Songsingums design

Instruments.fs

* Stores the waves, and instrument sounds that are used when generating a note
* Maybe make the instruments as compositions of sin waves, like Dmitry always wanted

TimeGenerating.fs

* Creates the different rhythms and note placements
* Functions like makeTriplet, or makeTwoNotes, or generate a chunk or measure from a given motif
* More complex functions like generate a bassLine that takes a melody and generates a bassline that meshes with its general rhythm
* Takes things like erraticity and how short or long notes can be in general, and whether or not they’re more 4/4 or different type time signature

NoteGenerating.fs

* Takes a group of note placements and generates the notes,
* So something like playMotif(motif) would have to use the makeMotif type function from TimeGenerating.fs and then put the notes in at the right spot
* Would have to decide to put chords in or how far in between the notes should be based on their lengths (big jumps in pitch in a short time are sometimes jarring) (also to make a way to tell it not to put chords in, because some instruments can’t do chords)
* More advanced functions that take a bassline and a melody, or a melody and a harmony and make the notes mesh,
* A basic function would be fillMelody(erraticity, pitchWeirdness) and different functions or options to use a preset key
* pitchWeirdness would be a discriminated union of options that said essentially how big the pitch ratios can be.

Singer.fs

* Singers would have a list of motifs if it uses them, its list of valid pitch ratios if it had a key, and its members would be the options in the discriminated unions that determined erraticity and tempo and how dissonant the notes would be allowed to be, to minimize how many numbers are actually in the singer class.
* Singers would have 1 or more states that would inform what instruments it can use, and other such properties, to control things like song progression
* Would have minimal amount of functions, just like nextMeasure, and getting the different members
* The nextMeasure function will handle all sound creation, and will take the current state it’s generating in as its argument. It will output a complete soundbuffer to be put into the audio
* Generating a sound will be threefold. First, the note placement will be created, and will exist independent of what notes actually occupy the spaces it generates. Then, notes will fill the placement, creating the sound chunk in a way that resembles a sheet music. Then, different instruments can use the same sheet music and it’ll sound different.
* Potentially, could add a fourth step that is a filter, like a low pass filter
* Only one type of singer will be made, its nextMeasure function will be an argument, as will its motifs and its key (or its allowance for dissonance) and its instruments in the different states
* Think of states like blue rondo a la turk. There’s the 9/8 parts with piano and clarinet, and the 4/4 parts with clarinet and upright bass. Those two things would be different states. A state could also emulate layering of voices, like how the arby n the chief theme starts with the percussion, then adds more parts until it crescendos and dies off. That could also be expressed by a state

POTENTIAL PROBLEMS

* The different states for a singer might be different enough to each need their own nextMeasure function, which makes them almost a different song. Maybe they’d be too different to be logically in the same Singer object. However, if you think about a Singer as the thing that creates a song, then they are similar enough because you can have a song with two very different parts (like blue rondo). Maybe some things should be completely global values, but maybe some things will be the same just by virtue of every state’s version of it being the same, not because they’re forced to be.
* Motifs and themes (first off, how to differentiate them). Themes are probably more overarching, but that could potentially be indistinguishable from a state. Motifs are obviously something like a riff. You can have the motif be changed, like change one note at the end in it, but have the same not placement. A motif could be played on one instrument predominantly, but it can sometimes be played in special circumstances by multiple instruments. Decide whether or not motifs have different levels they operate on, like have one motif be just note placements, another be some note placements and note values, and a different one that locks in the instrument too. And for these, should the lower levels of construction be accessible if it already has a preferred instrument, or note value? Should it be allowed for some motifs to be more incomplete and others more complete, even if you can access every level of each one of them? I’m leaning more toward the motifs having any level of completeness they want, with every level accessible. They might all have to have a full implementation to make it easier to store them in the motifs collection. Maybe add a number variable that just says what level they prefer to be used at, even if that’s clunky?