Alex Graham

To download the code for Improvbot and an MP3 of a demo from my Senior Recital, please visit:
http://alexgraham.net/improvbot/

Improvbot Documentation

Improvbot is a piece of software that is designed to improvise music in real time with a human performer. The human performer plays at a MIDI keyboard, and Improvbot improvises a solo melodic line (i.e., an oboe). Since this happens in real time, it allows for the human and the computer to react to each other as they perform together, creating a true group improvisation.

Improvbot is written in Common Lisp and Max/MSP. Lisp does all of the heavy calculations while the Max patch is responsible for timing as well as MIDI input, routing and output. The two communicate over UDP through a Lisp loop/Max object pair I wrote called "Lisper" which allows a Max patch to send data to, and call Lisp functions, and allows Lisp functions to send data to named Max objects (similar to the "send" and "receive" objects in Max). This allows both Max and Lisp to work together, filling in for each other's deficiencies.

Input

Once activated, Improvbot begins listening through the midiin functionality in Max and then sends note data to Lisp functions that store the data into memory. Max sends pitch and delta/duration information via the midi note parsing borax object, and sends groups of pitches for harmonic analysis with the quickthresh object.

Analysis

Currently, Improvbot analyzes the music it hears in a few key ways. The most critical is **beat detection.** This allows Improvbot to play along with the performer at the correct tempo. First, a list of note lengths is generated, based off a quarter note (1) being given the length of 800 ms. This list consists of the ratios

(1/6, 1/4, 1/3, 1/2, 2/3, 1, 2, 3, 4, 1 1/2, 1 1/4, 3/8)

which represent

(sixteenth triplet, sixteenth, eighth triplet, eighth, quarter triplet, quarter, half, dotted half, whole, dotted quarter, dotted eighth, dotted sixteenth).

As each note is input, the delta time (the time between a note and the note before it) is compared to the list of possible note lengths. When the closest one is found, it is averaged into the time, and the noteval table is updated according to this new time. This method is quite effective as it allows for the length of the beats to shift slowly over time.

In duo/accompanied mode, every pitch value is sent for **harmonic analysis**. As the improvisations are currently meant to be freely non-tonal, these pitches are analyzed through prime forms, an interval vector list, and a list of recent pitch classes. In order to create prime forms and vector lists that reflect

the current performance, Improvbot uses rotations and sum rotations, a rotation taking the oldest number out of a list as it inserts the newest one in, and a sum rotation subtracting the oldest number out as it adds the newest number in.

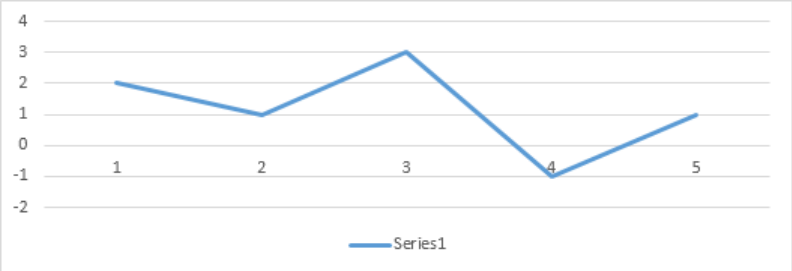
In terms of **melodic analysis** Improvbot is only able to analyze full melodies in call and response mode, as it currently does not have the capability of discerning the melody from the accompaniment when the piano is being played with both hands in duo mode. This is a difficult task which I hope to accomplish in the future using either an artificial neural network or normal fuzzy logic. Currently, in call and response mode (which was mainly used as a test bed for the development of melodic generation), when the "call" melody is heard, its notevals are analyzed using the system above, and the notes (pitch, noteval and velocity) are stored in a list. This list can then be used to create a response in multiple different ways, including using the rhythms and new notes, using an inverse melody of the call, and generating notes using a shape map (explained below) based on the call. Similar techniques will be used within duo mode when Improvbot is able to discern melody in the piano part.

Generation

Improvbot generates its melodic lines using a variety of techniques. Improvbot's control loop works in phrases. A phrase is generated as Max plays each note from the prior phrase. Each new note is generated (pitch, velocity and note value) based in the current position in the phrase until the phrase is done and a new phrase is created.

Phrasing was perhaps the biggest breakthrough I had while creating Improvbot. I attribute much of its current success to this concept. The phrase gives shape to a melodic line. When humans improvise and write music, we don't just consider what we're hearing at the moment and the note we just played, we consider where we started and where we're going.

A phrase simulates this by using what I call "shape maps." A shape map is generated for pitches and velocities. Each shape map is made up of 5 (although it could be more) numbers, from -5 through 5. This creates a graph over time (based on the length of the phrase) such as this: (2 1 3 -1 1).



This creates a map for the note generator to follow. Based on the current position in the phrase, the note generator will try to use intervals that keep the pitch roughly along the line of the graph. This creates melodic lines that have actual direction instead of pitches moving in random, erratic patterns. The velocity maps work in the same way. Currently these shape maps are generated randomly, but in the future will be generated to be similar to a humans, generated from analysis of human improvisations.

Rhythm Generation is done using a simple Markov table consisting of groups of note values and rhythms. Depending on the group of the note value/rhythm that was used previously, a different probability is applied to the groups for the following note value. For example, after a sixteenth note syncopation , there is a much higher chance for a normal eighth note pair to occur next than an eighth note syncopation. I created these tables based on speculation, and while they create convincing rhythmic patterns, I plan in the future to generate similar, but much more complex Markov tables based on pattern recognition of human input improvisations and music from the repertoire.

Pitch generation is done using a fuzzy logic system that takes multiple factors into account when generating notes. The fuzzy logic system itself works by generating 3 different intervals randomly, then scoring each note based on these different factors. The highest scoring interval, if above a certain threshold, is used. The most important factor is the position on the shape map as discussed above. When scoring an interval, if the interval creates a note that is much higher or lower than the current value of the shape map, it will get scored lower than a note that is closer. The fuzzy logic engine also takes into account the instrument range (the engine tends to stay out of the extreme registers), favoring smaller interval and the current pitches and vectors being used by the human.

Velocity generation, in order to create a smooth shape, currently follows the velocity map exactly. In the future I would like to base velocity on the current rhythm and melodic movement as similar to the way a human player might.

Output

Once Improvot generates the next note in the melodic line, it inserts it into a queue of note events. Each note event, when sent to Max, contains a pitch, velocity and duration. The duration is generated by multiplying the current "beat" duration by the note value. After receiving the note and playing it, the duration of the note is sent to a timer. Once the timer runs out, Max queries Lisp for the next note in the queue.

Future

As discussed throughout this document, I have plans to improve Improvbot substantially. First and foremost is through pattern recognition. This will assist with analysis as well as melodic generation. Essentially this will consist of a number of algorithms designed to find different types of patterns (pitch, rhythms, pitch and rhythm relationship, etc). As Improvot is exposed to more and more music (through human improvisations, as well as music from the repertoire input in), it will "learn" more and more as to what a melody should sound like under different circumstances. Ideally, this will not only include the patterns, but also data on how and when they are used. This method of "teaching" the software about music by actually having it “listen” to examples of the music it is designed to mimic is similar to how a human learns to play and compose music. There is only so much a person can learn through rules; only from exposure to music can a person internalize how to really create music. This idea of using pattern matching can be applied to a variety of different aspects such as form, melodic generation, rhythmic generation, style and mood. One area Improvbot is currently lacking is in the ability to recognize when the human player is starting a new phrase or near the end of the currrent one. With the correct algorithms and enough musical phrases to analyze, it may be possible for Improvbot to recognize certain

hints in the form of velocity or time between notes of when a phrase is ending without actually having rules explicitly programmed in.

In the end there are a multitude of different ways to expand upon and improve Improvbot. Ideally, Improvbot will be able to create music that is convincing and not only be able to follow along with a human player, but also give its own input. I hope to make Improvbot act as much like a human when improvising as possible. This means that Improvbot should not only be able to pick up musical cues such as changes in mood and style, but also recognize when it would be appropriate to incorporate its own mood and style. It will use pattern matching techniques as discussed above to generate a long term database of how music can be analyzed and generated, but also use short term memory to create a context for the current improvisation based on what is happening in the present moment.

I feel there are different directions I can take with Improvbot's technology. As well as developing Improvbot as a singular entity, I hope to create a library of Improvbot's code that other software can call to generate music based on real time input. I feel this allows for limitless possibilities, such as a system that expands the abilities of a human improviser, or a music generation system that reacts to different types of input other than musical input, such as that of interactive media.