Răzvan-Vasile Suciu 927

Synthesizer

Description

My project for this course is a simple sound synthesizer which can be used to generate basic waveforms, process them, and apply effects to achieve and play using the desired sound.

Project repository: https://github.com/razvan2na/Synth

Software requirements

To build the project, the following frameworks and libraries are needed:

.NET Core 3.1 : https://dotnet.microsoft.com/download

NAudio 1.10.0 : https://github.com/naudio/NAudio or through NuGet

To run the synthesizer, only .NET Core is required, as NAudio is included.

Usage

Once the project is built, the synthesizer can be opened like a normal Windows application. It includes two oscillators, an amplitude envelope, a filter, two effects (tremolo and delay), and an on-screen keyboard. Each of these modules have different parameters which can be modified to shape the sound.

The oscillators provide the 4 basic waveforms: sine, square, sawtooth and triangle. The signals generated by the oscillators then pass through an envelope to shape the attack, decay, sustain and release of their amplitude. Then, effects

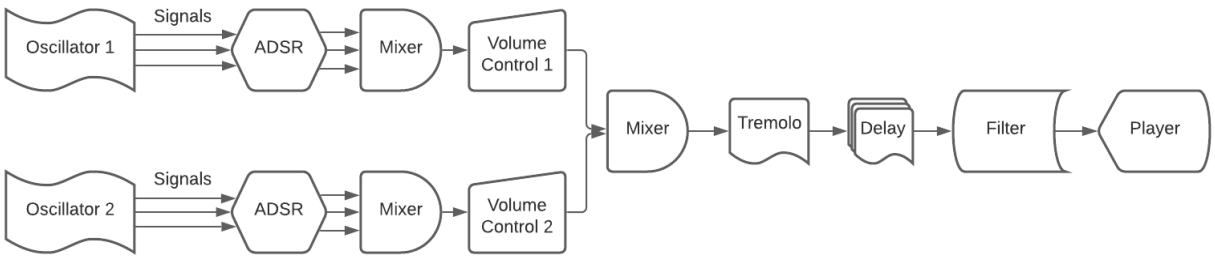
may be applied. Tremolo oscillates the signals' maximum amplitude at a low frequency, creating a "wobble" effect, while the delay simply repeats the signals at decreasing amplitudes. In the end, a filter may be applied to cut some undesired frequencies out of the mix.

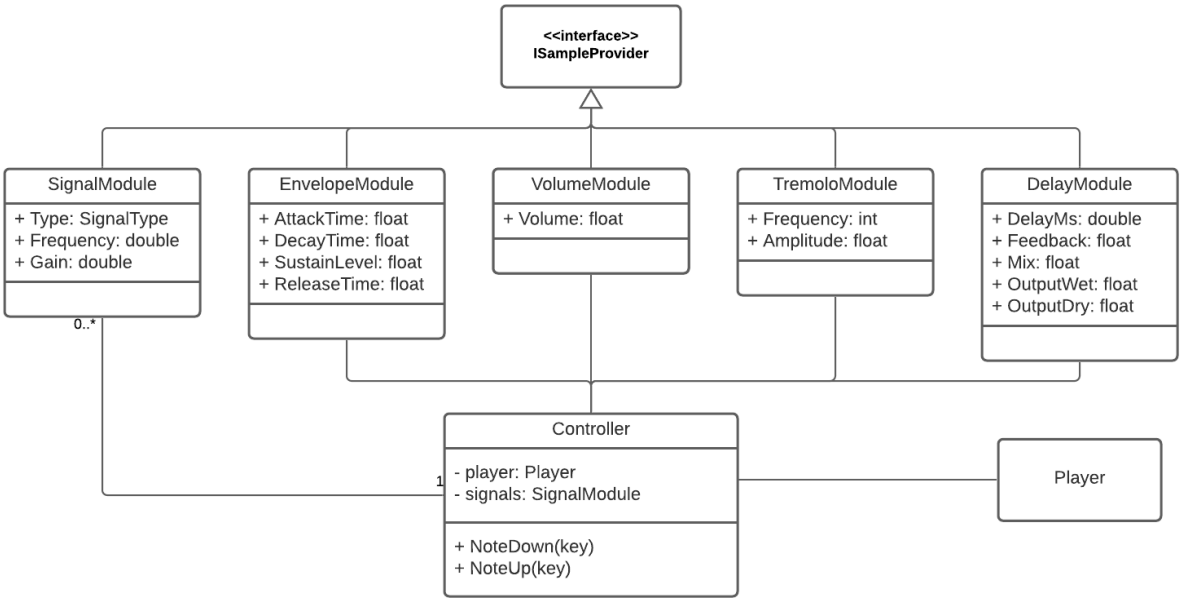
To play using the sound, simply use the on-screen keyboard or the computer's keyboard.

Implementation

The project's architecture is inspired by modular synthesizers, where modules can easily be added to or removed from the signal chain.

In this case, the signal chain is fixed, but modules can be enabled or disabled at will.





ISampleProvider, *Mixer* and *Player* classes are provided by NAudio.

The *Controller* class contains the signal chain shown in the first diagram. It also controls the signal creation and destruction process. When a key is pressed, a new signal is created, which is then processed through the chain and played. When a key is released, it signals the envelopes to start releasing the signal.

Each module implements the *ISampleProvider* interface. A module takes some samples as an input from another module (except *SignalModule*) and processes them in the **Read** method.

SignalModule creates the samples of a waveform with a given type (sine, square, triangle, sawtooth) and frequency.

EnvelopeModule gates a signal, passing it through 4 states (attack, decay, sustain, release). In the first one, the **attack** state, the amplitude rises from 0 to the signal's maximum amplitude within a given time. Then, the amplitude **decays** to the **sustain** level. When the note is released, the signal is also **released,** and the amplitude eventually reaches 0 again.

VolumeModule multiplies the input samples by a number between 0 and 1, consequently modifying the amplitude of the signal.

TremoloModule generates a low frequency signal, which is then used to “wobble” the original signal’s amplitude.

DelayModule keeps samples that pass through it in a buffer, then adds those samples later in the signal at lower amplitudes, creating a delay effect.

FilterModule implements a biquad filter. Passing samples through a biquadratic formula with certain parameters, undesired frequencies are cut out of a signal.