Beathoven

Sona Roy sbr2146 (Manager)
Jake Kwon jk3655 & Ruonan Xu rx2135 (Language Gurus)
Rodrigo Manubens rsm2165 (System Architect / Musical Guru)
Eunice Kokor eek2138 (Tester)

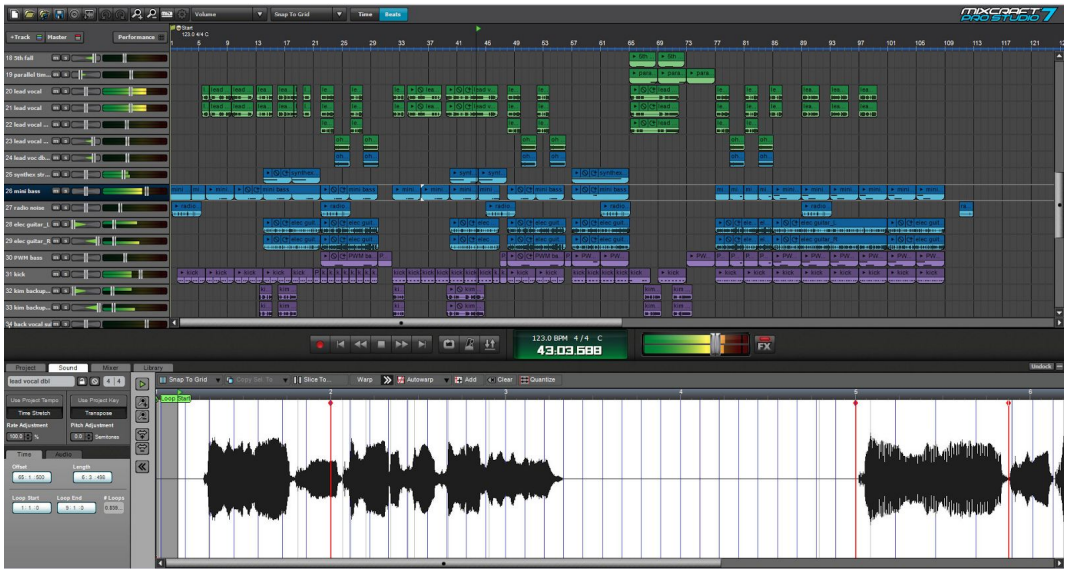
Background/Purpose

- Beathoven is a language built for MIDI/MusicXML generation so that anyone, even people who don't really know music, can create a song by putting together words that represent musical concepts into structures that look like actual music.
- Our language's output creates a MusicXML or MIDI file representing a music score. We chose this output because it can be imported into various music software programs, such as MuseScore (a free music composition software, which can generate score and play the notes). Additionally, we also chose it because its structure allowed us more flexibility in creating our language's basic functionality.
- One of our stretch goals with this language is to create a music file that contains both melody and lyrics. Lyrics will not have tone, but rather have a beat = best musical genre for this language will be rap.
- Another goal is to represent chords, notes, and improvisation such that different types of music creators (people who create music by relative pitch vs by absolute pitch) can have more flexibility.
- Another one of our goals with this language will be to easily generate "stacked" music scores, aka music that is played at the same time with the same key but that have different (ie polyphonic) melodies.

Goals and Features

Musical Production

- By writing a song's common elements with our language's syntax, functions and data structures, a user can digitally encode a composition of a song. Additionally our language can generate improvised song snippets by randomly generating melodies based off of parameters that describe musical patterns.



Syntax

Primitive Data Types	Description
pitch	**Absolute Pitch:** `C `G5 `F4# **Pitch Relative to Key:** `1 `5 `1+ `6-
duration	**e q h w** **1/4 1/8 1/16 1/2 1 2 3/4 3/8 7/16**
int	Normal integers
rest	Silence pitch type **Absolute Pitch:** ‘s **Pitch Relative to Key:** 0

Supported Data Types	Description
Note A Note is defined as a pitch, duration tuple. The default pitch of a note is “C” and its default duration is a quarter.	Note fSharp = `F4#; /*F4# pitch, 1/4 duration*/ Note fa = `4:1/4; Note cWhole = :w; /*C pitch, 1 duration */ Note note = :1/4+1/8; /*C pitch, 3/8 duration*/
Chord A Chord is a group of notes. The default inversion for chords is root inversion. A chord can also be described with a shorthand notation($) that details the main note of the chord and the bass note of the chord.	Chord cMajor1 = `C&`E&`G; Chord cMajor2 = cMajor1 & `C5; Chord chord = note1 & note2 & note3; Chord cMajor3 = `cMajor$ ’G4; /* Short hand for C chord Starting on G(ie third inversion) */

Modules	Example
Seq A Seq is made up of Notes or Chords. Seq will automatically group Notes and Chords into bars according to its attributes. The attribute, time signature, is inherited from Score if not specified.	Seq seq1 = [`1`1`5`5 `6 `6 `5:3/8]<4/4>; Score.setTimeSignature(4/4); Seq seq2 = `[4 4 3 3 2 2 1:3/8]; /* in shorthand */ Seq seq3 = [`5`5 seq2.(:6)];

Seq is mutable and its elements and subsequence can be easily accessed like in python. Seq cannot have nested structures. A seq within another Seq will be flattened before assignment. Sequences also have a measure_length attribute. Measure length attribute covers how many score defined measures each sequence covers. Additionally the sequences module has built in functions to modify sequences programmatically. For example, some of these functions allow the user to algorithmically modify and "improvise" new sequences based off of the last notes of a sequence. Other functions allow the addition of notes on two sequences simultaneously by appending notes that follow a specific musical motion	Seq seqConcat = [seq1 seq2 seq3 seq3 seq1 seq2]; Seq.add_note([input_sequence], [semitone/tone], [higher/lower/equal]) Seq.contrary_motion([first_sequence], [second_sequence], [up/down]) Seq phrase11 = `[3 2 1 2] Rhythm beats `[3 3 3]; Seq phrase2 = [phrase11.(0:-1) `3 `1];
Part A Part is a container for sequences that specifies what sequences should be played (and when they should be played) throughout a score. If a part has no specified sequences for a specific measure, a part autofills those measures with rest sequences until all of the measures in the part are covered.	Part part_1 = [seq_1*1, seq_2*2] -> [seq_1, seq_2, rest_seq]
Score A Score is a container of parts with unique attributes (such as meter, key and total number of measures) that help describe the musical piece it represents.	Score.parts < part1 < part2 Score.setKeySignature(`Ab)

Operator	Description
:	Indicates a duration
+ / -	Octave /
`	Indicates a note
&	Chord
* int	Concatenates a sequence or note int times.
<	Concatenates parts into a Score-ready object

;	End an assignment command

Example Code Usage

Example 1 - General Features

/* mary had a little lamb main https://musescore.com/user/73888/scores/92396 */

```
1  Score.setTimeSignature(4/4); /* global variable */
2  Seq beats = :[q q h];
3  Seq phrase11 = `[3 2 1 2] Rhythm beats `[3 3 3];
4  Seq phrase12 = `[2 2 2]:beats ‘[3 5 5]:beats;
5  Seq phrase2 = [phrase11.(0:-1) `3 `1];
6  Seq phrase22 = phrase11 phrase12 phrase2 `[2 2 3 2 ] `1:w;
7  Part maryHasALittleLamb = phrase22;
8  Score.parts < maryHasALittleLamb;
9  Render maryHasALittleLamb mLamb.py mLamb.xml;
```

Line 1: This uses the Java-like object construct to set a global variable Score’s time signature. setTimeSignature is a method within the Score module. Because every of our programs will be creating a MusicXML file that is similar to real sheet music, that line is necessary to set our time signature for the song.

Line 2: We define a basic beat Seq initialized with the default pitch value of C.

Line 3: This starts off with a sequence of notes, using a shorthand way of representing a group of notes: `[3 2 1 2] is short for [`3 `2 `1 `2]:[q q q q]. After that shortened sequence, Line 3 is adding on another sequence in which a function called Rhythm adds our beat Seq to a musical phrase with three 3 pitches. If we were to write that line completely it would be [‘3 ‘2 ‘1 ‘2 ‘3 ‘3 ‘3]:[q q q q q q h] so our shorthand allows for some simplification & less repetition.

Line 4: This also uses shorthand to cast the beats onto the notes.

Line 5: we borrow from python's list comprehension features and copy all notes of phrase11 except the last note and add two more. Seq does not have nested structure. Putting seq next to another seq concatenate them. In [phrase11.(0:-1) `3 `1], sub sequence phrase11.(0:-1) gets concatenated into the list.

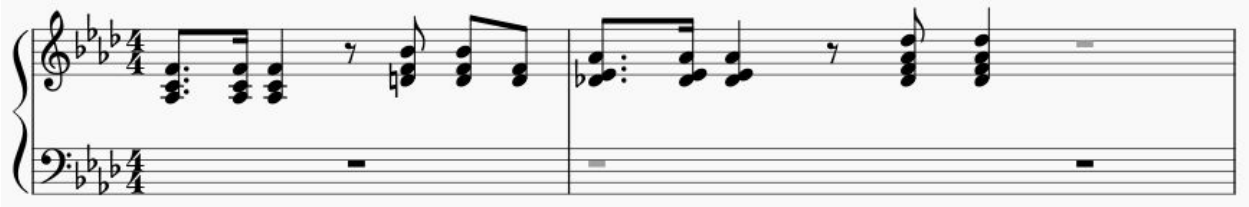
Line 6: This line is another Seq combining previously set up sequences and appending another sequence to the end.

Line 7: Adding the entire sequence just created to a Part.

Line 8: Adding the Part to the Score in order.

Line 9: We use Render in which the Part gets transformed into sheet music

Example 2 - Octaves + Chords Features



```
1 Score.setKeySignature(`Ab);
2 Score.setTimeSignature(4/4);
3 Seq beats1 = :[3/8 1/8 q e e e e];
4 Seq beats2 = beats1.(0:-2) :q;
5 Seq melody1 = `[ Fm$Ab3 Fm$Ab3 Fm$Ab3 0 BbM$D BbM$D BbM$D ]:beats1;
6 Chord Absus = `Db4 & `Eb4 & `Ab4;
7 Chord Dflat = `DbM$Db4 & Db5; /* this shorthand plus appended note creates a four note chord */
8 Seq melody2 = [ Absus Absus Absus `0 Dflat Dflat ]:beats2;
9 Seq smellsLikeTeenSpirit = melody1 melody2;
```

Line 1: This line sets the default key signature to A flat

Line 2: This line sets the default time signature to 4/4

Line 3: We define a basic beat Seq of 3/8, 1/8, quarter, eighth, eighth, eighth, and eighth initialized with the default pitch value of C.

Line 4: We define beats2 as beats1 from first index to third of last index. This function resembles that of Python.

Line 5: This starts off with a sequence of chords, using a shorthand way of representing a group of chords: `[ Fm$Ab3 Fm$Ab3 Fm$Ab3 0 BbM$D BbM$D BbM$D ]:beats1 is a short for `[Fm$Ab3 Fm$Ab3 Fm$Ab3 0 BbM$D BbM$D BbM$D]:[3/8 1/8 q e e e e]. [3/8 1/8 q e e e e]. Defining the elements of the Seq list, `Fm$Ab3 means a F minor chord made out of Ab3, C4, F4 notes. 0 means a rest. BbM$D means a B flat major chord made out of D4, F4, B flat 4 notes. Also, A letter followed by a number is an octave, which means, for example, C4 fourth octave of a C note.

Line 6: This line creates a chord with Db4, Eb4, Ab4 notes.

Line 7: This line creates a D flat major chord with D flat 5 note, which is Db4, F4, Ab4, and D flat 5.

Line 8: Creates sequence using Absus and Dflat chords, which we created in line 6 and line 7.

Line 9: Finish the song by concatenating melody1 and melody2, which we created in line4 and line8.

Example 3 - Automation Features with Cantus Firmus

```
1  Score.setTimeSignature(3/4);
2  Seq voice_1 = '[ 5 5 6];
3  Seq voice_2 = '[1 2 3];
4  Seq.add_note (voice_1, tone, lower)
5  Seq.add_note(voice_2, tone, equal);
5  Seq.parallel_motion(voice_1,voice_2, tone, down);
6  Seq.parallel_motion(voice_1,voice_2,tone, down);
7  Part part_1 = voice_1;
8  Part part_2  = voice_2;
9  Score.parts < part_1 < part_2;
10 Render Score cantus_f.py cantus_f.xml;
```

Line 1: setting time signature to the rhythm 3 quarter notes for the total phrase length

Line 2: setting sequence to pitches [5 5 6] with default duration

Line 3: setting sequence to pitches [1 2 3] with default duration

Line 4: appending the new note to sequence voice_1, with the tone desired and the lower frequency than the last note

Line 5: appending the new note to sequence voice_2, with the tone desired and an equal frequency to the last note

Line 6: Seq.parallel_motion, there are two stacked sequences, and when they move in parallel both sequences move down the same interval - for every call you append a note to each of the two sequences you have used in parallel

Line 7: setting the order of sequences such that voice_1 is played first

Line 8: setting the order of sequences such that voice_2 is played second

Line 9: inputting the parts in order into the score

Line 10: rendering the score into a python file (in order to create a MIDI file) and an .xml file

Example 4 - Control Flow

```
1  Seq Rhythm(Seq beats, Seq melody) = {  /* definition of the Rhythm function used in example 1 */
2          if (beats.length == 0) Exception("empty beats");
3          for (int i = 0, j = 0; j < melody.length; j = j + 1) {
4                    melody.(j).duration = beats.(i).duration;
5                    i = if (i + 1 < beats.length) {i+1} else {0};
6           }
7          return melody;
8  }
9
10  function increasePitch() {
11         Seq baseSequence = `[3 2 1 2] Rhythm beats `[3 3 3];
12  }
13
14  Score createScore(int totalMeasures, int totalParts) = {
15         Score score = new Score();
16         for(int i = 0; i < totalParts; i=i+1){
17                  score.addPart(Part(totalMeters));
18
19          return Score;
12  }
```