**Sharp11 – Semester Wrap-up**

I had two primarily goals for this semester. The first was to reorganize Sharp11, both in terms of adding functionality, and deciding how to structure it into multiple libraries. The second was to decide how I wanted a musician to interact with the jazz automaton, and to build an interface to allow for that.

The first step of reorganizing Sharp11 was to decide, more philosophically, what I wanted it to be. Previously, I had been branding it as a music theorization and improvisation engine. The addition of the automaton into the Sharp11 ecosystem caused me to rethink this. I decided that the automaton should be thought of as a project under the Sharp11 umbrella, rather than as a core part of the Sharp11 module. By this logic, so should the improv engine. I wound up separating out the improv logic from the core Sharp11 library and rebranding Sharp11 as a music theory multitool with a jazz focus. In general, the heuristic for whether a feature should be a separate library or part of Sharp11 is this: is the functionality a product for an end-user, or is it a tool that other projects could reasonably be built on top of?

Many projects will include both of these components. This wound up being the case for the automaton, and the development process wound up consisting of building the automaton in a branch of Sharp11, then refactoring the parts that made sense to remain in Sharp11 and extracting the rest. In general, I think this is a good way to develop in the Sharp11 ecosystem. It is reasonable that a Sharp11 project will wind up having some functionality that should become part of the core Sharp11 library. In the case of the automaton, I wound up adding modules for things like Mehegan roman numerals, chord charts, and corpora, all of which now live in the core Sharp11 library.
Throughout the semester, I also wound up making some general fixes that did not have to do with the automaton. For example, I added a module for note durations, made improvements to the chord module and improv engine, and refactored the MIDI module to decouple it from any object-specific logic. I also received some engagement from the community. I accepted two pull requests, one to add an option to return a list of possible names for a chord instead of a single chord, and one to fix an issue that occurred when a certain function was called with an empty array. Finally, I wrote a lot of tests and a lot of documentation. I can’t say it was a highlight of my semester, but I’m very glad that I did it.

The Sharp11 ecosystem now consists of the following libraries:

- **sharp11**: The core Sharp11 library—a music theory multitool with a jazz focus that provides the infrastructure necessary for building cool computer music projects such as…

- **sharp11-improv**: A jazz improvisation engine. Given chord charts, this library can generate jazz improvisations according to a variety of parameters.

- **sharp11-jza**: A probabilistic automaton model for analyzing and generating jazz chord changes, along with infrastructure for constructing such automata manually, training on data, analyzing chord sequences, generating chord sequences, and analyzing the internals of the automaton.

- **sharp11-jazz-parser**: A library for parsing .jazz files into Sharp11 chord chart objects. The parser is designed specifically for use with the iRb Corpus.

- **sharp11-irb**: A hosted version of the iRb Corpus parsed into Sharp11 chord chart objects. Can be used with sharp11-jza for analysis and training or with sharp11-improv for generating improvised melodies.
- **sharp1-web-audio**: A module for handling playback of Sharp1 objects in the browser.

The second goal of the semester was to figure out what I wanted to do with the automaton and present it in an accessible way. While the analytical aspect of the automaton (feeding it a song and getting an analysis) is interesting, it would have required a much more cumbersome interface and a solution to the difficult problem of choosing a single analysis from a set of possibilities. Instead, I decided to focus on the generative aspect of the automaton. It’s more fun for a musician to play around with, and it complements the improv engine nicely.

The first step to doing this was creating a module within the automaton for manipulating sequences of generated chords. My goal was to design this module such that building an interface around it would be fairly trivial. This meant thinking about what sorts of things a musician might want to do in order to generate a sequence of chords. The most apparent thing was to add and remove chords to/from the end of the sequence. I decided another important feature would be able to reharmonize a chord in the middle of the sequence. After all, reharmonization is one of the main themes of this project.

The difficult problem here comes from the fact that I wanted a notion of harmonic units that might be larger than a single chord. For example, the automaton allows for setting up major, minor, and diminished chords with an elaborating ii-V progression. These ii and V chords correspond to individual states, however we want to think of them in this case as a unit. The only way this information is encoded in the automaton is with a notion of end states. The automaton knows that the states corresponding to the ii and V chords are not end states, which means that they are part of one of these larger units. When we hit one of these states, we can keep following possible transitions until we hit an end state, which results in a harmonic unit.
This poses a problem in reharmonization. Reharmonization entails getting rid of a portion of a sequence, and then trying to stitch it together with another transition. Connecting with a single transition is not difficult: simply find all transitions between two particular states and pick one probabilistically. However, this becomes much more difficult when we consider larger harmonic units, because it requires us to look ahead. We are now considering paths instead of individual transitions, and there are far too many to do this in a reasonable amount of time. The solution I came up with is to only allow for paths of size two and three, which allows me to special case them and compute them more efficiently.

Another issue in generated sequences was uniqueness. You can have a chord that is repeated in the same state or in a different state. However, when generating chords that don’t have attached durations, it is unlikely that you would want to generate a repeated chord. It’s easy when generating a new chord at the end of a sequence to make sure it’s not the same as the last one, but much more difficult when reharmonizing a sequence in the middle. The solution is to check for uniqueness and attempt to splice repeated chords in a way that allows you to stitch the sequence back together, which is usually, but not always, possible.

One of my big takeaways from this project is that I should have designed the automaton with an actual use case in mind. I designed it to be what I thought was the best abstract representation of jazz harmony, not necessarily the most user-friendly way to generate and manipulate chord sequences. I went into the project not knowing exactly what I wanted it to be to an end-user, which might be an okay way to do research, but will likely lead to problems later. I realized that I should have treated harmonic units as such. I also realized that my model of chords as transitions and functions as states is simple mathematically, but proves to be quite cumbersome in practice.