Road to RubyKaigi: Making Tinny Chiptunes with Ruby

@makicamel

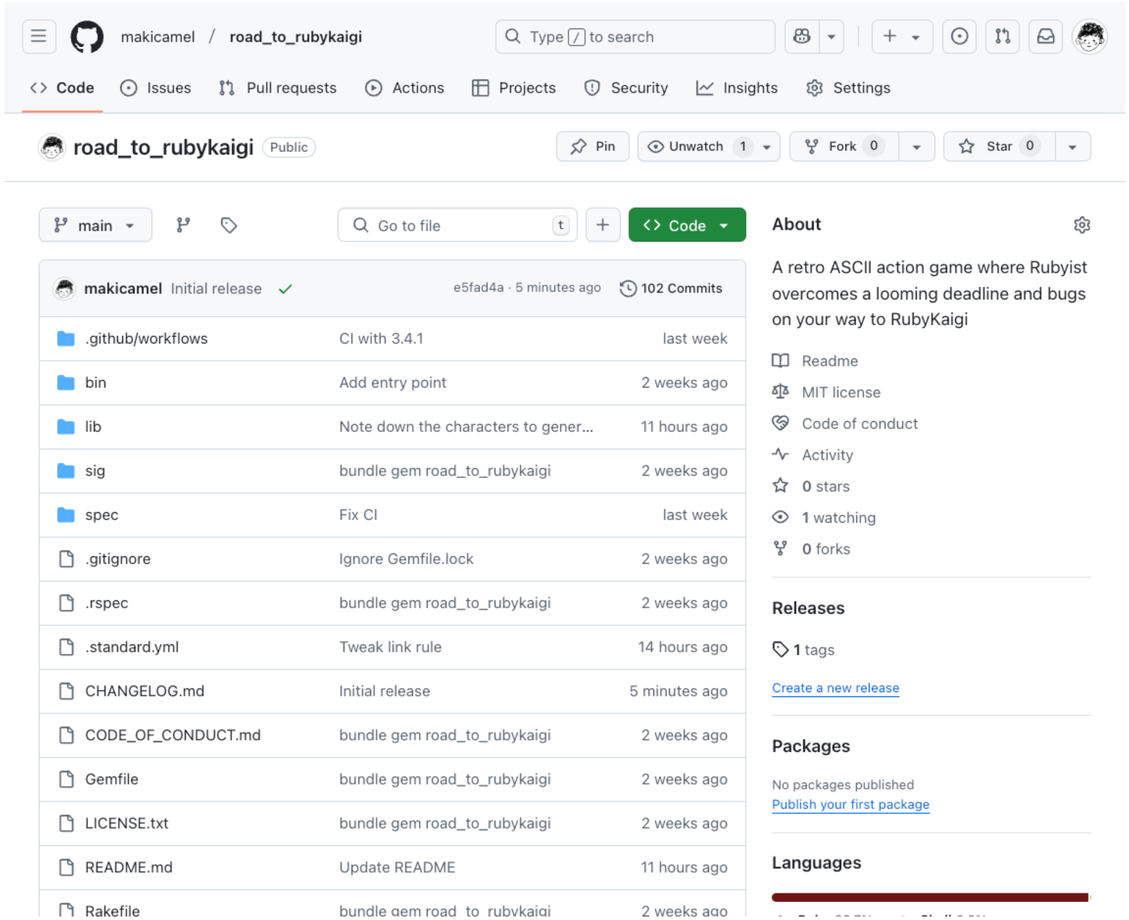
RubyKaigi 2025

2025.04.17

Self Introduction

- @makicamel / Maki Kawahara
- Loves Ruby 💎, beer 🍻, and alcoholic drinks
- AndPad Inc.
- Creater of "Road to RubyKaigi"

Road to RubyKaigi



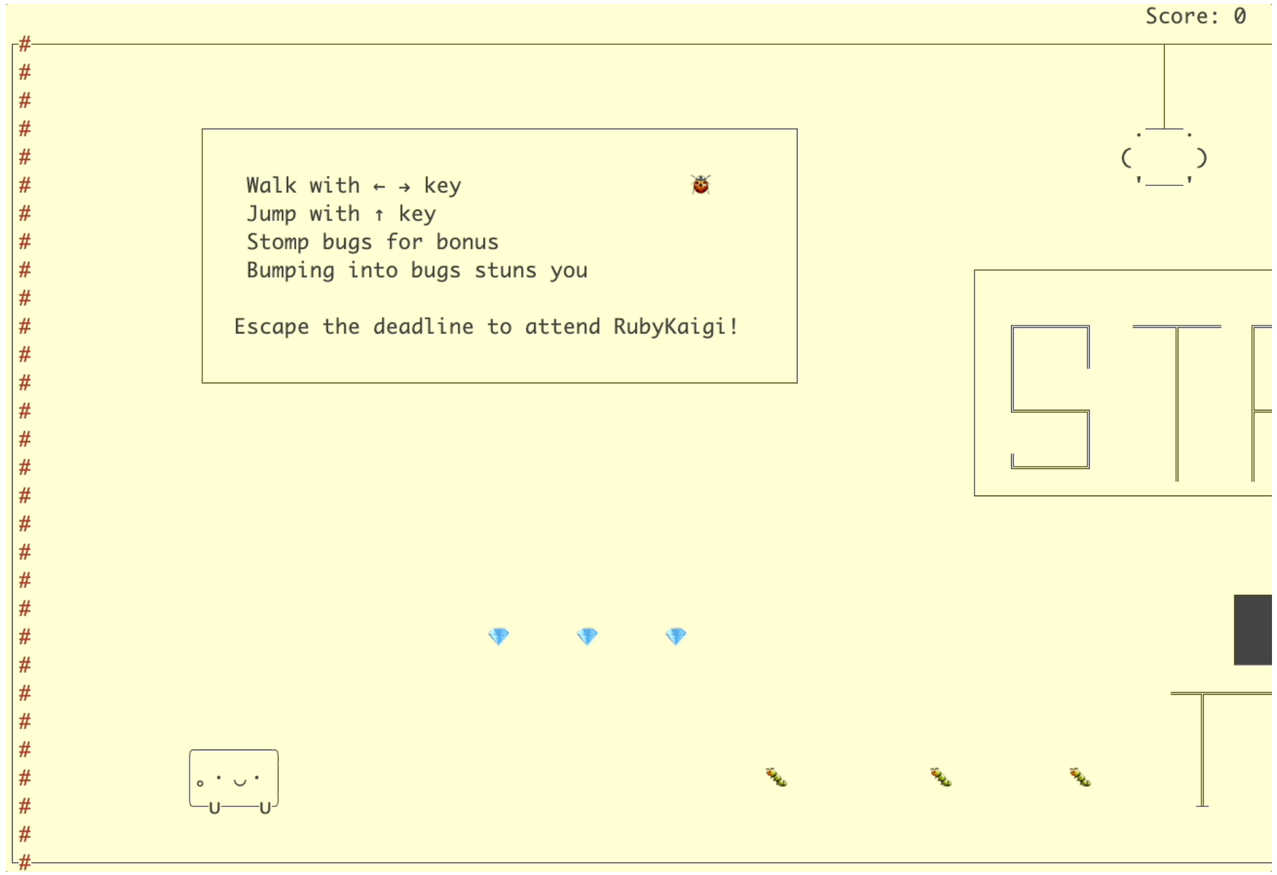
Road to RubyKaigi https://github.com/makicamel/road_to_rubykaigi

Road to RubyKaigi

- A Ruby-made game by Rubyist for Rubyists
- A side-scrolling action game played in the terminal
- Defeat bugs and escape deadlines to reach RubyKaigi venue
- `gem install road_to_rubykaigi`

Road to RubyKaigi

Gameplay Video



Road to RubyKaigi

- Something is missing...
- Music!

4 sound tracks!

Yuya Fujiwara
@asonas

I am Software engineer.

JA

How to make the Groovebox

Ruby isn't just for the Web. Did you know you can create music with it too?

In this talk, I will demonstrate how groovebox-ruby was built from scratch as a functional groovebox using Ruby. What is a groovebox? It's a machine for sketching musical ideas, equipped with tools like synthesizers, step sequencers, filters, samplers, and interactive controls such as buttons, knobs, and displays.

With groovebox-ruby, I began by implementing synthesizer basics like VCOs (Voltage Controlled Oscillators) and VCFs (Voltage Controlled Filters) to understand how these components shape sound. I then added a step sequencer, allowing users to program patterns and interact dynamically with MIDI inputs.

Reimplementing these features in Ruby deepened my understanding of synthesizers, particularly the roles of oscillators and filters. Ruby's extensibility also enabled modular design, such as chaining filters, using MIDI controllers, and building a distributed step sequencer with dRuby.

Ryo Ishigaki
@risgk

Member of Hamamatsu.rb. Embedded software engineer. I have been making synthesizers that run on microcontrollers using C/C++ as a hobby.

Making a MIDI controller device with PicoRuby/R2P2

There are many wonderful synthesizers and electronic instruments in the world. And instruments and devices that support MIDI (Musical Instrument Digital Interface) can easily be played together.

I am making a simple MIDI controller device PRMC-1 for use in electronic music performances using PicoRuby/R2P2 (Ruby Rapid Portable Platform). I can adjust the synthesizer parameters by turning the knobs, or play chords as arpeggios with the sequencer function. The hardware is made with Raspberry Pi Pico, M5Stack Unit 8Angle and Unit MIDI, and no soldering.

All code is written in Ruby! Would you like to create your own musical device using Ruby?

mame & the judges
@tric

JA

TRICK 2025: Episode I

"Feel, don't think."

makicamel
@makicamel

Rubyist. A web application developer. Interested in making games with Ruby.

Road to RubyKaigi: Making Tinny Chiptunes with Ruby

"Road to RubyKaigi" is an action game made in Ruby that you can play right in your terminal. We defeat bugs, dodge deadlines, and race toward the RubyKaigi venue. All the graphics are rendered with text, and the background music is performed on the fly using Ruby. In this talk, I will talk how to implement the BGM performance and present a live demo of the game in action.

Music

- macOS does not have a built-in active sound source by default
  - The user must adjust system settings
  - Inconvenient

A Haiku Here

A Haiku Here

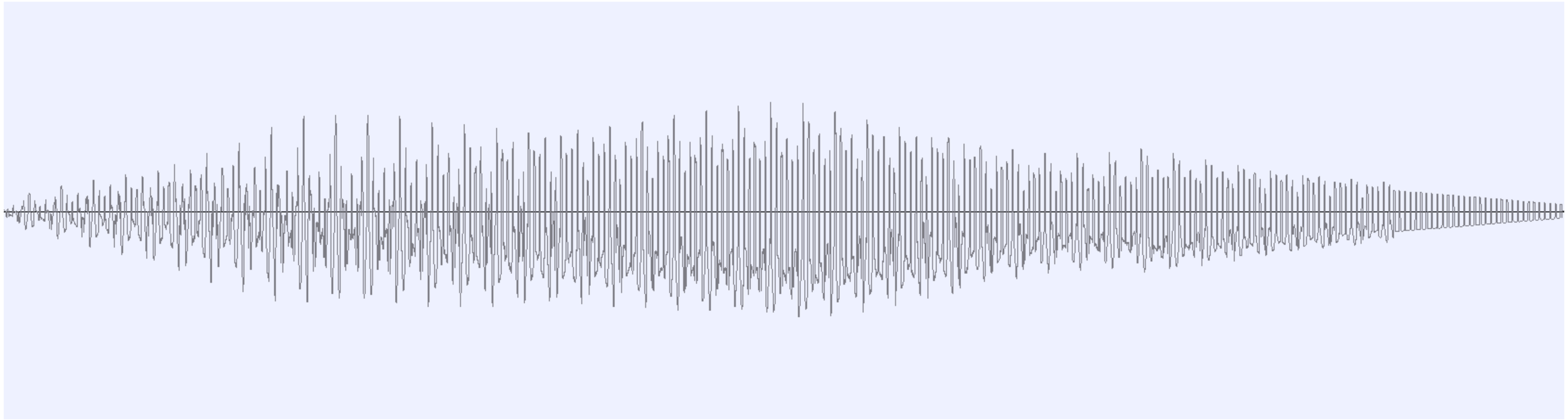
ほしければ
When we want it

コードを書くだけ
just write the code

Rubyist
Rubyist

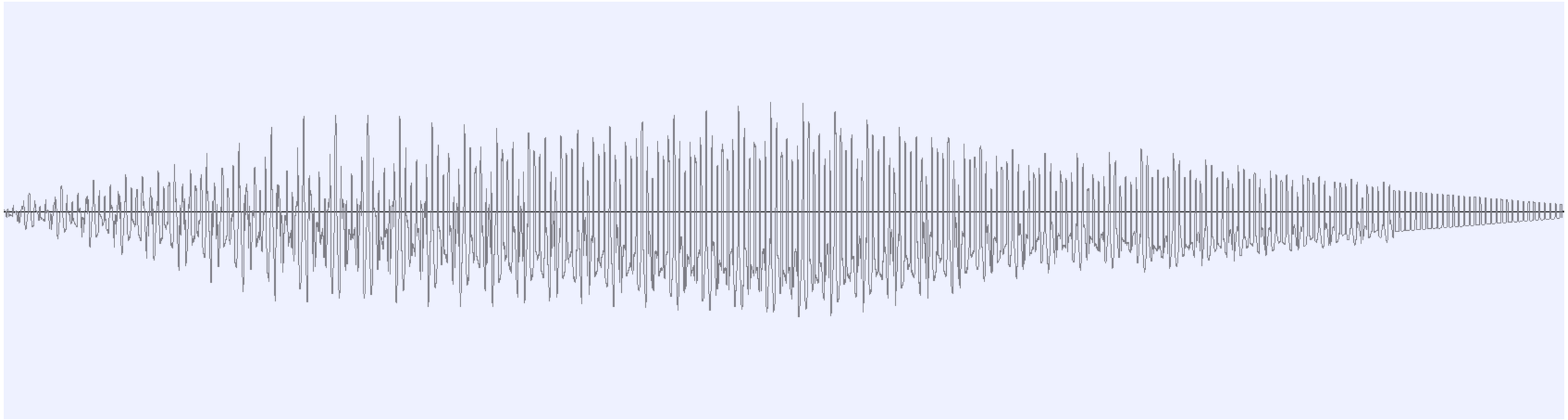
Making Music

- Sound is a wave



Making Music

- A sound source creates the waves
  - Ruby is the sound source

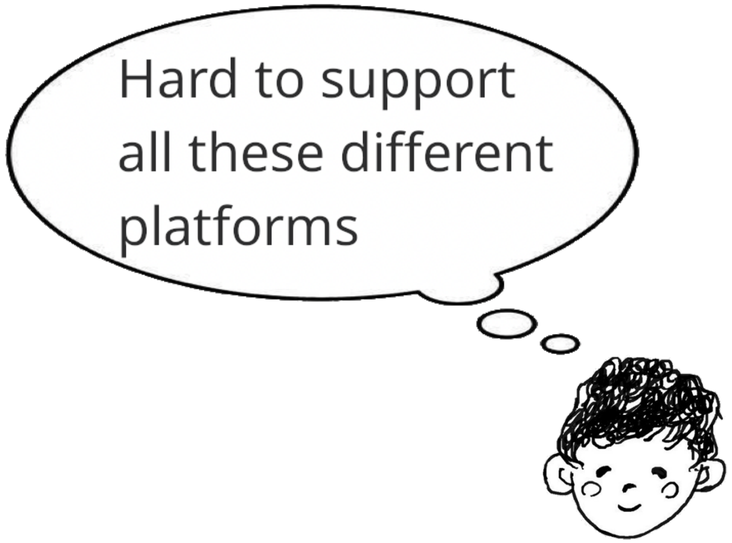


Making Music

- Devices that output the created wave
  - Audio devices
    - e.g. Speakers, Headphones

Making Music

- Audio device I/O APIs
  - macOS
    - Core Audio
  - Windows
    - Windows Audio Session API
  - Linux
    - Advanced Linux Sound Architecture



Making Music

- A cross-platform library that provides audio I/O APIs
  - PortAudio https://github.com/PortAudio/portaudio
- The Ruby binding for PortAudio
  - ffi-portaudio https://github.com/nanki/ffi-portaudio

Making Music

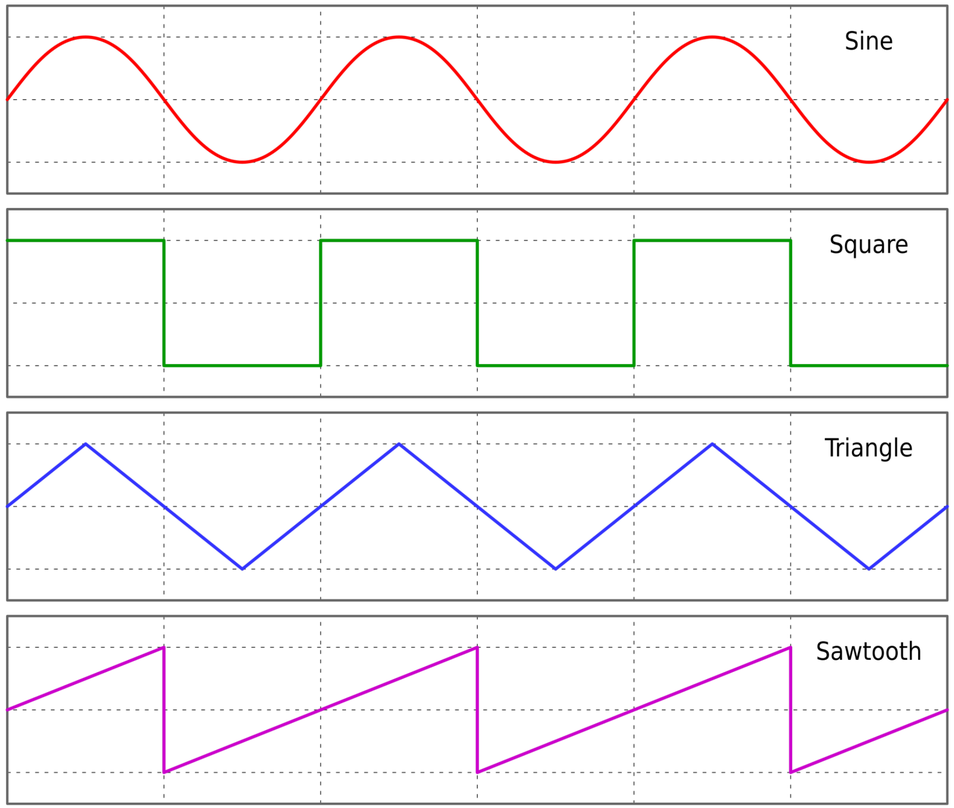
1. Create a wave (audio data)
2. Pass it to ffi-portaudio
   - Output it through PortAudio to an audio device
3. Play sound with Ruby!

Basics of Creating Waves

- "Sound is a wave"
  - What does it mean?

Basics of Creating Waves

- Representative waveforms
  - Sine wave
  - Square wave
  - Triangle wave
  - Sawtooth wave

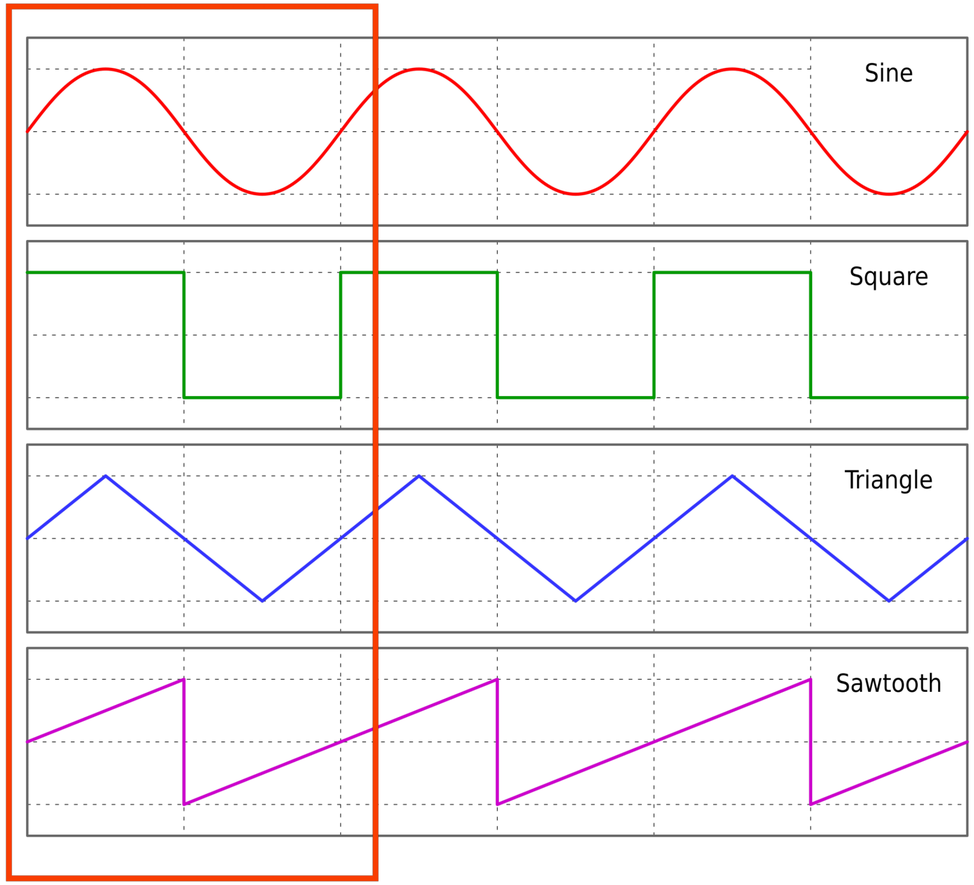


Square wave (waveform) https://en.wikipedia.org/wiki/Square_wave_(waveform)

Basics of Creating Waves

- Periodic
  - The changes in signal amplitude follow a repeating pattern
- Frequency
  - Number of cycles per second
    - e.g. 444.0Hz

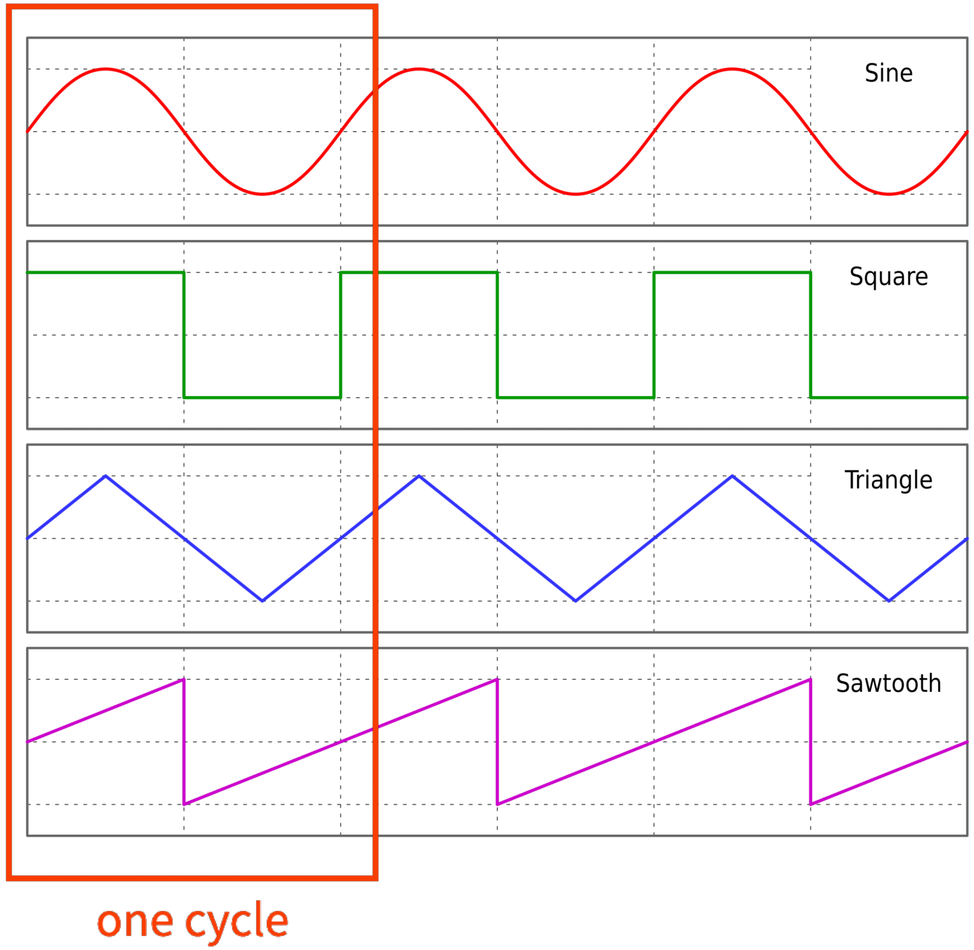
      Repeat 440 cycles per second



Square wave (waveform) https://en.wikipedia.org/wiki/Square_wave_(waveform)

Basics of Creating Waves

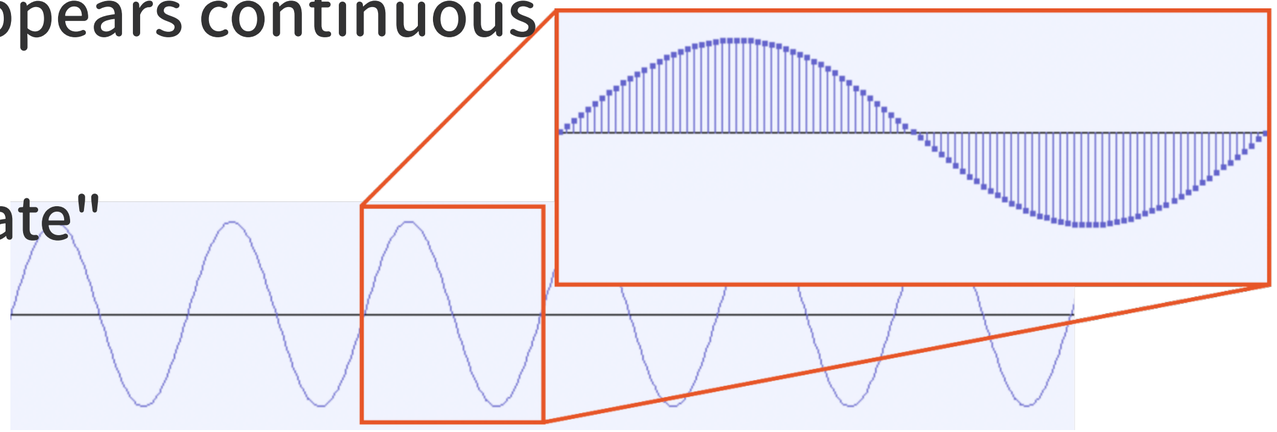
- Period
  - The time it takes for one cycle



Square wave (waveform) https://en.wikipedia.org/wiki/Square_wave_(waveform)

Basics of Creating Waves

- Analog
  - The amplitude of a wave changes continuously within a cycle
- Digital
  - A continuous signal amplitude cannot be recorded
  - Instead, record so that it appears continuous a fixed number of samples
    - This number is "sample rate"
  - Sample rate
    - The number of samples per second



Basics of Creating Waves

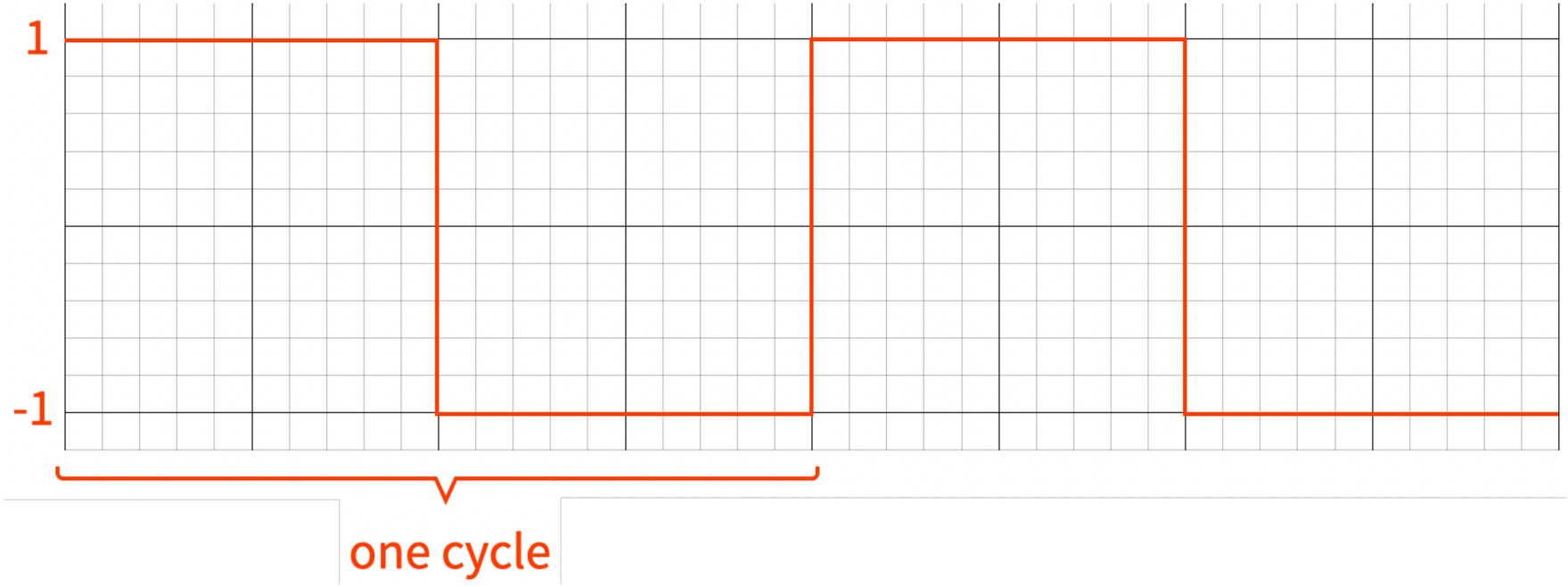
- How to generate waves
  - Just collect the changes in the signal and repeat them

Practical Approach to Creating Waves

- Generate the "pico-pico" sound like the NES
- The waveforms generated by the NES
  - Square wave
  - Triangle wave
- In this talk, we focus on square waves

Creating Waves: Square Wave

- Only two values
  - 1 and -1
- The typical "pico-pico" sound



Creating Waves: Square Wave

- The first half of a cycle is 1
- The second half is -1

Creating Waves: Square Wave

- Frequency
  - The number of cycles per second
  - Time per sample: 1 second / sample rate
  - Increment in current phase per sample: frequency / sample rate

```
class SquareOscillator
  def tick(frequency:)
    @phase += frequency.to_f / sample_rate
    @phase -= 1.0 if @phase >= 1.0 # Phase is managed from 0 to 1, reset when reaching 1
    @phase
  end
```

Creating Waves: Square Wave

- Waveform generation

```
class SquareOscillator
    # ...
    def generate(frequency:)
      # Get the current phase within the cycle
      phase = tick(frequency: frequency)
      # 1 if in the first half, -1 if in the second half
      phase < 0.5 ? 1.0 : -1.0
    end
 end
```

- Collecting samples produces a square wave

Creating Waves: Square Wave

- Passing to PortAudio

```
class AudioEngine < FFI::PortAudio::Stream
  def process(_input, output, framesPerBuffer, _timeInfo, _statusFlags, _userData)
    samples = (0...framesPerBuffer).map { @oscillator.generate(frequency:) }
    output.write_array_of_float(samples)
    :paContinue
  end
# ...
```

- The sound plays! 🎉

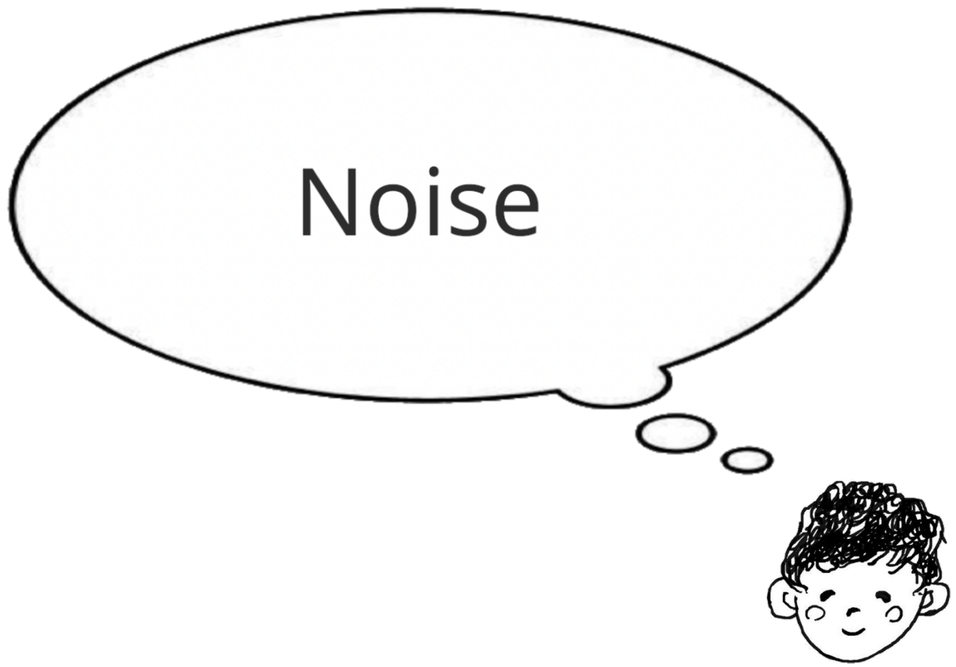
Creating Waves: Square Wave

- Waveform output from the audio device

Creating Waves: Square Wave

- Waveform output from the audio device

Creating Waves: Square Wave



Creating Waves Advanced

- Changing the signal strength abruptly generates noise
  - Many high-frequency components are produced
  - Digital-to-analog conversion

Creating Waves Advanced

- Noise reduction
  - Smoothing the corners

To reduce noise caused by abrupt signal changes, a filter is effective
but since we want "pico-pico" sound, not adopt in this talk

Creating Waves Advanced

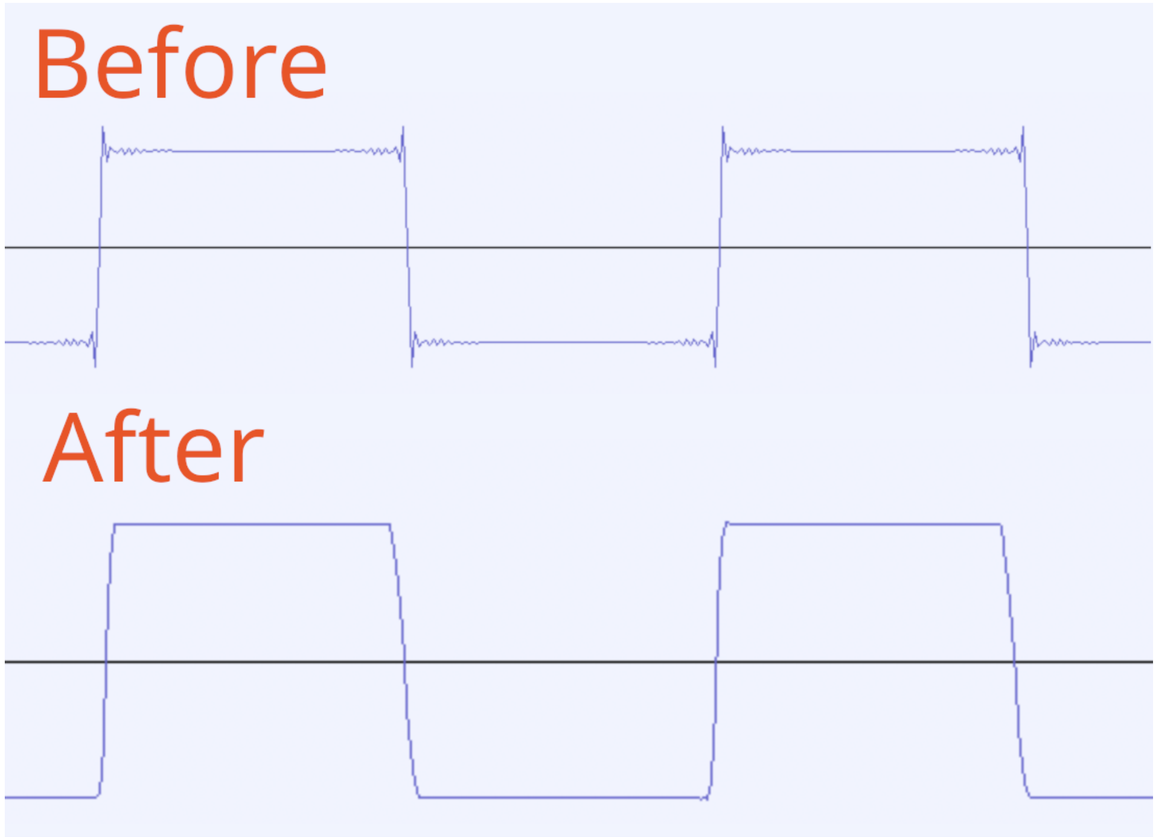
- When the current position in the cycle is at a sharp corner add offset to the signal's strength

Creating Waves Advanced

- When the current position in the cycle is at a sharp corner add offset to the signal's strength
  - Use the cosine function to draw curves like

Creating Waves Advanced

- The waveform becomes clean 🙌



Making Music

- Then, write scores

```
class MelodySequencer < SequencerBase
  GENERATOR = RoundedSquareOscillator
  STACCATO_RATIO = 0.35
  SCORE = [ # 6 Measures
    { frequency: %i[F5], duration: 0.5, envelope: { a: 0.05, d: 0.0, s: 0.5, sl: 0.4, rl: 1.0 } },
    { frequency: %i[C5], duration: 0.5 },
    { frequency: %i[F5], duration: 0.5 },
    { frequency: %i[F5], duration: 0.5 },
    { frequency: %i[C5], duration: 0.5 },
    { frequency: %i[F5], duration: 0.5 },

    { frequency: %i[C5], duration: 0.5 },
    { frequency: %i[F5], duration: 0.25 },
    { frequency: %i[F5], duration: 0.25 },
    { frequency: %i[G5], duration: 0.5 },
    { frequency: %i[F5], duration: 0.5 },
    { frequency: %i[C5], duration: 0.5 },
    { frequency: %i[F5], duration: 0.5 },

    { frequency: %i[REST], duration: 0.5 },
    { frequency: %i[A5], duration: 0.5 },
    { frequency: %i[G5], duration: 0.5 },
    { frequency: %i[F5], duration: 0.5 },
    { frequency: %i[E5], duration: 1.0, envelope: :melody_long, staccato: 0.95 },

    { frequency: %i[F5], duration: 0.5, envelope: { a: 0.15, d: 0.15, s: 0.5, sl: 0.4, rl: 0.9 } },
    { frequency: %i[C5], duration: 0.5 },
    { frequency: %i[A4], duration: 0.5 },
```

Making Music

- Several combinations
  - Triangle wave
    - Pseudo triangle wave
  - Envelopes
  - Sound effects

Making Music

- Simply pass it to the API

```
class AudioEngine < FFI::PortAudio::Stream
  def process(_input, output, framesPerBuffer, _timeInfo, _statusFlags, _userData)
    samples = (0...framesPerBuffer).map { @oscillator.generate(frequency:) }
    output.write_array_of_float(samples)
    :paContinue
  end
# ...
```

Demo 🔊

Special Thanks

@youchan

Enjoy Creating!