Musical Language "Melody" -- Proposal

Jingsi LI, jl4165
Shuo YANG, sy2515
Tong GE, tg2473

9/25/2013

Overview
As we all know, musical language has gradually gained popularity among both music amatuers and professional musicians. We call our lovely language as "**Melody**" which is an efficient tool to compose and edit beautiful electronic music. We three guys are all music fans with basic music knowledge, so it just matches us so well to select this topic as our course project for Programming Languages and Translators. This music programming language allows us to create music by simply coding and playing on the computer keyboards and trying different music effects you just create. It will be a fantastic thing to join the music creation and code at the same time, and enjoy what such a cool thing can give us.. By manipulating music elements, including notes, bars, tracks, tempos and rhythms, users can create original melodies and play it directly.

Purpose
Hey guys! Do you enjoy coding and music at the same time? Cool! because it is the right time for you to use our Melody. Actually, our musical language makes your music creating free and easy. And you can truly enjoy music via coding! Of course, Melody is designed for electronic music creation, and even for the synthetic instrumental music. Most importantly, we obey the rules of music creation, and makes our coding as the same as real music creating process. Before designing our language syntax and programming process, we thoroughly thought what the real music creation process is, and what coding process or pattern we should obey. Then we subtly designed our language. Also, you can define different tracks to represent different music tracks in real music world, such as the main melody, the drums, the guitar and the bass. You know that, the real cool thing is that you can not only create melody, but also create melody with different timbres. Thus you can truly create a rich, real, beautiful music! Last but not least, you can play your own music and save it as ".mp3" file. Think of that, it is really cool things when your friends are amazed at what you just created!

Features
- Define key tuning used for different sessions in your music
- Use different tracks to create your music, which includes main melody, drums, bass and so on
- Use bars to construct a track, in which you can include single note editting and module inputting
- Increase or decrease steps for a bar or a whole track, or repeat your bar or track as well
- Compiling your code to play or save as standard format
- Coding in the process just as you are in real music creation
- Enjoy music and coding at the same time!:)

Syntax

Type

Type	**Description**
Int	used to represent attributes like speed
String	used to represent compositions in chord, or timbre as an attribute of a track
Note	has value defining the pitch e.g. Note note1 = C4 (absolute pitch) Note note2 = 500 (frequency) Note note3 = $#1 (relative pitch, “$” is the identifier for relative pitch)
Bar (String chord)	unit to compose track has chord(String) as attribute, none chord if not defined Bar(C&E&bB) bar=... two ways to define a bar: 1. By enumerating every note inside --[(Note n, Int notetype), ...] 2. By applying an pre-defined rhythm and put notes into corresponding positions --{(Rhythm r), ((Note n), …)}
Rhythm	can be pre-defined and applied to bar to simplify its definition e.g. [down, up*4, down, up*4]
Track (String timbre, Int length, Tempo tempo, String key, Int speed)	made up of bars can be played together with other tracks has attributes of timbre (e.g. violin, piano), length (number of bars inside), tempo (e.g. 2/4), key (e.g. C), and speed (e.g. 60 bars/min) e.g. Track(violin, 5, 2/4, C, 60) track ={bar1, bar2, bar3, bar4, bar5}
Tempo	the tempo of track
Melody	the composed melody consists of many tracks e.g. Melody melody1=track1&track2&track3

Operators

Operators	Description
=	Variable assignment
*	Repeat an element several times
&	Snythetize tracks into a melody
()	Bracket a pair or a group of elements
{}	Used to apply pre-defined rhythm patterns
/* */	Block comments
//	Line comments
;	End of statement

Keywords

Keyword	Description
Int	“Int” primitive
Note	“Note” primitive
Bar	“Bar” primitive
Rhythm	“Rhythm” primitive
Track	“Track” primitive
Tempo	“Tempo” primitive
Melody	“Melody” primitive
Define	Define a constant variable
up	Describe the rhythm
down	Describe the rhythm

Standard Functions

Function	Description
power(Int start, Int end, String type)	Control the power of one or more consecutive bars according to the specific type
play(Melody melody)	Play one track or play multiple tracks simutaneously
save(Melody melody, String path, String filename	Save the melody composed of one or more tracks under the specific path
toneUp(Int times)	Rise the tone by half the degree several times
toneDown(Int times)	Fall the tone by half the degree several times

Sample codes

```
//define constants
define key1 C;
define key2 D;

//define notes
Note note1= C4;
Note note2= 1#;
Note note3= 120;
Note note4= B6;
Note note5= 500;

//define rhythm
Rhythm rhythm1=[down, up*2, down, up*2];//so there should be 1+2*1+1+2*1=6 notes

//map notes according to rhythm into a bar
Bar(C&E&bB) bar1={(rhythm1), (note1, note2*2, note1*2, note 2)};
Bar() bar2=[(note1, 8), (note2, 8), (note3, 8), (note4, 8)]; //four eighth notes in this bar
Bar() bar3=bar2.toneUp(1); //rise the tone of bar2 by half degree to create bar3

//set bars into tracks
Track(violin, 4, 2/4, C, 60) track1={bar1, bar2, bar1, bar2};
Track(piano, 4, 2/4, C, 60) track2={bar2*3, bar3};
```

```
//set power of tracks
track1.power(0,1,mp); /*set mezzo piano (from Italian, means “median weak”) as the power of
the first and second bars in this track*/

//define melody with many tracks played together
Melody melody1=track1&track2

//play the composed melody
play(melody1);

//save the composed melody
save(melody1, “D:/files”, “myFirstMelody”);
```