



Rusting Game

Rusting game tech / sounds

Cleuton Sampaio.

Fala, #dev! Explorando um pouco mais os recursos e o ecossistema **Rust** vamos criar alguns sons para **Games**. Que tal uma sequência de metralhadora e explosão?

Vamos usar o **crate rodio** para isso! A metralhadora consegui acertar logo, mas o som da explosão ainda é um "work in progress", mas chegaremos lá...

Rodio

Vamos usar a biblioteca **Rodio** para gerar sons em Rust, criando efeitos sonoros como o de uma metralhadora e uma explosão. A seguir, explicarei como o código funciona e destacarei os parâmetros e funcionalidades do Rodio que podem ser utilizados para manipular e criar sons.

Principais Componentes do Rodio Utilizados

- **OutputStream:** Representa o fluxo de saída de áudio. É usado para inicializar o sistema de áudio e obter um manipulador para enviar sons ao dispositivo de reprodução.
- **Sink:** Atua como um coletor de fontes de áudio. Você pode adicionar sons a um **Sink**, e ele gerenciará a reprodução desses sons, permitindo mixagem e controle de tempo.
- **Source:** É uma trait que representa uma fonte de áudio. O Rodio fornece várias implementações de **Source**, como **SineWave** e **Noise**, e também permite que você crie suas próprias fontes personalizadas.

- **SineWave:** Gera uma onda senoidal em uma frequência específica. Útil para criar tons puros e sons baseados em frequências definidas.
- **Mix:** Permite combinar múltiplas fontes de áudio em uma única, misturando os sons para serem reproduzidos simultaneamente.
- **Amplify:** Método usado para ajustar o volume (amplitude) de uma fonte de áudio.
- **take_duration:** Limita a duração de uma fonte de áudio, permitindo que você controle por quanto tempo o som será reproduzido.

Gerando Sons com Rodio

1. Criando Tons Simples com SineWave

Você pode criar um tom puro de uma frequência específica usando `SineWave::new(frequência)`.

```
let onda = SineWave::new(440.0); // Cria um tom de 440 Hz (nota Lá padrão)
```

2. Ajustando o Volume com amplify

Para aumentar ou diminuir o volume de uma fonte de áudio, use o método `amplify(fator)`, onde `fator` é um multiplicador da amplitude original.

```
let onda_amplificada = onda.amplify(0.5); // Reduz o volume pela metade
```

3. Limitando a Duração com take_duration

Para tocar um som por um período específico, utilize `take_duration(duração)`.

```
let onda_curta = onda.take_duration(Duration::from_secs(2)); // Toca por 2 segundos
```

4. Misturando Sons com mix

Para combinar dois ou mais sons, use o método `mix`.

```
let combinado = onda1.mix(onda2); // Combina onda1 e onda2
```

Gerando Ruído Branco Personalizado

No código, é criada uma estrutura `RuidoBranco` que implementa a trait `Source`, permitindo gerar ruído branco personalizado. O ruído branco é útil para simular sons como explosões ou vento.

Principais pontos sobre RuidoBranco:

- **Gerador Aleatório (`SmallRng`)**: Utiliza um gerador de números aleatórios para produzir valores que representam o ruído.
- **Implementação de Iterator**: Gera valores aleatórios em cada chamada de `next()`, que são os samples de áudio.
- **Implementação de Source**: Define as propriedades do som, como número de canais, taxa de amostragem e duração total.

Reproduzindo Sons Complexos

Som de Metralhadora (`som_metralhadora`)

- **Simulação de Disparos**: O código simula 15 disparos, cada um composto por ondas senoidais de frequências graves e agudas.
- **Configuração de Cada Disparo**:
 - **Duração do Tiro**: Cada tiro tem 40 milissegundos de duração.
 - **Pausa entre Tiros**: Há uma pausa de 60 milissegundos entre os tiros.
 - **Criação das Ondas**:
 - **Onda Grave**: `SineWave::new(60.0)` cria uma onda de 60 Hz.
 - **Onda Aguda**: `SineWave::new(120.0)` cria uma onda de 120 Hz.
 - **Ajuste de Volume**: As ondas são amplificadas para ajustar o volume relativo.
 - **Mixagem**: As ondas grave e aguda são misturadas para criar um som mais rico.
- **Reprodução**: Cada som combinado é adicionado a um `Sink` e reproduzido imediatamente.

Som de Explosão (`som_explosao`)

- **Fase Inicial (Impacto)**:
 - **Onda de Impacto**: Uma onda senoidal de 50 Hz é criada para representar o impacto grave da explosão.
 - **Duração e Volume**: A onda tem 300 milissegundos de duração e é amplificada.
- **Deslocamento de Ar (Ruído Branco)**:
 - **Ruído Branco**: Um ruído branco de 500 milissegundos é adicionado para simular o som do ar deslocado.
 - **Ajuste de Volume**: O ruído é amplificado para ser audível.
- **Decaimento do Som**:
 - **Frequências Decrescentes**: Ondas senoidais com frequências de 300 Hz a 100 Hz são reproduzidas em sequência.
 - **Duração e Volume**: Cada onda tem 200 milissegundos de duração com volumes decrescentes.

- **Reprodução Sequencial:** As ondas são adicionadas ao `Sink` uma após a outra, simulando o decaimento do som após a explosão.

Parâmetros e Funcionalidades do Rodio Disponíveis

- **Frequência das Ondas:** Ajuste a frequência em `SineWave::new(frequência)` para alterar o tom.
- **Duração do Som:** Use `take_duration(Duration)` para definir quanto tempo o som deve durar.
- **Amplitude (Volume):** Utilize `amplify(fator)` para aumentar ou diminuir o volume do som.
- **Mixagem de Sons:** Combine várias fontes de áudio com `mix` para criar sons mais complexos.
- **Taxa de Amostragem:** Embora o padrão seja 44.100 Hz, você pode implementar `sample_rate` na sua fonte personalizada para alterar a taxa.
- **Número de Canais:** Defina se o som é mono ou estéreo implementando `channels` na sua fonte.
- **Criação de Fontes Personalizadas:** Implemente a trait `Source` para criar tipos de som específicos, como diferentes tipos de ruído.
- **Efeitos e Filtros:** Embora não utilizados no código, o Rodio permite aplicar efeitos como reverberação, eco, entre outros.

Outras Funcionalidades Úteis do Rodio

- **Reprodução de Arquivos de Áudio:** Você pode carregar e reproduzir arquivos de áudio como MP3, WAV, FLAC, etc.
- **Streaming de Áudio:** Reproduza áudio em streaming de fontes online ou buffers de dados.
- **Controle de Reprodução:**
 - **Pausar e Retomar:** Controle a reprodução dos sons adicionados ao `Sink`.
 - **Volume Global:** Ajuste o volume de todos os sons em um `Sink`.
- **Manipulação de Eventos de Áudio:** Responda a eventos como o término de um som para encadear ações.

O código

O código desse projeto requer instalar dois `crates`, então gere o projeto com:

```
cargo new sound_demo --bin
```

Modifique o arquivo `Cargo.toml` para incluir o `rodio` e o `rand`:

```
[package]
name = "sound_demo"
version = "0.1.0"
```

```
edition = "2021"

[dependencies]
rodio = "0.17"
rand = { version = "0.8", features = ["small_rng"] }
```

Eis o código-fonte:

```
use rodio::{source::{SineWave, Source}, OutputStream, Sink};
use rand::{Rng, SeedableRng};
use rand::rngs::SmallRng;
use std::time::Duration;

// Estrutura para gerar ruído branco manualmente
struct RuidoBranco {
    duracao: Duration,
    restante: f32,
    taxa: u32,
    gerador: SmallRng, // Gerador de números aleatórios compatível com
    `Send`
}

impl RuidoBranco {
    fn new(duracao: Duration) -> Self {
        let taxa = 44100; // Taxa de amostragem padrão
        Self {
            duracao,
            restante: duracao.as_secs_f32() * taxa as f32,
            taxa,
            gerador: SmallRng::from_entropy(), // Inicializa o gerador
        }
    }
}

impl Iterator for RuidoBranco {
    type Item = f32;

    fn next(&mut self) -> Option<Self::Item> {
        if self.restante > 0.0 {
            self.restante -= 1.0;
            Some(self.gerador.gen_range(-0.5..0.5)) // Gera valores
menores para ajustar o volume
        } else {
            None
        }
    }
}

impl Source for RuidoBranco {
    fn current_frame_len(&self) -> Option<u32> {
        None
    }
}
```

```
fn channels(&self) -> u16 {
    1
}

fn sample_rate(&self) -> u32 {
    self.taxa
}

fn total_duration(&self) -> Option<Duration> {
    Some(self.duracao)
}
}

// Função para reproduzir som de metralhadora
fn som_metralhadora() {
    let (_fluxo, manipulador_fluxo) =
OutputStream::try_default().unwrap();
    let duracao_tiro = Duration::from_millis(40); // Cada tiro é curto
(40ms)
    let pausa_entre_tiros = Duration::from_millis(60); // Pausa curta
entre tiros

    for _ in 0..15 { // Simula 15 disparos
        let tanque = Sink::try_new(&manipulador_fluxo).unwrap();

        // Cria um disparo misturando frequências graves e ajustando o
volume
        let onda_grave =
SineWave::new(60.0).take_duration(duracao_tiro).amplify(13.0);
        let onda_aguda =
SineWave::new(120.0).take_duration(duracao_tiro).amplify(12.0);
        let combinado = onda_grave.mix(onda_aguda);

        tanque.append(combinado);
        tanque.sleep_until_end();
        std::thread::sleep(pausa_entre_tiros);
    }

    println!("Som de metralhadora reproduzido!");
}

// Função para reproduzir som de explosão
fn som_explorao() {
    let (_fluxo, manipulador_fluxo) =
OutputStream::try_default().unwrap();

    // Fase inicial: impacto grave
    let tanque = Sink::try_new(&manipulador_fluxo).unwrap();
    let onda_impacto = SineWave::new(50.0)
        .take_duration(Duration::from_millis(300))
        .amplify(5.0); // Volume ajustado
    tanque.append(onda_impacto);
}
```

```
// Adiciona ruído branco para simular deslocamento de ar
let ruido = RuidoBranco::new(Duration::from_millis(500)).amplify(3.0);
// Volume ajustado
tanque.append(ruido);

tanque.sleep_until_end();

// Decaimento: frequências aleatórias com volumes decrescentes
for (frequencia, amplitude) in [(300.0, 4.0), (200.0, 3.0), (150.0,
2.0), (100.0, 1.0)] {
    let tanque = Sink::try_new(&manipulador_fluxo).unwrap();
    let onda_decay = SineWave::new(frequencia)
        .take_duration(Duration::from_millis(200))
        .amplify(amplitude);
    tanque.append(onda_decay);
    tanque.sleep_until_end();
}

println!("Som de explosão reproduzido!");
}

// Função principal
fn main() {
    println!("Reproduzindo som de metralhadora...");
    som_metralhadora();

    println!("Reproduzindo som de explosão...");
    som_explosao();
}
```

Substitua o conteúdo de `src/main.rs` por este arquivo.

Para compilar e executar:

```
cargo run
```



rustingcrab.com