CompMus 2020 - Roteiro para a Tarefa Prática 3 - 09/10/2020
Fundamentos do processamento sonoro em tempo real

1) Abra o Puredata, crie um novo patch, acrescente nome e número USP como comentários e salve (com o nome que quiser). Lembre-se de fazer entregas parciais a cada etapa da implementação, submetendo o trabalho no link Terceira Tarefa Prática (TP3).

2) Nesse trabalho vamos explorar o modo de operação principal do Puredata, que é o processamento de fluxos sonoros em tempo real. O objetivo é entender como funcionam as conexões entre objetos que processam sinais de áudio (objetos **~**), reproduzindo o comportamento de alguns objetos elementares. Para isso, começaremos com uma implementação em Puredata "puro" usando os objetos **[+~]**, **[*~]**, **[dac~]**, (2x) **[hslider]** ("Controle Deslizante Horizontal"), sendo um configurado para produzir valores de frequência logaritmicamente distribuídos entre 200...2000 Hz e o outro configurado para produzir valores de amplitude logaritmicamente distribuídos entre 0.0001...1, **[noise~]** e **[osc~]**, além de um **[vradio]** (“Botão Seletor Vertical”) de 4 posições e um **[multiplex~]** (use **[declare -lib zexy]** antes de criá-lo) para selecionar uma das 4 opções de sinal: silêncio, ruído, senoide e ruído+senoide. Conecte estes objetos de tal maneira que você consiga selecionar e ouvir as 4 opções acima, além de controlar a frequência do oscilador e o volume do sinal que sai do multiplex.

3) Nos bastidores, o Puredata está interrompendo o processamento (e os sinais) a cada 64 amostras (tamanho de bloco default), e chamando uma função (*perform*) para cada objeto que processa áudio (por convenção, esses são os objetos terminados em **~**). Cada um desses objetos recebe um ou mais sinais de áudio, ou seja, blocos com 64 amostras, e devolve um ou mais sinais de áudio. Tome por exemplo o objeto **[+~]**: sua implementação em ofelia/Lua seria:

```
ofelia function -s21;
for i=1,#a1 do;
a1[i] = a1[i]+a2[i];
end;
return a1;
```

   Experimente criar esse objeto, e substituí-lo pelo **[+~]** em sua implementação anterior. Tudo deverá funcionar da mesma maneira! Observe que o parâmetro `-s21` cria 2 inlets (associados aos sinais de entrada a1 e a2) e 1 outlet (associado ao `return a1`). Observe também que `#a1` representa o tamanho do primeiro vetor, que como todas as demais tabelas em Lua é indexado de 1 até `#a1`, sendo usado tanto para a entrada quanto para a saída[1].

4) Uma maneira interessante de encapsular implementações e tornar nossos objetos mais fáceis de serem reutilizados é guardá-los em *abstrações*, que nada mais são do que arquivos Pd com objetos **[inlet~]** e **[outlet~]** representando entradas e saídas. Crie um novo arquivo "`soma~.pd`" com o código ofelia acima, conectado aos objetos (2x) **[inlet~]** e **[outlet~]**. Salve

[1] Seria possível definir um outro vetor para a saída, declarando por exemplo "`local b1 = ofTable();`", mas isso tornaria essa implementação computacionalmente menos eficiente, tanto no uso do espaço quanto do tempo.

o arquivo, e substitua o **[+~]** da implementação original por **[soma~]**, verificando que o funcionamento continua igual. De forma análoga, substitua o **[*~]** por uma abstração **[mult~]**.

5) Vamos agora recriar o objeto **[noise~]** fazendo uma abstração "`ruído~.pd`". Use uma função ofelia com uma entrada e uma saída de áudio, preenchendo os valores do vetor `a1` com a expressão `2*math.random()-1`, que corresponde a um ruído uniforme entre -1...1. Substitua sua implementação no patch original, certificando-se de que o funcionamento não mudou.

6) Nosso próximo passo será criar um seletor de sinais como o **[multiplex~]**. Esse objeto receberá um valor de um seletor vertical e quatro sinais de áudio, e produzirá na saída o fluxo da entrada correspondente ao índice do seletor. Para isso, vamos criar um *módulo* ofelia, que é um objeto que possui variáveis locais (*atributos*) para armazenar seu estado e funções para processar entradas (*mensagens*) de tipos diferentes. Comece com a declaração "`ofelia define -s41;`" para definir um módulo com quatro entradas de áudio e uma saída. Em seguida, use a declaração "`local estado=0;`" para criar uma variável local *do módulo* que armazenará o estado do interruptor. Esta variável é um *atributo* do objeto, mantendo seu valor Depois disso, crie duas funções: "`function ofelia.float(a);`" que receberá valores numéricos do interruptor, ligado ao primeiro inlet, e guardará esse valor na variável estado; e "`function ofelia.perform(a0,a1,a2,a3);`" que fará o processamento dos 4 sinais da entrada, devolvendo o sinal correspondente ao valor da variável `estado`. Não se esqueça que todas as funções precisam de um "`end;`". Encapsule esse módulo em uma abstração "`seletor~.pd`" com um **[inlet]**, quatro **[inlet~]** e um **[outlet~]**, e use-o no lugar do seletor da implementação original. Obs: tanto o **[inlet]** que recebe o valor do seletor quanto o **[inlet~]** que recebe o primeiro sinal devem ser conectados ao primeiro inlet do módulo ofelia.

7) Nosso último exercício será recriar o objeto **[osc~]**, através de uma abstração `oscilador~.pd` que usa um módulo "`ofelia define -s11;`" para produzir um sinal senoidal a partir de um controle (contínuo) de frequência na entrada. Esse é um problema menos simples do que parece: se usássemos uma expressão do tipo `math.sin(2*math.pi*freq*n/R)` para um contador `n` que sempre cresce, não conseguiríamos controlar a frequência de forma contínua (cada mudança repentina do valor `freq` gera uma descontinuidade naquela fórmula). Ao invés disso, usaremos uma variável (do módulo) "`local fase=0;`" que será incrementada a cada amostra com o valor correspondente à mudança de fase (atualização do ângulo que aparece dentro do seno) entre uma amostra e outra. Para cada valor de frequência f, essa variação de fase é dada pela expressão $2\pi f\frac{n+1}{R} - 2\pi f\frac{n}{R} = \frac{2\pi f}{R}$, que deve ser atualizada a cada amostra em função da frequência instantânea correspondente. Coloque sua implementação lado a lado com o **[osc~]** original, comparando as saídas das duas implementações ao variar o **[hslider]** que controla a frequência (coloque por exemplo cada uma das implementações em um canal de saída do **[dac~]** para compará-las perceptualmente). Uma situação de teste interessante consiste em controlar um oscilador com frequências oscilantes geradas por um segundo oscilador (o efeito musical do *vibrato*), fazendo por exemplo **[osc~ 1]→[expr~ 400+20*$v1]→[osc~]**.

8) Guarde todos os arquivos `.pd` gerados (o arquivo principal e também `soma~.pd`, `mult~.pd`, `ruído~.pd`, `seletor~.pd` e `oscilador~.pd`) em um arquivo `zip` ou `tgz`, e entregue o trabalho no PACA.