Design Document

Group Members

Will Drevo (drevo)
Jeremy Sharpe (jesharpe)
Daniel Mendelsohn (dmendels)

Grammar

We decided to use the EBNF grammar, as described here:
http://www.mit.edu/~6.005/fa12/projects/abcPlayer/assignment_files/abc_subset_bnf.txt

The only modification to that grammar that we made is that tuplets can **only** of size 2, 3, or 4, and Nth repeats can only be of the form "[1" or "[2". The EBNF description allows for any digits in those two cases.

Tokens for Lexing:

For Header:

- INDEX – of the form (X: integer \n)
- TITLE – of the form (T: text \n)
- COMPOSER – of the form (C: text \n)
- LENGTH – of the form (L: fraction \n)
- METER – equal to "C" or "C|" or of the form (M: fraction \n)
- TEMPO – of the form (Q: integer \n)
- VOICE – of the form (V: text \n)
- KEY – of the form (K: validKeyString \n)

For Body:

- OCTAVE - some number of one type of octave character (e.g. ')
- BASENOTE - a letter that can be a note (e.g. A-Ga-g]
- ACCIDENTAL - (one of "_", "__", "^", "^^", "=")
- REST - z
- DURATION - in regex its DIGIT* (SLASH DIGIT*)?
- BAR – (one of "|", "||", "|]", ":|", "|:")
- NTH_REPEAT - (one of "[1" or "[2")
- TUPLE_START – (one of "(2" or "(3" or "(4"
- CHORD_START – just an open bracket
- CHORD_END – just a close bracket
- VOICE_BODY – an in-body voice field, such as "V: upper\n"
- COMMENT – a comment of the form (% text \n)

Parsing Strategy
1) Parse header
2) Parse body
- whenever we hit a token that can start a measure, we descend into a routine called parseMeasure(). Similarly, within parseMeasure(), we discover when we're hitting the beginning of a note or chord we descend into those subroutines and so forth.

Class hierarchy (theme: "cascading interfaces")
Interface HighElement (these are at the "measure, bar" level
- extended by Element
- implemented by Measure
- implemented by NthRepeatElement
- implemented by BarlineElement

Interface Element (these are entirely within measures)
- extends HighElement
- extended by SyncElement
- implemented by TupletElement

Interface SyncElement (these are things that start at one specific time)
- extends Element
- implemented by ChordElement
- implemented by NoteElement
- implemented by RestElement

One major function of the interfaces is to group classes

Data Structures:
```
class Song {
    Header header;
    List<Voice> voices;
}

class Header {
    // fields like beats per minute, etc go here
}

class Voice {
    List<HighElement> measures;
```

```
}

class Measure extends HighElement {
        List<Element> elements;
}

class NthRepeatElement extends HighElement {
        //represents Nth repeat
}

class BarlineElement extends HighElement {
        //represents some sort of bar line
}

class TupletElement implements Element
        List<SyncElement> elements;
}

class ChordElement implements SyncElement {
        List<NoteElement> notes;
}

class NoteElement implements SyncElement {
        //represents a single note
}

class RestElement implements SyncElement {
        //represtents a rest
}
```

We also created a useful PlayableNote class that represents a note ready to be loaded in a SequencePlayer.

Evaulation Strategy

1) Go through Song and expand out tuplets, making step 2 possible
2) Go through Song and determine necessary number of ticks
3) Expand repeats out so we're just evaluating a list of measures
4) Descend into each measure and compile a list of PlayableNote as we go

5) After iterating through whole piece, we load our PlayableNote objects into the SequencePlayer() and play().