Music-mike

Introduction

Western music is usually notated on a five-line staff, on which **notes** are given a **duration** based on symbol type, and **pitch** based on location in the staff. Composers can use proprietary software such as Sibelius or Finale to manipulate a virtual five-line staff through mouse clicks or keyboard gestures. Fans of computer music might instead use music synthesis libraries to programmatically create music in languages such as C++, but such libraries can be unintuitive for musicians unfamiliar with signals and waves.

We propose Music-mike, a strongly typed, high-level, functional programming language, to give users an alternative option in music creation. Music-mike is designed for users to create music based on varied manipulations of short patterns. We owe this idea to Note Hashtag, a previous project completed in COMS W4115. However, unlike Note Hashtag, Music-mike is **modal** rather than **key-based**. Furthermore, lists - treated as the fundamental building block of music - are manipulated with special list operators (syntactic sugar) which create an intuitive interface based on traditional staff notation.

Design Ethos

The most basic unit in music is a **note**, which can be decomposed into pitch and duration. A simple melody can thus be described as two lists: one list of pitches and another of durations. A **chord** is a collection of notes played simultaneously.

A set of pitches is defined as a **mode**. All modes are subsets of the chromatic scale, which contains all twelve pitch classes used in Western music. Most music constrains the pitches of its notes to a small set of familiar modes, such as the major and minor scales. The sound of a **chord** is very much dependent on the **mode** that its notes come from.

Music-mike is based on the following observations regarding Western music: one, that Western music is fundamentally **chordal** and **modal**. Two, that Western music is repetitive and manipulative: simple building blocks of music are modified, then repeated multiple times in a piece. Finally, and most importantly, that these simple building blocks can described using lists and altered using a functional paradigm.

Features

In programming speak: - Functional Programming (Maps and stuff) - Immutable Everything - Intuitive List Defintion and Manipulation - Sweet Syntactic Sugar

(Don't worry, these features will be outlined in the example program.)

In music speak: - Phrases can be: augmented, diminished, transposed, or even mapped to different modes! - Melodies can be placed on a timeframe allowing for easy polyphonic composition.

Example Program

The bread and butter of Music-mike are the unique list constructors. Lists can only take one type in Music-mike, similar to OCaml.

Square brackets construct a "normal" list. Double square brackets [[]] are designed for convenient creation of rhythm lists and take expressions that are converted to floats. Angular brackets < > facilitate pitch list creation. These constructors manipulate their contents in subtle ways that are explained in the comments below.

```
// Happy Birthday. Oh by the way, this is a single line comment. Nested comments will
// not be supported.

mode = MAJOR;
// Alternatively could define as major=[1 3 5 6 8 10 12];

rhythm = [[ 8o 16 4 4 4 2 8o 16 4 4 4 2 1 ]];
// o is an operator that “dots” a rhythm unit.
// The actual rhythm here is: Dotted Eighth Note - Sixteenth Note - Quarter Note ... etc.
// So [[ ]] constructor takes int-expressions and takes their inverse.

pitches = <1&5 5 6 5 ^1 7 5 5 6 5 ^2 ^1 0>;
// The prefix operator ^ moves up pitch by an octave.
// The infix operator & creates a chord. Actually, every note is a chord of one for type consistency.
// Zero notates a rest.

melody = wrap(mode, pitches, rhythm);
// Function arguments in parentheses and separated by commas, “C4” is string representing start note.

fun Ascend x = x#;
new_pitches = map(Ascend, pitches);
new_melody = wrap(mode, new_pitches, rhythm);
//now our tune is in key of C# major

fun Augment x = x*2;
new_rhythm = map(Augment, rhythm);
more_melody = wrap(mode, pitches, new_rhythm);
//now twice as long as the original melody

timeline = plot(more_melody, 0);
// This assigns a start time to the melody by placing it in a timeline object.
synth(timeline);
// This generates .wav file.
```

Team Members

Name	UNI	Role
Husam Abdul-Kafi	hsa2136	Systems Architect
Mounika Bodapati	lmb2254	Manager

Name	UNI	Role
Kaitlin Pet	khp210	Tester
Harvey Wu	hw2473	Language Guru