Design Milestone

Chau Vu, Diony Rosa, Kharis Xavier

March 28, 2013

1 Overview

The implementation is split into three major parts, the Lexer, Parser, and Player. The lexer reads the input file, and generates a list of Tokens, representing basic lexical elements of the file, and pass them to the parser. The parser will read the list of tokens and produce an AST representing the piece. The player will finally walk this tree using the visitor pattern and produce a sequence of MIDI notes. The grammar we use is the same as what is given in Project 1 site.

2 Lexer

The lexer steps through the input text and produces a list of Token objects. A Token object is an immutable object whose members are an enumerated type, and a string value. The value member is not used for most tokens, for example, *BeginChord*. Tokens such as *BaseNote* would have a value such as "1/4". A full list of token types is available in `docs/tokens.txt`.

3 Parser

First, the parser will create a Piece object. The parser will store the NumberField, TitleField, ComposerField, DefaultLengthField, MeterField, TempoField, VoiceField, and KeyField (see docs/tokens.txt file) tokens (from the first part of the abc file, the header) into the appropriate instance variables of the Piece class (repeats, composer, key, defaultLength, meter, tempo, title, index, and voices variables see AST.txt file).

The parser must read the tokens in the order described previously. If the tokens are not given in the correct order, an error message must be sent. When the parser reads a BeginMajorSection token, it will create an empty Measure object with the voice given from the voices variable from the Piece object. It will read tokens corresponding to notes, rests, chords, accidentals, and tuplets until it reads an EndMajorSection token and fill the Measure object with the values from those tokens (in the order given). If a BarLine, BeginMajorSection, or BeginRepeat token is not after the EndMajorSection token, the parser will send an error message. The parser will also add up the duration of the notes found within the measure object to determine if the duration of the Measure objects matches the meter variable in the Piece object.

When the parser reads a BeginRepeat token, it will create a Repeat object. It will fill the Repeat object with Measure objects (up to two Measure objects if FirstRepeat and SecondRepeat tokens are read) until the EndRepeat token is read. If a BarLine, BeginRepeat, or BeginMajorSection is not read after the EndRepeat token, the parser will send an error message. When the parser reads a BeginDoublet, BeginTriplet, or BeginQuadruplet token it will create an empty Tuplet object. It will fill the Tuplet object with the next chord it reads next. If it does not read a chord next, the parser will output an error message. When the parser reads a BeginChord token, the parser will create an empty Chord object. The parser will fill the Chord object with the next note tokens it reads. If it does not read a note token (or any token that represent accidentals or changing the octave), then the parser will output an error message.

When the parser reads a BaseNote token it will create a Note object. If there is no NoteLength token immediately after the BaseNote token, then the duration of the Note object will be set to the value of the Duration object that used the value of the DefaultLengthField token. Otherwise, the value of the NoteLength token will be used to create the Note object. If the parser reads a Rest token, a Note object will be created with isRest set to True and duration set to either DefaultLengthField or the value of the next NoteLength token. If the parser reads a Sharp, DoubleSharp, Flat, DoubleFlat, Natural, OctaveUp, or OctaveDown token immediately after the BaseNote token (as long as the BaseNote is not a rest), then it will create a Note object with the octave and accidental set accordingly. Otherwise, the Note object will be created without an accidental and octave change.

4 Player

The player will walk through the nodes of the AST produced by the Parser using the visitor pattern. Every `Note` encountered will be mapped to a MIDI note, and will update the position into the song. Objects encountered such as `Tuplet`s and `Chord` will affect how the player calculates the duration and current position into the song. Each voice will keep track of independent positions into the song, so that each voice's measures will play in parallel.

5 Testing

The Lexer and Parser will be expected to have possibly invalid inputs. The Lexer will be tested with both valid and invalid ABC files. The Parser will be be tested using lists of both valid and invalid tokens. The Player will be tested with ASTs that exhaustively test every musical structure that affects how the music should be interpreted, such as Chords, Tuplet, Voice changes, Repeats, etc.... Diony will write tests for the Parser and Player, Chau for the Lexer and Parser, and Kharis for the Lexer and Player. This will improve the quality of the tests by making sure that the tests are not written with a specific bias or implementation in mind.