CPSC 675 Project : Multiagent Music Composition

Brae Stoltz

Due date: March 26, 2018

1 Idea & Model

The underlying idea of this project was to compose music using what we have learned in the multiagent systems (MAS) course. We chose to model the system as two agents that place notes in a composition. These two agents correspond to a lower (bass) part, and an upper (soprano) part.

The agents have a limited perception of the past, and no perception of what notes the other agent is choosing. To then choose notes that fit the entire composition, agents must communicate their perceptions to one another.

This program operates a little different from a conventional MAS. Traditionally, MAS systems are designed to be used in real-time applications such as monitoring. However, music composition is inherently not a real-time problem. Our algorithm thus operates more to the tune of an optimization algorithm. That is, our system gives an output, which is the music composition.

2 Formalization

This formalization closely follows the formalization set out in [1], apart from the musical definitions.

2.1 Agents

Our system has two agents Ag_L for the lower agent and Ag_U for the upper. So the set of agents AG is $AG = \{Ag_L, Ag_U\}$.

The allowable pitches an agent can place is a set P. We use MIDI note values for these pitches so that we only have to deal with positive integers. The allowable pitches for the agents are defined as $P_{Ag_L} = \{38, 39, 40, \ldots, 60\}$ and $P_{Ag_U} = \{55, 56, 57, \ldots, 79\}$.

The allowable durations is defined as $D = \{W, H, Q, E, S\}$, where W is a whole note, H is a half note, Q is a quarter note, E is an eighth note, and S is a sixteenth note. Importantly, the allowable durations are the same for all agents.

A note is a tuple $N_i = \langle p, d, \varphi \rangle$ where N_i is the note at the i-th position and p is the pitch, d is the duration, and φ is the position of that note.

Because a sixteenth note is the smallest delineation in time in our model, position can be defined as the number of sixteenth note durations from the beginning of the composition. At any given position, there is either a note immediately starting or one that is already playing. By using position, agents can reason whether two notes are happening at the same time.

2.2 Environment

The environment Env is a triple $Env = \langle E, e_0, \tau \rangle$ where E is the set of all environment states, e_0 is the initial environment state, and τ is the state transformer function.

Since the environment is the music composition itself, E is the set of all possible music compositions (of two parts). There are limitations here, for example our system deals only with a subset of possible note durations and restricts pitches. Nevertheless, the set of all environment states is infinite.

An environment state can be defined as $e = \{\psi^L, \psi^U\}$, where e is some environment state in E, ψ^L is the lower part, and ψ^U is the upper part. Each of these parts are sets of notes.

We defined the initial environment state as $e_0 = \{\psi_0^L, \psi_0^U\}$ where $\psi_0^L = \langle 48, W, 0 \rangle$ and $\psi_0^L = \langle 72, W, 0 \rangle$. So the composition initially starts with a whole note in each part, with a pitch of $C3$ for the lower part and $C5$ for the upper.

2.3 Actions & Runs

In our system, there is only one action, and that is to place a note in the composition[1]. Formally, $AG = \alpha$, where AG is the set of all actions. In this case, we can simplify and just use α. Even though there is only one action, it is based on an agent's beliefs, and so the action will not simply place a single predefined note.

A run r is an interleaved series of environment states and actions:

$$r : e_0 \xrightarrow{\alpha} e_1 \xrightarrow{\alpha} e_2 \xrightarrow{\alpha} e_u$$

Here e_u is the terminal state. Importantly, the system does not have any specific terminal state; the algorithm runs until stopped by a human. The output of the algorithm is then e_u.

Let R be the set of all runs, R^E be the set of all runs ending in an environment state, and R^α be the set of all runs ending in the action. The state transformer function τ, as stated in the Env, is defined as $\tau : R^\alpha \to E$. An agent simply performs an action on the environment; $Ag : R^E \to \alpha$.

3 Placing Notes

Although our system only has one action, it is a complex one. It involves three steps:

1. Selecting the duration
2. Selecting the target interval sounding
3. Selecting a pitch that matches this interval sounding

[1] Note that this action is atomic

An interval sounding is a metric of how pleasant an interval is to the ear [2]. An interval is simply the distance between two notes. We assign an ID to each specific interval and associate a sounding.

Interval ID	Interval name	Interval Sounding
0	Unison	Perfect Consonance
1	minor 2nd	Sharp Dissonance
2	Major 2nd	Mild Dissonance
3	minor 3rd	Imperfect Consonance
4	Major 3rd	Imperfect Consonance
5	Perfect 4th	Perfect Consonance
6	Tritone	Sharp Dissonance
7	Perfect 5th	Perfect Consonance
8	minor 6th	Imperfect Consonance
9	Major 6th	Imperfect Consonance
10	minor 7th	Mild Dissonance
11	Major 7th	Sharp Dissonance
12	Octave	Perfect Consonance

The interval ID I of two notes N_1 and N_2 is defined as $I = (|N_1 - N_2|) \mod 12$.

Composers tends to gravitate towards imperfect consonances over anything else, and so agents should have a higher probability of selecting notes that cause them. It's generally known that repeated perfect consonances has the effect of boring listeners. Sharp dissonances have an irritating effect. The probabilities are another adjustable parameter of the system. This is an oversimplification of counterpoint [3].

3.1 Duration & Target Sounding Selection

Each duration and target sounding is assigned a probability of occurance. We then use a selection algorithm based on cumulative probabilities. To illustrate this we go through a simplification of the duration selection. Say we have the following probabilities of selecting a duration:

Duration	W	H	Q
Probability	0.2	0.3	0.5

We then find the cumulative probabilities:

Duration	W	H	Q
Probability	0.2	0.5	1

Now we generate a random number between 0 and 1. If that number is between 0 and 0.2, we select W. If it is between 0.2 and 0.5 we select H, and so on.

3.2 Pitch Selection

A pitch is selected that matches the target sounding S_t. To do this, we go through all possible pitches available and calculate the interval ID I_p of it and the past note (the note played previously). We also calculate the interval ID I_o using the pitch in the other part (the note that is overlapping in time). The corresponding sounding to the intervals, S_p and S_o, must both be equal to the target sounding ($S_p = S_o = S_t$).

There are two cases where this can go wrong:

1. In the case where we cannot find a pitch that satisfies $S_p = S_o = S_t$.
2. In the case where the agent does not know the other agent's pitch, which is the case when one agent gets too far ahead.

In both of these cases, we must make a decision based solely on I_p. Thus we search for a pitch such that $S_p = S_t$. We will *always* be able to find a pitch that satisfies this.

Furthermore, we restrict the pitch to be within a certain interval, $M6th$ of the past note to avoid the case of a part jumping to far away notes. This interval is often chosen to be the limit for a note to jump, as in classic counterpoint [3].

4 System Behaviour

4.1 Perception & Lagging

Each agent has a limit on their perception. In other words, we can say that the agent only remembers their recent actions, forgetting those too far in the past. An agents perceptions can be said to be their beliefs.

We say that an agent is lagging if it is too far behind another agent. From this, agents that have a higher probability of selecting shorter notes will typically lag behind other agents with a higher probability of selecting longer notes. A larger lagging length has the effect of forcing an agent to make decisions based only on their perception more often, but blocks agent agents less.

Typically, parts lower in pitch tend to play longer notes, and parts higher in pitch tend to play shorter notes. However, we let the user select these probabilities (see §6).

4.2 Goals, Beliefs & Communcation

The bass agent starts with the belief that it is their turn (*myturn*), and the soprano agent starts with the belief that it is not their turn (∼*myturn*). Each agent starts with the goal of asking the other agent to give them their past notes (the notes in range of their perception), however they can only ask if it is their turn. Once an agent receives these notes, they

- add the notes to the *vertNotes* belief,
- execute the action,
- remove the *vertNotes* belief, and
- forfeit their turn.

For giving their notes, the agent

- sends their perceived notes to the other agent,
- notifies the other agent that they can now place a note, and
- believes it is their turn.

In the case of action failure, the agent forfeits their turn.

4.3 Keeping in Range with Action Failure

Failure to place a note happens when the agent realizes that it is too far ahead of the other agent. It does this by using the notes that the other agent communicated to it. Specifically, it calculates the distance between the current position and the formost note in the notes it has received, and then sees if it is greater than some threshold. This threshold can be set by the user (see §6).

5 Program & Usage

For musical data structures we use the jMusic library [4]. This also allows us to view the music compositions in a GUI. Program output is in the form of a MIDI file, which is done through the jMusic library. This MIDI file can be open in a variety of different software, including open source software. Modern OS' have support for MIDI files, as they are a standard.

The program searches for a settings.txt file to load the different algorithm parameters from (see §6). If no file is found, the default settings are used. Apart from the standard Jason console, there are two other windows, the user input window and the composition GUI.

5.1 User Input

The user input window allows the user to insert notes *in real time*. This uses the same function the agents use for executing actions, and so is atomic. That is, an agent placing a note should not effect the system.

5.2 Composition GUI

There are many different display times for scores (compositions) in jMusic. We allow the user to choose between three types:

- **NOTATION**, which shows musical notation of the composition.
- **MIDISCORE**, which shows a more MIDI-esque display of the composition. In jMusic this is refered to as the Score view.
- **SKETCH**, which shows the composition as a graph.

The length of the music composition quickly becomes to big for the NOTATION view to show the entirety of it, and the SKETCH view does not show much information. The best view for this project is the MIDISCORE view, as it shows a good amount of information and scales the composition.

Because jMusic was not built for real-time applications, we have to use quite a hacky method to display this score. Their API does not allow for updating within the window, and so to update the score view we dispose of the swing panel and create another view. This is the only way around this problem without changing jMusic itself.

6 Program Arguments

1. **Delay of agents**, in miliseconds.
   - Default: 1000ms
2. **Perception length** Ψ, in terms of whole notes. For example, a value of 1 will make the perception length 1 whole note.
   - Default: 1
3. **Lagging length coefficient** L. Here $\Phi = L * \Psi$, where Φ is the lagging length.
   - Default: 0.25
4. **Beats per minute (bpm)**. This is stored in a byte and so values above 128 are not possible.
   - Default: 100
5. **Bass duration probabilities**: a list of probabilities of selecting certain note durations for the bass part. Organized from whole note to sixteenth note.

- Default: 0.1, 0.1, 0.7, 0.1, 0

6. **Tenor duration probabilities**. Unused in the current implementation.
7. **Alto duration probabilities**. Unused in the current implementation.
8. **Soprano duration probabilities**
   - Default: 0, 0.1, 0.2, 0.5, 0.2
9. **Sounding probabilities**: a list of probabilities of selecting certain target soundings. Organized as: {Perfect Consonance, Imperfect Consonance, Mild Dissonance, Sharp Dissonace}.
   - Default: 0.04, 0.9, 0.04, 0.02
10. **User Input Flag**: boolean value letting the system know whether to open the user input window.
    - Default: true
11. **Use GUI Flag**: boolean value letting the system know whether to open composition GUI.
    - Default: true
12. **GUI Display Type**: What type of jMusic display to use. Allowable values are: NOTATION, MIDISCORE, or SKETCH.
    - Default: MIDISCORE

Note that there is a 0-th argument for the number of agents, although this will not work for any value other than two.

7 Developmental Issues

7.1 Problems

The most difficult part of the project was designing agent communication. Although the code to do so is minimal, much time went into devising it.

Another difficulty was working with Jason. Particularly that Jason does not let generic types as beliefs. Implementing our note definition means that the agent needed two separate beliefs, one for the pitch and another for the position. However we were working with lists of notes, so we had to use two separate lists instead of one list with tuples.

7.2 Limitations

Originally, the project was thought to model a four-part composition, meaning that there would be four agents. Within the code and algorithm parameters, there are hints to this, however the problem quickly became too daunting. Devising a communication model for four agents was deemed too difficult in the given time frame.

We also hoped that the program would be capable of selecting the number of agents. However, the problem is such that a one, two, and four agent solution is vastly different.

7.3 Changes

Originally the system was thought of to able to add rests into the composition when it was unsure about the pitch selection. The simplification to two agents instead of four eliminated the possibility of not being able to find suitable pitches. We also needed to change how notes were defined. Particularly, we needed the notion of position in order to reason about notes happening at the same time.

Fortunately, we were able to allow the user to interact with the environment in real time as well as display a GUI thanks to jMusic.

7.4 State of the Project

The two agent case is complete from a MAS point of view, although there might be a more optimal way to communicate (presumably one not based on turns). From a music theoretical point of view, there is much to be desired. Note selection is based of a simplification of counterpoint, and so could be made better. From a presentation point of view, the program can be made better by using a better musical library that allows real-time updates of a particular composition. However, there does not exist such a library to our knowledge.

Source code is available at

https://github.com/braem/MultiagentComposition

References

[1] Michael Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, 2009.

[2] George Thaddeus Jones. *Music Theory: The Fundamental Concepts of Tonal Music Including Notation, Terminology, and Harmony*. 1974.

[3] Alfred Mann. The study of counterpoint. *WW Norton, New York*, 1943:1971, 1965.

[4] Andrew Sorensen and Andrew Brown. jMusic. https://sourceforge.net/projects/jmusic/, November 1998.