Craig Darmetko
COMS w4115 Project Proposal

HOLLABACH

HOLLABACH will be a domain specific programming language for composing music. Developers will be able to design, play and reuse musical compositions using HOLLABACH's easy to use and expressive programming language. They can use it for designing whole compositions, backing rhythm loops, or partial melody components which can be easily added to other musical compositions without having to rewrite or re-record complicated progressions. The language will use an OCaml compiler to generate MIDI files, which can be played back on most devices. HOLLABACH will use the OCaml-Portmidi[1], a library for cross platform MIDI support in OCaml, to create and manipulate MIDI files. Because MIDI describes the structure and notes but not the timbre, HOLLABACH compositions are instrument independent. HOLLABACH will also include a robust set of standard library components that includes common chords and useful samples.

For example, HOLLABACH can be used by a DJ to create Samples, which are self contained and reusable music snippets[2]. The DJ can then easily create compositions that use the sample or distribute these samples for others to use. Once the composition is compiled, it can then be played on any MIDI enabled device.

Because HOLLABACH is representing musical compositions as code, musicians can utilize source code tools and paradigms, such as version control software and code reuse, for collaboration and increased productivity.

Here is an example Sample from a file called mad_groove.bach:

```
def SAMPLE rhythm{
	#time_sig=4/4
	measure{
		#schema: note/chord/sample name(octave, start beat, note length)
		A(3, 1, 1)
		chord C(1, 2, 2)
	}
	measure{
		…
	}
}
```

This code defines a Sample named rhythm and sets its time signature to 4/4 time. It has two measures. The first measure contains two pieces, a single A note that lasts for one beat on beat one of the measure and a C chord that lasts for two beats starting on beat 2.

Next, is a sample composition:

```
import stdmus
import mad_groove

def chord A#7{
    some notes here
}

def SONG{
        #BPM = 240
        #time_sig = 4/4
        FOR(i=2){
            measure{
                mad_groove.rhythm(1,1,1)

                #melody
                A(2,1,1)
                B(2,2,1)
                C(2,3,1)
            }
            measure{
                chord A#7(1, 1, 3)
                if(i=2){
                    …
                }
            }
        }
        #BPM = 120
        #time_sig = 3/4
        measure{
            …
        }

}
```

As you can see, we first import the stdmus, the standard library, and the previous samples in mad_groove.bach. A Sample does not have to be a single measure in length. Calling it in a measure triggers a start time, and it will play until completion. Next we create a custom chord called A#7. After, we start the actual composition with the declaration of 'SONG'. This is similar to a Java/C main method and is actually run to create a MIDI output file. In this composition, we first define the Beats Per Minute (BPM) and time signature. All measures after that define have these values. Next we enter a loop of two measures. In the first measure, we call the previous sample, rhythm, in beat 1 and a melody over the rhythm. In the second measure, we play the custom chord in beat 1. This loop repeats twice before continuing. After the loop, we change the BPM and time signature and play an additional measure.

[1] - https://github.com/aplusbi/ocaml-portmidi
[2] - http://en.wikipedia.org/wiki/Sampling_(music)