M(usic)arma(dillo)(koo)lade:

A Music Creation Language

Team:

Cathy Jin (ckj2111) - Tester
Savvas Petridis (sdp2137) - Language Guru
Raphael Norwitz (rsn2117) - System Architect
Uzo Amuzie (ua2144) - Manager

September 30, 2015

Introduction:

Marmalade is a music writing and playback language that will focus on creating songs. A user can use Marmalade to create a complex beat by defining a sequence of notes for each particular component of a drum kit and then splicing this sequence into measures. But the fun does not stop there! Other instruments, such as a guitar or piano can be layered over the beat. Sequences of single notes or chords can accompany the beat, thereby adding a melodic component to your unique composition. At the same time, Marmalade allows users to bypass creating every single aspect of their work, by generating chord progressions or rhythms algorithmically through stochastic models the user defines. Of course some basic functions, like altering the volume of certain notes will be included as well, allowing a wide range of customization.

What really sets Marmalade apart from other musical programming languages is its adherence to tried and true programmatic paradigms, modeling the syntax and structure off of Java/C. The beauty of a language like Marmalade is its ability to combine the idea of composition of music into the data structures we commonly encounter when programming in an object-oriented fashion.Thus ideally someone with a musical background could learn object-oriented programming properly by using Marmalade to make some jams.

If it's a reasonable proposition, we'd like to write the Marmalade compiler for the Raspberry Pi so that younger kids who like music but don't have access to full computers can get use out of it too (and for the added challenge of making it space efficient).

Language Overview:

Built-in Types

`int`	an integer value
`quarter`	represents a quarter note in a measure
`half`	represents a half note in a measure
`whole`	represents a whole note in a measure

`string`	data representing a simple text word
`song`	data representing a full composition
`pattern`	data representing a sequence of notes for a particular instrument in a song
`[]`	array data type
`time_sig`	The specified time signature of a song (used by the compiler to check measures
`instrument_library`	
`measure_seq`	

Control Flow

- while loops
- for loops
- if/else statements

Operators

{}	defines the scope of the blocks of code	**;**	indicates end of each line in the code
=	assigns value of right side operand to left side operand	+	add operands on the left and right side
-	subtract right side operand from left side operand	*	multiple left and right side operand
/	divide left side operand by right side operand	&&	logic AND
||	logic OR		

Keywords

- import
- Pattern
- song

Comments

- /* */ Multi-line comments
- // Single line comment

Whitespace

Marmalade is not sensitive to whitespace, since it keeps track of scope through the use of braces.

Defining Variables & Functions

- func function_name(arg1, arg2, arg3, ...)

Sample Program:

This program defines the Pattern (class) mary_little_lamb, or the notes to the song mary has a little lamb expressed in measures, and then plays the notes mapped to a variety of instruments. In main, you can see the Patterns rearranged algorithmically (using the transpose method).

```
/**
 * imports for instrument/sound libraries will simply take an absolute path
 * to the library files so as to give the user experience with system paths
 **/

import ../mp3/piano.library
import ../mp3/guitar.library

pattern Mary_little_lamb
{
	time_sig time;
	instrument_library instrument;
	measure_seq ms = [];

	Mary_little_lamb(instrument_library instr)
	{
		this.instrument = instr;
		int i = 0;

		/* variable declaration */
		int repeats = 4;
		/* a while loop, in action */


		//OPTION 1: NOTE-BY-NOTE
		while (i < repeats) {
			curr_measure1 = [];
			curr_measure1.push(instr.quarter(44));	// E
			curr_measure1.push(instr.quarter(42));	// D
			curr_measure1.push(instr.quarter(40));	// C
				// Middle C is labeled as '40'
			curr_measure1.push(instr.quarter(42));	// D
			measure_seq.push(curr_measure1);

			curr_measure2 = [];
			curr_measure1.push(instr.quarter(44));	// E
			curr_measure1.push(instr.quarter(44));	// E
			curr_measure1.push(instr.half(44));  // E
			measure_seq.push(curr_measure2);

			curr_measure3 = [];
			curr_measure1.push(instr.quarter(42));	// D
			curr_measure1.push(instr.quarter(42));	// D
			curr_measure1.push(instr.half(42));  // D
			measure_seq.push(curr_measure3);
```

```
        curr_measure4 = [];
        curr_measure1.push(instr.quarter(44));     // E
        curr_measure1.push(instr.quarter(47));     // G
        curr_measure1.push(instr.half(47));  // G
        measure_seq.push(curr_measure4);

        curr_measure5 = [];
        curr_measure1.push(instr.quarter(44));     // E
        curr_measure1.push(instr.quarter(42));     // D
        curr_measure1.push(instr.quarter(40));     // C
        curr_measure1.push(instr.quarter(42));     // D
        measure_seq.push(curr_measure5);

        curr_measure6 = [];
        curr_measure1.push(instr.quarter(44));     // E
        curr_measure1.push(instr.quarter(44));     // E
        curr_measure1.push(instr.quarter(44));     // E
        curr_measure1.push(instr.quarter(44));     // E
        measure_seq.push(curr_measure6);

        curr_measure7 = [];
        curr_measure1.push(instr.quarter(42));     // D
        curr_measure1.push(instr.quarter(42));     // D
        curr_measure1.push(instr.quarter(44));     // E
        curr_measure1.push(instr.quarter(42));     // D
        measure_seq.push(curr_measure7);

        curr_measure8 = [];
        curr_measure1.push(instr.whole(40));       // C
        measure_seq.push(curr_measure8);
        i = i + 1;


    //OPTION 2: BY MEASURE
        curr_measure1 = [];
        curr_measure1.push(instr.quarter(44), instr.quarter(42),
                           instr.quarter(40), instr.quarter(42)); // E D C D
        measure_seq.push(curr_measure1);

        curr_measure2 = [];
        curr_measure2.push(instr.quarter(44), instr.quarter(44),
                           instr.half(44)); // E E E
        measure_seq.push(curr_measure2);

        curr_measure3 = [];
        curr_measure3.push(instr.quarter(42), instr.quarter(42),
                           instr.half(42)); // D D D
        measure_seq.push(curr_measure3);

        curr_measure4 = [];
        curr_measure4.push(instr.quarter(44), instr.quarter(47),
                           instr.half(47)); // E G G
```

```
                measure_seq.push(curr_measure4);

                curr_measure5 = [];
                curr_measure5.push(instr.quarter(44), instr.quarter(42),
                                   instr.quarter(40), instr.quarter(42)); // E D C D
                measure_seq.push(curr_measure5);

                curr_measure6 = [];
                curr_measure6.push(instr.quarter(44), instr.quarter(44),
                                   instr.quarter(44), instr.quarter(44)); // E E E E
                measure_seq.push(curr_measure6);

                curr_measure7 = [];
                curr_measure7.push(instr.quarter(42), instr.quarter(42),
                                   instr.quarter(44), instr.quarter(42)); // D D E D
                measure_seq.push(curr_measure7);

                curr_measure8 = [];
                curr_measure8.push(instr.whole(40)); // C
                measure_seq.push(curr_measure8);
                i = i + 1;
            }
        }
}

void main() {

        /* declare song lamb */
        song lamb;

        /* set time signature of the song lamb to 4/4 */
        lamb.time_sig = '4-4';

        /* create two patterns for the song 'Mary Had a Little Lamb' */
        /* the first is the sequence of notes played by a piano, the second is the same
        sequence played by a guitar */
        pattern mary_piano = new Mary_little_lamb(piano);
        pattern mary_guitar = new Mary_little_lamb(guitar);

        /*
        * play is a function defined in the standard header of the language
        * It iterates through the measure_seq object, playing each note in the measure
        * It takes the starting measure as an explicit parameter
        */

        /* function 'transpose' is called - transposes the song from C Major to F Major */
        pattern mary_piano_trans = transpose(mary_piano, 5);

        lamb(mary_piano.play(0));
        lamb(mary_piano_trans.play(0));

        /* play guitar after 8 measures */
        lamb(mary_guitar.play(8));
```

```
        /* mp3_out generates mp3 for lamb song */
        lamb.mp3_out();

}

/**
 * This function transposes the song into a different key,
 * based on the input of steps given.
 **/

func pattern transpose(Pattern song, int steps) {
        int i = 0;
        while(i < song.measure_count) {
                int j = 0;
                while (j < measure_count[i].length) {
                        song.measures_seq[i][j] = song.measure_seq[i][j] + steps;
                }
        }
        return song;
}
```