are passed by the individual musebots. Broadcast messages are general messages sent from one musebot to the ensemble, and include the ID of the originating musebot, the message type, and the data (see Figure 1).

Fig. 1. Musebot configuration for messaging. musebot_A sends a *notepool* message with its ID, and this message is passed to the entire ensemble.



The Musebot conductor provides synchronisation (timing) and message passing infrastructure, but beyond this musebot designers need to specify protocols of coordination. Our investigations focused on strategies for minimal effective musical coordination with minimal reliance on 'top-down' intervention. This is modeled on a conductorless small ensemble where independent agents (musicians) agree on a few musical constraints (e.g., tempo, key, meter, genre) and then improvise around these.

An example message, given the musebot configuration in Figure 1 might be:

```
broadcast/notepool/musebot_A 60 62 65 67 68
```

The information shared between musebots has tended to be surface detail, such as a current pool of pitches, density of onsets, and volume. For example, the harmony generating musebots have been used to provide a generated chord progression, sent as both pitch-class sets (independent of range) and notepool messages (which indicate specific pitches). Musebots have used this information to constrain their choice of pitches, resulting in congruent harmony. Other musebots (usually the beat generators) have mes-

saged their current onset density; musebots that respond to this message can decide to follow, or oppose, this density, resulting in an audible interaction.

To reiterate, the broadcast messages passed between musebots are not themselves defined in the specification, and have instead been decided upon by the designers of individual musebots; messages can be as low level – for example, realtime timbral analysis has been messaged – or as high level as required. The messages that a musebot sends, and responds to, are contained within a separate human readable text file – info.txt – so that other musebot designers can access and use these messages, and ensembles can be more easily curated. These messages are also provided within the online musebot repository[2].

Musebot ensembles have previously been limited to five minute performances within installation settings, for two reasons. The first is diplomatic: to allow as many combinations of musebots to be presented to listeners as possible, in as wide a variety of styles as possible; the second is more pragmatic: although having virtual performers audibly agree upon certain parameters – for example, density and harmony – may suggest musically successful machine listening, these levels of interaction become mundane surprisingly quickly. The more subtle interactions that occur in human improvisation ensembles, and their development over time, have not yet been successfully modeled within communally developed musebot ensembles. This is not a limitation of the musebot messaging system (see Eigenfeldt 2016); however, designing a communal set of useful messages that satisfy a broad range of musical goals has proven to be more challenging, despite a shared document for this very purpose. This paper reports on one of the first attempts to collectively address these issues.

Challenges

Musebot ensembles have, for the most part, remained a proof of concept. Our next goal is to continue iterating musebots so as to allow further autonomy, on multiple fronts. This might include getting musebots to decide with which other musebots they play well, how to collaboratively determine key, meter and tempo, and methods to organise how and when major events occur, including the most major of all musical events: starting and ending. Of course, there is no *a priori* need for this to occur in a bottom-up self-organised rather than top-down dictatorial way; however, a research question of interest to us is what types of musebot organisations are creatively fruitful and effective, in a metamusical creative context?

Experiments in collaborative Musebots

Our recent experiments involved the four authors each developing a musebot that took on particular ensemble roles—melodic, harmonic, bass, and drums—in support of an 'experimental prog rock' performance. Each developer worked in a different software environment (Pure Data, Max, Extempore and Java) and were free to implement any algorithmic processes. Following discussion about possible minimal coordinating parameters, we chose to use *musical density* and *vote to end*. Musebots were designed to vary their generative output based on the density level (from 0.0 - 1.0) and to respond to a majority vote to end the performance. Periodically musebots broadcast

1. their own density level and,
2. a suggested future density target (based on analysis of the ensemble density profile), and
3. a vote to end or not.

As the project developed, we added the communication of harmonic context (current pitch pool) to these initial parameters. Musebot developers could choose to store the history of broadcast data if they felt it enhanced their musebot's decision-making processes.

We were interested to see whether density was a sufficient level of abstraction for musical coordination and, if density does provide a level of musical success, what could be learned in terms of effective design principles. Also we hoped to see if our musebot interactions, using only the parameters described, could be scaled to include more dimensions.

The resulting musebot source code is available, and recordings of example performances that resulted are available online. Analysis of the operation of the musebots and the resulting musical interactions are reported below.

Approaches to algorithmically addressing musical parameters

The Musebots reported in this paper were developed during a one-week programming sprint toward the end of 2016 where the authors were co-located and could readily communicate about issues as they arose. Despite authors' co-location, an interesting aspect of this Musebot experimentation was the independence of implementation and agreed parametric coordination. This separation is reminiscent of that maintained in human ensembles where the logic of a musician's decision making remains opaque to others, and interaction is based on an observation of behaviors. In this section, each developer outlines their approach to the main defining characteristics of the performance: density, deciding to end, and responding to the pitch pool. These detailed descriptions were *not* available to the other designers, nor were they discussed until after the initial performances.

Pd Musebot - Andrew Brown

The Pd Musebot (*PD_MIDI_bot*) generated melodic material based on a random walk pitch contour and rhythms derived from probabilistic beat subdivisions. Note level data were sent in real-time via MIDI to a software synthesizer for playback. The density was largely a factor of rhythmic sparseness achieved by varying the probabilistic likelihood of notes occurring. Rhythmic coherence was

[2] See, for example, http://musicalmetacreation.org/musebots/beat-generators/

maintained by coordinating between levels of metric emphasis (downbeat and subdivisions) so as to avoid filtering out metrically salient notes. In addition, density influenced note dynamic level resulting in quieter performance in less dense sections.

Pitch selection was limited to the pitch classes provided in the available pitch pool. When a new pitch pool was broadcast, the musebot updated its pitch class set to match.

Density suggestions were made on the basis of managing 'boredom' and 'confusion'. A proxy for these emotional states was the degree of variation in collective density. When the ensemble density remained static for some time (approximately 36 bars), then a density suggestion that deviated from this was made. Conversely, if the density variation over time was wide then a suggestion to maintain the current ensemble density was made.

Ending decisions involved a combination of current extreme levels of ensemble density (very low or very high) and historical trends of density change (upward or downward over several bars). A final 'catch all' was to vary the thresholds for these parameters over time to gradually increase the likelihood of end decisions, thus avoiding performances of unacceptably long duration.

Max Musebot - Arne Eigenfeldt

The Max musebot (*SynthBassBot*) assumed the role of a bass synth; as with many musebots, it made performance decisions resulting in representations very similar to MIDI data (i.e. pitch, volume, and duration) and then produced its own audio based upon a variety of pre-existing samples. At the beginning of every performance, the synthesiser settings would be randomised from within a constrained range, including sample folder, amplitude envelope settings, filter envelope settings, and filter settings. Some of these – amplitude sustain, filter cutoff – were varied during the performance based upon the environment. For example, when the agent became "bored" (described shortly), the possibility of producing a phrase-long filter sweep increased; when the density became low, the envelope sustain increased to "fill the space" with longer notes that occurred less frequently.

Pitches were derived from a pitch set provided by a separate musebot that produced a suggested harmonic progression, and were transposed to a low to low-mid pitch range. Because the root of the chord could be inferred from the messaged harmonic progression, certain notes (i.e. the root) were given a greater probability of occurring. With each new pitch set received, a new pitch ordering, which included repeated pitches and octave transpositions, was created, which was maintained for the duration of that particular harmony. New pitch orderings could be generated every phrase beginning with a 33% probability, which reordered the existing pitch collection.

Similarly, new onset patterns could be initiated with the same probability (but not necessarily at the time). These onsets corresponded to metric placements of sixteenth notes (semiquavers) within the measure: the downbeat is onset 0, the second beat is onset 4, as shown in Figure 2 A. The extrapolation to eighth notes (quavers) can be seen in Figure 2 B; this pattern is reordered by potentially switching primary onsets (0 4 8 12) and secondary (2 6 10 14). For example, the unordered pattern could switch primary onsets 0 and 8, and the second switch onsets 2 and 14, resulting in a reordered pattern of (8 14 4 6 0 10 12 2). The number of onsets performed within a measure is dependent upon the current density; given a density of 0.5, only the first four onsets would be chosen (8 14 4 6), resulting in a final ordered pattern of (4 6 8 14), shown in Figure 2 C.

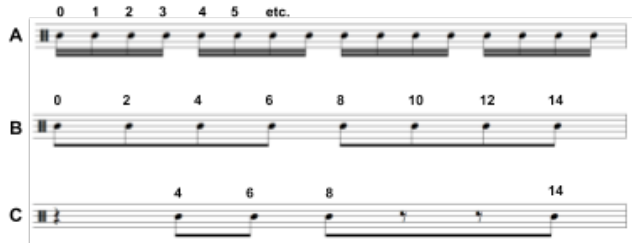


Fig. 2. Varying onset patterns in *SynthBassBot*

The *SynthBassBot* could become "bored" if it felt there hadn't been much change in the ensemble's mean density. Every two phrases (eight measures), it would check if there had been more than a 10% change in cumulative density; if there hadn't, it would begin a series of tests to decide its boredom state:

1. on each downbeat, scale its internal boredom parameter exponentially over 16 measures (the default setting for these initial run throughs was 0.2). Count the number of measures which have lacked significant change; when this value becomes greater than the scaled boredom number of measures – using the default setting of 0.2 resulted in two measures – proceed to test #2;
2. Generate a random value between 0. and 1. Compare this to the number of measures which have lacked significant change (scaled by 0.1). If the random value is less, the agent becomes bored.

If the agent became bored, it would cast a vote to alter the current density setting, and potentially execute a filter sweep. The new requested density would be derived from its own ongoing activity model, generated at the beginning of the composition.

The bass musebot monitors the ensemble density, keeping track of the trends over the previous four measures, and determines if the density has been rising, falling, or remaining stable. It uses this judgment to decide on whether to vote to end the performance, based upon the following three criteria:

1. the performance has progressed beyond a minimum number of measures (a logarithmically scaled curve between 1 and 120, dependent upon the musebot's boredom attribute), and
2. the ensemble density over the past four measures is either falling or stable, and
3. the current density is less than what the average density has been so far.

Extempore Musebot - Toby Gifford

The Extempore musebot *pad_bot* (originally called *negotiation_ybot* before its musical function solidified) adopted the role of generating background harmonic pads — long notes with slowly evolving timbre intended to be a musical backdrop to the melodic and rhythmic elements. After initial experiments, an aesthetic decision was made to produce some degree of sonic output continuously in contrast with other bots' more literal interpretation of density.

Density was manifested through rate of triggering — higher density corresponded to faster triggering of pad notes. When a new note was triggered, the currently playing note(s) faded out, though with some overlap. Because of this, higher density also resulted in greater polyphony as more successive notes overlapped.

An internal 'boredom' measure was implemented to discourage extended periods of extreme density (whether high *or* low). The boredom measure determined suggestions of both density and ending. When a boredom threshold was reached, the musebot would suggest a change of density to be as different as possible to the current density. Similarly, after a fixed minimum performance length of 64 measures, the musebot would suggest ending whenever it was bored.

Note selection was determined by the notepool broadcast message, and by the recent history of note pools. Specifically, the pitch selection for each successive triggered pad note cycled through an internal pitch-set. This pitch-set was initially identical to that of the first notepool message. Upon receiving another notepool message, the internal pitch-set is restricted to the intersection of itself with the new notepool. If there are no pitches in common (i.e. the intersection is empty) then the pitch-set is reset to the new notepool.

Java Musebot – Oliver Bown

The Java musebot plays drums. As with the other bots, the system works only with common, simple and well-studied generative processes that produce pleasant and generatively varied drum sequences. The generative strategy is based on an additive, rather than divisive, metrical approach, resulting in irregular meter structures. The density parameter, shared with the other bots, is combined with a syncopation parameter, which is used internally only. Both parameters are represented internally as integers between 0 and 9. A single kit of drum samples is used, ordered as follows: *closed hat, open hat, kick, snare1, half hat, snare2, rim, clap, ride, crash*. The ordering is used in the weighting of stochastic drum sound selection, with earlier drums in the list being chosen with higher probability. The generative algorithm creates a new pattern each 4-bars. It consists of two stages. First, a base-pattern is created. This consists of a sequence of sixteenth-note drum events, where each event may be zero or more individual hits, with each hit consisting of a drum sound and velocity. Then a "tala" structure is generated. This is a series of subpattern lengths, with a start offset for each subpattern. Subpatterns are played concatenatively. A subpattern is a playback of the base pattern with the given start offset. For example, if the base pattern is [kick,hat,snare,tom,…] and the tala pattern is [[3,1],[3,1],[2,2],...] with the first number indicating the length and the second number indicating the start offset, then the resulting pattern would be [kick, hat, snare, kick, hat, snare, hat, snare…].

The density and syncopation parameters are used to direct the content of these generated patterns in simple ways. Greater density parameters result in more events in the base-pattern, including greater occurrences of snares and cymbals. Above a certain density threshold, a kick drum on the first sixteenth note and a snare drum on the fifth is guaranteed. The density also contributes to shorter subsequences, whilst the syncopation parameter leads to a greater number of odd-length subsequences over even-length ones. A random number generator is used in the exact selection of values, but this random number generator is seeded according to the density and syncopation parameters, meaning that for any given density-syncopation value pair, the exact same pattern is generated each time, making it easier to gain an understanding of the system's behaviour (both as developer and as listener).

The musebot then performs two core actions as part of its negotiation and planning. It selects a new desired syncopation and density state from a lookup table. This lookup table is generated at start time and maps current state pairs to future desired state pairs. The table largely maps states to neighbouring states (gradual changes), but has a small percentage of 'wormholes' that map to different areas of the state space, resulting in sudden changes. Besides these properties the state transition table is randomly generated. The result is an arbitrary-but-not-random behavioural plan, i.e., the transition behaviour is consistent over time even if it derives from an arbitrary source. This has been argued by Cook and Colton (2015) to be a meaningful strategy for generative systems.

The system then broadcasts its density intention. It updates its actual density based on the agreed strategy of taking the average of all intentions. The system simply uses its desired syncopation as its actual syncopation value in the next 4-bar cycle. The system votes to stop when the density is below a threshold of 10%.

A few other minor behavioural details are as follows. The system looks at the size of the forthcoming change and if the change is small can decide not to create a new pattern for the forthcoming 4-bar cycle. If the forthcoming change is very large, it can plan to perform a fill, which involves generating a new pattern just for the last bar of the 4-bar cycle. Lastly, the system periodically updates its density-syncopation lookup table.

Analysis

All four musebots broadcast their current density 'periodically'. This relative interval was interpreted independently by the different developers, ranging from every second, to the first beat of every measure, to every beat. It was agreed that musebots must vote on a suggested density before the beginning of the third measure in every four bar phrase, although musebots were free to change their own density at any time. Despite musebot suggestions often being quite dramatic — requesting extreme low or high densities (see Figure 4) — the musebots generally progressed towards these extremes, without ever achieving these levels themselves.

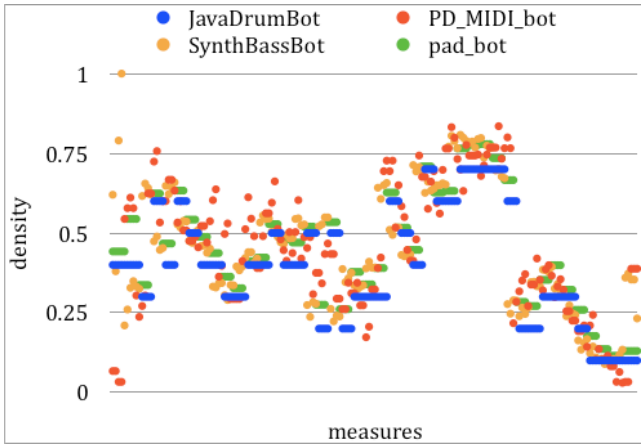


Fig. 3. Actual musebot densities over a 180 measure performance[3].

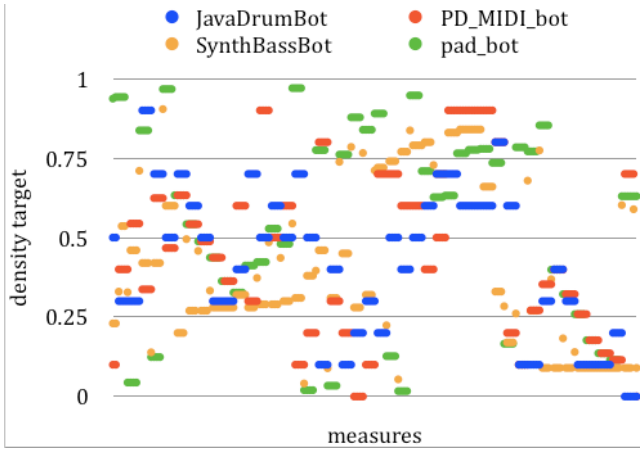


Fig. 4. Suggested density targets.

Figures 3 and 4 display the actual densities (top) versus the suggested density targets (bottom), displaying how the individual musebot interpretation of target suggestions results in an undulating mean ensemble density. The suggested targets converge three times – near measures 40, 120, and 170; in each case, these targets follow the apparent direction of the ensemble. The measures in which there is little agreement upon targets – 60-100, and 135-150 – result in the greatest discontinuities in ensemble density: a dramatic leap upward at measure 92, and downward at 135.

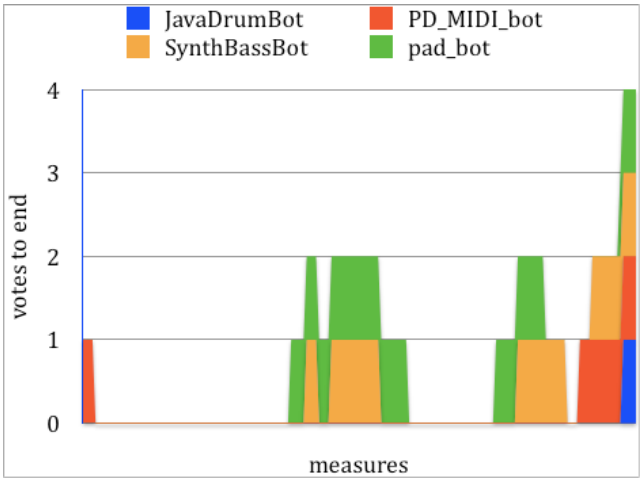


Fig. 5. Votes to end.

Figure 5 displays when musebots voted to end the performance. These votes clustered in three locations that correlate to the lowest actual ensemble densities. Comparing these locations with the suggested density targets of the two musebots that voted to end (*SynthBassBot* and *pad_bot*) shows that these musebots also proposed even lower ensemble densities. This illustrates the intended result of our system design: that the musebots are not simply reactive – voting to end at a predetermined ensemble density – but proactive – continuing their attempt to influence the overall ensemble density.

Comparing Figure 3 and 5 shows how an ending was negotiated and agreed upon once all four musebots voted to end. In fact, only a majority vote, rather than unanimity, is required for an ending; in this case, all four musebots recognised and agreed upon the potential ending. *SynthBassBot* anticipated the ending by requesting densities below the current ensemble mean – which was already decreasing – because it recognised the conditions approaching a possible ending. *SynthBassBot* has its own internal density model – including an ending; this can be seen by the alternation between two states in its density target voting: an initial vote based upon its model, then a second vote averaged between its model and the mean of the ensemble targets.

Comparing the actual ensemble density to the mean target requests (Figure 6) displays the expected correlation. In most cases, the targets precede the ensemble response; for example, the dramatic drop ¾ of the way through. Points of disagreement in between targets (e.g. the section prior to the midpoint) do not provoke any significant changes when

[3] A recording of the performance can be found here: https://youtu.be/azWnFTMCNic

compared to when the targets more closely align (e.g. the immediately preceding section).

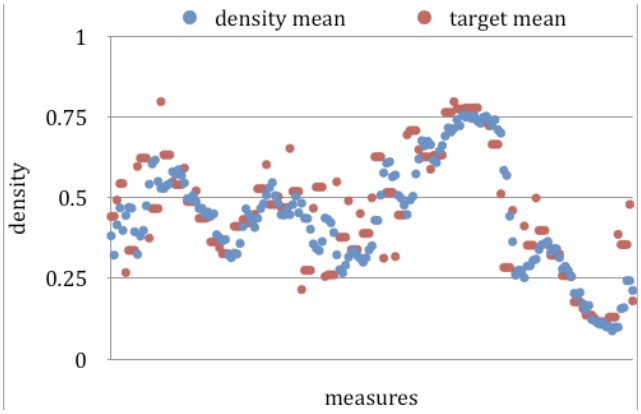


Fig. 6. Mean ensemble density versus mean target suggestions.

Discussion

An iterative practice-led development approach was adopted for these experiments. Correspondingly, it is informative to report on both the practice-based learning that arose and the results of performances generated by the musebot ensemble.

Successes and Failures

During the communal design stage, we settled upon a limited set of parameters for interaction once we determined that sending around too much complex musical data was too challenging to program. This reaffirmed notions of maintaining simplicity when dealing with complex information. We also found that limiting the communication around a single parameter – density –greatly reduced the creative demands of the task (a contrary position was suggested, in that we are not attempting to necessarily simulate a human band, so why rely upon such obvious modes of interaction?). Limiting interaction to 4-bar cycles was also convenient, although it forced the resulting music into a 4-bar mode.

Likewise, the authors found that from a practical working point of view the musebot paradigm of communication at a meta-musical layer was creatively effective, and information flow from the musical surface to the concept layer was not required *for successful interaction*. Because musebots interacting this way communicate *what they are doing*, other musebots are not required to analyse each other's output. For example, musebots need not know the specific pattern a drumbot is playing, but instead the parameters the drumbot is using to generate that pattern – e.g. density and syncopation. At the same time, this adds additional load on the programmers, who must consider both generating actions and also communicating those actions, as well as intentions. Thus the design problem seems to involve a trade off between the value of mediating interaction between the musical surface and the value of communicating agreed meta-parameters. We discussed extending the communication to further parameters – valence, to match the arousal suggested by density – but this was left for future work. An important question is whether further parameters can be included without a debilitating explosion in complexity.

Possible New Designs and Strategies

Our initial questions included:

- how each musebot should relate its decision making to its music generation strategy;
- how it should respond to the collective, updating both its future decisions and musical plan;
- what decision messages should be used;
- whether decision making agents should share common code;
- whether decision making agents should strictly conform to given agreements about the decision making process.

The first three questions are partially addressed in the above discussion. We see value in this formulation and believe it can form the basis for effective music creation as well as research around musical decision-making. However, the process also pointed to alternative designs that could be more effective.

To begin with, developing four different systems that implemented the same basic framework of decision making, let alone performing any machine listening, style modeling and so on, suggested that a shared codebase or service architecture would be effective. A service architecture would be a way to compile successful strategies in machine listening, style modeling, decision making and algorithmic composition strategies into a common repository. Individual agents would be able to query services for information such as what is going on in the music at the moment, or what would make a good note or chord to follow a given sequence given a certain style database or even information contained in the current performance.

It could also provide services to support negotiation. In essence, under the current scheme, each programmer is building a generative music system as well as very simple virtual psychology underlying the decisions surrounding density planning; for example, the agent can be fickle, or conformant. This is hard, repetitive, and prone to errors or mistaken understanding. There is great potential for farming out such decision-making to well-developed models that incorporate aspects of psychology.

Thus the experiment points to an agent-based architecture which would break quite significantly from one in which each agent is the equivalent of a human musician, listening, making decisions and generating music. An ensemble would still consist of a series of agents that would have the same agency to perform musical acts, but much of their cognitive machinery would exist in 3rd party agents that provided services and might also act as a form of distributed cognition. This would allow certain aspects of top-

down organisation and would form a heterogeneous distributed system.

Future Directions

While the performances produced by these musebots demonstrate interesting variations in the musical surface in density, these are minor developments in musebot ensemble design, albeit with the additional of a negotiated ending. From a musical perspective, a continued limitation a lack of large-scale change within the way the ensemble performs over time. A common criticism of young improvisers and composers is that the final minute, for example, does not vary a great deal from the first minute. In our case, the musebots did not alter the way in which they interpreted density, or even the way in which they fulfilled their roles within the ensemble: *PD_MIDI_bot* freely improvised melodically in the same general fashion, *pad_bot* played pads, *SynthBassBot* played the same general bass line using the same timbres, and *JavaDrumBot* performed the same basic rhythmic patterns. At no time, for example, did any part actually stop playing, or take over the foreground role. Such foreground/background negotiation is standard in improvised music, and would be the next step in achieving more musical interaction within this musebot ensemble.

Another utility of algorithmic experimentation of this kind is that it requires the articulation and testing of theories of behavior. This is a musicological activity when implementing musical performance outcomes, as in the musebots case, and is often contributive to this field (Brown et al. 2009). But the method can be applied more generally to behavioural understanding in many fields, and particularly to the dynamics of creative collaboration and interaction.

Conclusion

Musebots are based upon a state-driven communication system, rather than an output-driven system that requires feature analysis found in many metacreative systems: for example, the *P* in Blackwell and Young's *PfQ* model (Blackwell and Young 2005) or the Listener in Rowe's model (1993). Our goal in this paper was not to compare the two models, but to accept the musebot ideal as outlined in the original manifesto (2015), and explore its potential as a platform for better scaffolding musical intelligence in large modular systems.

By limiting the messages passed by musebots to current density, proposed density, and a vote to end, we feel that our ensemble of four independent systems – coded without explicit collaboration in algorithm design – demonstrated some ways in which multiple musical metacreative creators can make ensemble performances mediated by machine interaction.

The code for the musebots described in this paper is available here:
https://vault.sfu.ca/index.php/s/O4AY9DBSR5wBITI

A recording of the performance can be found here: https://youtu.be/azWnFTMCNic

Acknowledgements

The authors wish to acknowledge the Social Sciences and Humanities Research Council of Canada (SSHRC) for funding this research.

References

Blackwell, T., and Young, M. 2005. Live Algorithms. Available online at www.timblackwell.com.

Bown, O., Carey, B., and Eigenfeldt, A. 2015. Manifesto for a Muse-bot Ensemble: A platform for live interactive performance between multiple autonomous musical agents. *Proceedings of the International Symposium of Electronic Art*, Vancouver.

Brown, A. R., Gifford, T., Narmour, E. and Davidson, R. 2009. Generation in Context: an exploratory method for musical enquiry. *Proceedings of the 2nd International Conference on Music Communication Science*, Sydney.

Collins, N, and McLean, A. 2014. Algorave: Live performance of algorithmic electronic dance music. *Proceedings of the International Conference on New Interfaces for Musical Expression*, London.

Cook, M., and Colton, S. 2015. Generating code for expressing simple preferences: Moving on from hardcoding and randomness. *Proceedings of the Sixth International Conference on Computational Creativity*, Park City.

Eigenfeldt, A., Bown, O., and Carey, B. 2015. Collaborative Composition with Creative Systems: Reflections on the First Musebot Ensemble. *Proceedings of the ICCC*, Park City.

Eigenfeldt, A. 2016. Exploring Moment-Form in Generative Music. *Proceedings of the Sound and Music Computing Conference*, Hamburg.

Rowe, R. 1993. Interactive Music Systems. Cambridge, Massachusetts: MIT Press.

Wright, M. 1997. Open Sound Control - A New Protocol for Communicating with Sound Synthesizers. *Proceedings of the International Computer Music Conference*, Thessaloniki.