Revised Design

Chau Vu, Diony Rosa, Kharis Xavier

April 2, 2013

This document records changes that we have made since design-milestone.pdf.

# Lexer

No significant changes.

# Parser

Instead of having repeats as an instance variable, the Piece object now has measures, which is a list of measures parsed by the parser. If the abc file has more than one voice, the parser will parse all parts played by voice 1 first, skipping parts played by other voices; then it will do the same for voice 2, voice 3, etc. Each measure also knows which voice it belongs to.

Also, repeats are no longer part of the AST. Instead, the parser transforms repeats encountered in the ABC file into multiple measures with the proper pattern.

# Player

The Player uses the visitor pattern to visit every node of the AST. The Player has a Map of String → Integer called 'positions' that keeps track of each voice's current number of sequencer tics into the song. In the event that a piece has no voices specified, a default voice is used. For each measure visited, the currentVoice instance variable keeps track of the measure's voice, in order to update 'positions' properly.

There is a Map of Pair<Character, Integer> → Accidental called currentAccidentals. currentAccidentals keeps track of which notes on which octaves have an accidental applied to them. If the note does not explicitly have an accidental applied to it, currentAccidentals is used to lookup the proper accidental. If there is no accidental in currentAccidentals, Player refers to the Key signature in order to apply a set of default accidentals. For example, in G Major key, all normal F notes have a sharp applied to them by default.

For every chord encountered, each note in the chord has its MIDI note calculated as a function of its pitch, octave, and accidentals, and they are all sequenced at the same time.

# Test

Lexer (in LexerTest.java)

There are currently 3 tests in lexer. The first two tests for piece1 and piece2. The third test tests for tokens that are not covered by the first 2 tests. The strategy is to add tokens manually to expectedOutput, and compare the string it prints out with the output of Lexer.

Parser (in ParserTest.java)

We implemented a lot of tests here because the parser was divided into two parts: body and header.

The first two tests test for header. The first one tests if all header fields are parsed correctly, given that they are all present. The second test has to parses the header with no optional fields (meter, tempo, defaultlength, and composer) and is asked to print out default values for these fields. Because X, K, T are compulsory fields, we simply throw an error if any of them is not present in the header.

From here on, we are testing the body of the music file.

testBody() tests different basenotes (lowercase, uppercase), with different ways to represent notelength(1/4, /, 3/, … ) and their combinations ( in chords, tuplets).

testSingleNote, testSingleChord, testSingleTuplet, testSingleRepeat, testRepeatsx, testEmpty are tests for different units of the body. See ParserTest.java for more spec.

Player ( in PlayerTest.java)

We played all given pieces in docs and all of them work well.

We added more abc test files in docs/, they are: AddingtonHighland.abc, afterMidnightWaltz.abc, LesLignes.abc, LesLignes\_NoVoices.abc

AddingtonHighland.abc is good for our testing against repeats without a beginRepeat token.

afterMidnightWaltz.abc has a lot of tuplets and chords and their combinations; it also has a very similar repeat pattern with AddingtonHighland.abc.

LesLignes.abc has three voices and in each voice there is a repeat pattern.

LesLignes\_NoVoices.abc is the same as LesLignes.abc, except that the three voices are not specified in the header. When the parser realizes this problem, it should throw a RuntimeException saying that Voice x was not specified in header.