Proposal: Systematic Music Creation with Loopy

A DSL that provides tooling for electronic dance music creators

Edward Cai c4n0b@ugrad.cs.ubc.ca
Kwang Hee Park m8b9@ugrad.cs.ubc.ca
Shobhit Bhatia j9k0b@ugrad.cs.ubc.ca
Ro Lee n3l0b@ugrad.cs.ubc.ca
Benjamin Willox s2g1b@ugrad.cs.ubc.ca

ABSTRACT

Our proposal is a domain specific language, called Loopy, built for EDM (electronic dance music) creators. Loopy will be designed such that it provides solutions to the repetitive nature of EDM, such as loops and notes. This proposal will go through an introduction on the background of Loopy and outline our approaches in building it through the various stages of the project milestones.

1. INTRODUCTION

EDM (electronic dance music) has been around historically since the 1960s, but only ever reached popularity here in North America around the early 2010s after American music industries pushed to rebrand rave culture with EDM (Reynolds, 2012). Artists such as Hardwell, Skillrex, and Steve Aoki hit millions of views as EDM rose to popularity and eventually triggered Billboard to introduce a new EDM-focused Dance / Electronics Song Chart in 2013 (Pietroluongo, 2013).

At a time where EDM is expanding its territories and artists are experimenting tools to create music more efficiently, our team wanted to come up with a tool that can help artists easily create electronic dance music through programming. We, specifically, wanted to target the repetitive nature of EDM which is comprised of numerous loops and notes that elevate and diminish in both volume and pitch.

In our initial research, we could not find any suitable programming language that was specifically targeting our needs. So, we decided to create our own domain specific language, namely, Loopy.

2. OUR APPROACH

The 80% Background Research Report

To complete this milestone, we plan to research extensively on the functionalities and semantics of the *rsound* DrRacket library. We will selectively choose functionality that we think would benefit our EDM creation domain specific language.

We will create both an abstract and concrete syntax, in which music developers can model and play various types of EDM. From our initial research the main functionalities we will include in our syntax are loops and notes. Loops will be used to represent various electronic beats and notes will represent a specific sound at a set pitch and volume. We will also research and implement a parameter that will change the overall pitch and volume of all loops and notes that temporally follow it.

We will describe the abstract syntax that we create, the ins and outs of the concrete syntax, and a guide illustrating how music creators can benefit from our domain specific language. We will also include a sample program that when parsed and interpreted would play a song. We will describe how eager evaluation is crucial for our language's success and give a counter example showing how lazy evaluation would not work. We will focus our research on areas that will be useful to us, such as Racket's sound library *rsound* that can be used to play, save, and edit concurrent sounds. Researching this will allow us to design (and later implement) our language in such a way that content creators will benefit from it. We will also look at competing music creation languages and express why our language would be used over existing choices.

It is at this milestone that we will design the abstract and concrete syntax, and articulate why this music creation domain specific language has a purpose and audience.

The 90% Proof-Of-Concept

To achieve this milestone, we must first create a basic parser and interpreter. At this step we plan to lay out the outline for the language's design, breaking it down into specific tasks that we can fulfill.

To complete the proof-of-concept, we plan to build a basic parser and interpreter that can demonstrate the necessary components of our language; these are mainly loops and notes. We will include a small library of sample beats that users can loop and a small variety of notes they can play. Though this prototype will not be fully featured, it will communicate that the end product we are aiming for is possible. To parse user input and render musical sounds, we will utilize Racket's library *rsound*. We will also begin to look at different value types and parameter passing that will optimize our language.

Along with the prototype, we will present an updated plan detailing what is necessary to create a fully featured language (i.e. what the proof-of-concept language is missing). We plan to implement these features in our initial 90% DSL:

1. A parser that can recognize cases from our concrete syntax and desugar them into an abstract syntax that our interpreter can execute.
2. An interpreter that can read and execute abstract syntax that our parser generates.
3. The following functionality in our language:
    a. Loops – Sounds that can be repeated for a set amount of time.
    b. Notes – Single sounds that are repeated once. These sounds can have a set pitch or volume and users can load their own songs into the language by specifying file paths.
    c. Specifications – Commands which can change various aspects of all loops and notes that follow the said specification. These aspects include volume and pitch.

The 100% Final Project

To achieve this final project milestone, we will be expanding on the 90% project of creating music from a single beat loop and melody to multiple beats looping with staggered starts and ends for each loop along with a melody. To achieve this extension will require careful specification of new syntax that will allow music creators to easily insert different beat loops at various points in the whole song without requiring repetitive actions, as well as layering different looped beats on top of each other.

The Poster

Our poster will be mainly centred providing a live demo for potential users to create music with. We plan to have a laptop with basic sound files and loops loaded to provide a foundation for the users to demo the product with headphones. The poster itself will contain instructions on how to use the product as well as the evolution of prototypes of syntax.

3. STARTING POINTS

We have found several documentations and resources that are helpful throughout our project:

On DSL implementation using Racket

Racket Documentation. Programming Languages: Application and Interpretation, https://docs.racket-lang.org/plai/index.html

John Clements. Racket Documentation. RSound: A Sound Engine for Racket, https://docs.racket-lang.org/rsound/index.html

On Studying the Relationship between Music and Computer Science

Suzanne Muller. Computer Science and Music Technology, http://www.musicthinktank.com/mtt-open/computer-science-and-music-technology.html

On Observational, Exploratory Studies on Existing Music Specific Language

Ge Wang, et al. ChucK: A strongly Timed Computer Music Language, Stanford University, https://www.gewang.com/publish/files/2015-cmj-chuck.pdf

Stanford University, Introductory Video on ChucK Programming Language - ChucK: A Computer Music Programming Language, https://www.youtube.com/watch?v=2rpk461T6l4

4. SUMMARY

In a world where EDM is constantly increasing in popularity, we are building a domain-specific language called Loopy. It will help artists and enthusiasts make EDM using the already existing library in DrRacket called *rsound*. We will make a parser and an interpreter to provide functionality like loops, notes and specifications.

REFERENCES

Reynolds, Simon. "How Rave Music Conquered America." The Guardian, Guardian News and Media, 2 Aug. 2012, www.theguardian.com/music/2012/aug/02/how-rave-music-conquered-america.

Pietroluongo, Silvio. "New Dance/Electronic Songs Chart Launches With Will.i.am & Britney at No. 1." Billboard, Billboard, 21 Jan. 2013, www.billboard.com/articles/news/1510640/new-danceelectronic-songs-chart-launches-with-william-britney-at-no-1.