[WIP] Relatório de Pré-Dissertação Mestrado em Engenharia Informática

Pedro Miguel Oliveira da Silva

Setembro, 2019

1 Sinopse

Candidato	Pedro Miguel Oliveira da Silva
Tema	DSL para programação de teclados e acompanhamentos musicais dinâmicos virtuais
Orientação	José João Almeida
Instituição	Departamento de Informática Escola de Engenharia Universidade do Minho

2 SoundFonts

O formato *SoundFont* foi originalmente desenvolvido nos anos 90 pela empresa E-mu Systems para ser usado inicialmente pelas placas de som Sound Blaster. Ao longo dos anos o formato sofreu diversas alterações, encontrando-se atualemente na versão 2.04, lançada em 2005[1]. Atualmente existem diversos sintetizadores de software *cross platform* e open source capazes de converterem eventos *MIDI* em som usando ficheiros SoundFont, dispensando a necessidade de uma placa de som compatível com o formato. Alguns destes projetos são TiMidity++, WildMIDI e FluidSynth.

Um ficheiro de SoundFont é constítuido por um ou mais bancos (*banks*) (até um máximo de 128). Cada banco pode por sua vez ter até 128 *presets* (por vezes também chamados instrumentos ou programas).

TODO

3 FluidSynth

A biblioteca FluidSynth é um *software* sintetizador de aúdio em tempo real que transforma dados MIDI em sons, que podem ser gravados em disco ou encaminhados diretamente para um *output* de aúdio. Os sons são gerados com recurso a SoundFonts[1] (ficheiros com a extensão `.sf2`) que mapeiam cada nota para a gravação de um instrumento a tocar essa nota.

Os *bindings* da biblioteca para C# foram baseados no código *open source* do projeto NFluidSynth[2], com algumas modificações para compilar com a versão da biblioteca em Linux.

3.1 Inicialização

Para utilizar a biblioteca FluidSynth, existem três objetos principais que devem ser criados: Settings (`fluid_settings_t*`), Synth (`fluid_synth_t*`) e AudioDriver (`fluid_audio_driver_t*`).

O objecto **Settings**[3] é implementado com recurso a um dicionário. Para cada

chave (por exemplo, "`audio.driver`") é possível associar um valor do tipo inteiro (`int`), *string* (`str`) ou *double* (`num`). Alguns valores podem ser também booleanos (`bool`), no entanto eles são armazenados como inteiros com os valores aceites sendo apenas 0 e 1.

O objeto **Synth** é utilizado para controlar o sintetizador e produzir os sons. Para isso é possível enviar as mensagens MIDI tais como `NoteOn`, `NoteOff`, `ProgramChange`, entre outros.

O terceiro objeto **AudioDriver** encaminha automaticamente os sons para algum *audio output*, seja ele colunas no computador ou um ficheiro em disco. Os seguintes *outputs* são suportados pela biblioteca:

Linux: jack, alsa, oss, PulseAudio, portaudio, sdl2, file

Windows: jack, PulseAudio, dsound, portaudio, sdl2, file

Max OS: jack, PulseAudio, coreaudio, portaudio, sndman, sdl2, file

Android: opensles, oboe, file

3.2 Utilização

Com os objetos necessários inicializados, é necessário ainda especificar qual (ou quais) a(s) *SoundFont(s)* a utilizar. Para isso podemos chamar o método `Synth.LoadSoundFont` que recebe dois argumentos: uma *string* com o caminho em disco do ficheiro *SoundFont* a carregar, seguido dum booleano que indica se os *presets* devem ser atualizados para os da nova *SoundFont* (isto é, atribuir os instrumentos da *SoundFont* aos canais automaticamente).

A função `Synth.NoteOn` recebe três argumentos: um inteiro a representar o canal, outro inteiro entre 0 e 127 a representar a nota, e finalmente outro inteiro também entre 0 e 127 a representar a velocidade da nota.

O canal (**channel**) representa qual o instrumento que vai reproduzir a nota em questão. Cada canal está atríbuido a um programa da SoundFont, e é possível a qualquer momento mudar o programa atribuido a qualquer canal através do método `Synth.ProgramChange`. Caso se tenha carregado mais do que uma *SoundFont*, é possível usar o método `Synth.ProgramSelect`, que permite especificar o `id` da *SoundFont* e do banco do instrumento a atribuir.

A chave (**key**) representa a nota a tocar. Sendo este valor um inteiro entre 0 e 127, é necessário saber como mapear as tradicionais notas músicais neste valor. Para isso, basta colocarmos as *pich classes* e os seus respetivos acidentais *sharp* numa lista ordenada (`C, C#, D, D#, E, F, F#, G, G#, A, A#, B`) e associar a eles os inteiros entre 0 e 11 (inclusive). Depois apenas temos de somar a esse número a multiplicação da oitava da nota (a começar em 0) por 12. Podemos deste modo calcular, por exemplo, que a *key* do C central (C4) é igual a 48 ($0 + 4 * 12$).

$$N + O * 12$$

A velocidade (**velocity**) é também um valor entre 0 e 127. Relacionando a velocidade com um piano físico, esta representa a força (ou velocidade) com que a tecla foi premida. Velocidades maiores geram sons mais altos, enquanto que velocidades mais baixas geram sons mais baixos, permitindo assim ao músico dar ou tirar enfase a uma nota relativamente às restantes. De notar que um valor igual a zero é o equivalente a invocar o método `Synth.NoteOff`.

A método `Synth.NoteOff`, por sua vez, recebe apenas dois argumentos (canal e chave), e deve ser chamada passsado algum tempo para terminar a nota. Podemos deste modo construir a analogia óbvia que o método `NoteOn` corresponde a uma tecla de piano ser premida, e `NoteOff` corresponde a essa tecla ser libertada.

References

[1] Soundfont technical specification. `http://www.synthfont.com/sfspec24.pdf`, February 2006.

[2] Atsushi Eno. Nfluidsynth. `https://github.com/atsushieno/nfluidsynth`, 2019.

[3] Fluidsynth settings. `http://www.fluidsynth.org/api/fluidsettings.xml`.