Halmstad's university

Advanced Oriented Object Programming

Sound editor framework

Authors:
Rémi Gourdon
Hichame Moriceau

Supervisor:
Dr. Veronica Gaspes

May 26, 2014

Contents

Introduction

Our framework offers to: synthesise, visualize and tweak sounds through different modifiers. It comes with built-in generators, views, filters and effects, but, according to the project specifications, we did out best to make room for extensions. Two implementations are provided, to demonstrate various ways of using it, and different manners of assembling its components.

The question of modularity was for us a top priority from day one. Indeed, audio creation is typically a domain where the possibilities are endless. The sound generators can range from basic signal generators to complex synthesizer, either mimicking real instruments or creating completely new sounds. The same goes for filters and effects, which can be as simple as low-pass and high-pass, delays, but also flangers, reverbs, etc. For these modules, that we called modifiers, we also quickly realized that the possibility of adding several of them to the same sound would really expand the possibilities.

Design

2.1 Framework

2.1.1 Modularity

The core of the framework is the class Sound. Everything depends on it, but we tried to limit the connections between the different parts of the framework, because we think it is generally a good practice. The model then is constructed in modules, that anyone can easily extends to its own needs.

Considering that we had to give the possiblity to derive both simple and more complex implementations from this framework, it was essential to keep the modules well separated. As a result, we came up with a design where the only mandatory components to get a working application are a Sound object, a Generator and a Player. It means that our framework could also be used as an external library, dedicated to the generation and playing of sounds. Indeed, the data model relies on a simple array of samples, which could very well be integrated in any application requiring sound synthetis. The generators and views by themselves could also be used as a third party library and integrated in other sound models.

2.1.2 Extensibility

When we started to think about the implementation, we wanted to give the user the ability to easily append new classes to any part of the framework. This is why we aimed for as much modularity as possible, and genericity for the superclasses and interfaces. On this road, the necessity of using good design practices, and patterns, appeared rapidly. We can see that we used the model/view/controller architecture, which is represented here with the Sound as the model, the View is obviously a view.

2.1.3 Flexibility

An objective was to come up with a structure allowing the user to build upon it both command line utilities or graphical interfaces. As the requirements were to provide a graphical user interface, we decided to design a flexible one. It means that a client application could very well be working fully from the command line, though we didn't provide any example of this kind, because it wasn't our goal. It could also be an hybrid, like the basic editor, which is taking its initial input from the console and then provides a graphical interface for further tweaking, playing and visualization of the signals. Finally it

can be fully graphical, as you can see in the matrix editor implementation, which provides a prebuild set of sounds, and the panels to adjusts their parameters further on.

2.1.4 Reliability

The Strategy pattern is used several times, for the choice of the signal, of the filter and the effect.

2.2 Basic Editor

2.3 Matrix Editor

The inspiration for the non-trivial application came from an embedded ActionScript application[1] allowing an easy creation of sound patterns.

[1] http://www.hisschemoller.com/2009/step-sequencer-drum-machine/

Testing

Some code we're proud of

Results

Sources