ILDA BASSO (Org.)
JOSÉ CARLOS RODRIGUES ROCHA (Org.)
MARILEIDE DIAS ESQUEDA (Org.)

II SIMPÓSIO INTERNACIONAL DE EDUCAÇÃO
LINGUAGENS EDUCATIVAS:
PERSPECTIVAS INTERDISCIPLINARES NA ATUALIDADE

BAURU
2008

S6126 Simpósio Internacional de Educação (2. : 2008 : Bauru, SP)

Anais [recurso eletrônico] / 2. Simpósio Internacional de Educação / Ilda Basso, José Carlos Rodrigues Rocha, Marileide Dias Esqueda (organizadores). – Bauru, SP : USC, 2008.

Simpósio realizado na USC, no mês de junho de 2008, tendo como tema : Linguagens educativas – perspectivas interdisciplinares na atualidade.
ISBN 978-85-99532-02-7.

1. Educação – simpósios. 2. Linguagens educativas. I. Basso, Ilda. II. Rocha, José Carlos Rodrigues. III. Esqueda, Marileide Dias. VI. Título.

CDD 370

Csound e Python como ferramentas de composição

Rafael Salgado Ribeiro
Universidade do Sagrado Coração

Resumo: O presente artigo busca apresentar e discutir o uso de duas ferramentas para a composição assistida por computador, CSound e Python, fazendo uma breve introdução às duas ferramentas e sugerindo algumas possibilidades de integração. O CSound foi uma das primeiras linguagens de programação para áudio a adotar o conceito de ser modular e a primeira a ser escrita em C. Python é uma linguagem relativamente nova, cujo foco é a portabilidade e a facilidade em programar e em manter um código limpo. Abordamos assuntos como composição assistida por computador utilizando redes neurais, recursividade, intervenção do usuário, lógica *fuzzy* e modelos matemáticos, além de sugerir várias outras linhas de pesquisa que poderiam ser abordadas a partir deste trabalho inicial, como composição de música aleatória, integração de qualquer tipo de sistema físico (desde que as entradas sejam devidamente trabalhadas), pesquisa de timbres e até mesmo desenvolvimento de linguagens de programação para *real-time scripting*, uma espécie de improvisação com o computador. Todos os exemplos que demos no decorrer deste artigo são gerados a partir de funções randômicas, o que faz com que a cada execução de um determinado algoritmo seja produzida uma saída diferente e imprevisível. Vale ressaltar, ainda a respeito dos exemplos, que todos eles estão comentados linha a linha, salvo algumas exceções em que o autor julgou que não haveria necessidade de se comentar por já haver explicado a função ou bloco de código do qual foi omitido o comentário previamente. Procuramos explicar todas os termos técnicos utilizados, tanto no que diz respeito à música quanto à programação, como a definição de funções e classes, conceitos importantes da Programação Orientada a Objetos, no entanto, por fugir do escopo deste artigo, não explicamos elementos específicos do Python ou do CSound, nos limitando a explicar a integração entre ambos e a sugerir textos para os que quiserem se aprofundar em alguma das ferramentas ou em ambas. Nos esforçamos para recomendar os softwares que fossem gratuitos e estivessem disponibilizados para download na internet, permitindo, desta maneira, que os usuários não encontrassem dificuldades para os achar ou comprar, no entanto, lembramos que, devido à dinâmica de atualização dos softwares livres ser muito rápida, alguns endereços na internet podem mudar. Utilizamos como padrão de saída os arquivos Wave, o mesmo padrão utilizado em CD's que, conseqüentemente, pode ser

reproduzido em qualquer reprodutor de mídia capaz de ler CD's. Um estudo mais aprofundado dessas ferramentas e suas possíveis integrações é muito válido para o músico, encarando nessa união novas possibilidades de composição.
Palavras-chave: Música, CSound, Python, composição assistida por computador

Abstract: This article wants to present and discuss two computer-aided composition tools, CSound and Python, making a short introduction to then and suggesting some interaction possibilities. CSound was one of the first audio program language that adopted the modular concept and the first wrote in C language. Python is a new language, and is focused in portability and facility of programming and let the source code as clean as possible. We addressed some subjects like computer aided composition using neural networks, recursion, user intervention, fuzzy logic and mathematics models, and suggested some researches that could be addressed from this initial work, like random music composition, physical systems integration, timbre researches or even development of program languages for real-time scripting, a kind of improvisation with the computer. All the examples in this article are generated from random functions, what make that each execution of the code give a different output. The examples are commented line by line, except the cases that the author judge that is no reason to comment because they were explained previously. We tried to explain all music or programming technical terms, like functions and classes definition, important concepts of Object Oriented Programming, but we omitted some specific elements of Python or CSound, explaining only the integration of them and suggesting texts for who wants to study some of the tools or even both. We tried to recommend free softwares that are available to download. By this way, users will not have trouble in finding or buying them, but free software have many updates, so links may be changed. We used wave files as default output, the same used in CD's and can be played in every player able to play CD's. A study of these tools and possible integrations is a good study for the musician, seeing in this union the new possibility of composition.
Keyword: Music, CSound, Python, computer-aided composition

1. Introdução

Este artigo tem como objetivo introduzir duas poderosas ferramentas para composição assistida por computador. Trata-se do CSound e do Python que, como veremos, podem

transformar a composição em algo muito além de nossa imaginação sem nunca fugir de nosso controle. Aos que não têm conhecimento de programação, sugerimos que leiam os tutoriais em www.python.org

2. Conhecendo as ferramentas

2.1 Csound

"Csound é uma linguagem de programação desenvolvida e otimizada para áudio. A linguagem consiste em mais de 1300 opcodes – os códigos operacionais que o designer de som usa para construir seus instrumentos (timbres)".[1] Trata-se de uma linguagem muito flexível, a ponto de o único limite que o designer de som enfrenta é seu próprio conhecimento ou imaginação, mas nunca a linguagem em si. O nome CSound vem da linguagem de programação em que foi escrito, o C, ao contrário de outras linguagens antecessoras a ele. Sua principal característica é ser modular, o que permite sempre ampliar a linguagem sem alterar suas estruturas básicas. Com ele é possível criar desde timbres simples até muitíssimos complexos, além de escrever a partitura. É open-source, ou seja, qualquer um pode alterar seu código fonte e ajudar no desenvolvimento da linguagem, o que proporciona um grande grupo de pesquisadores e programadores trabalhando em conjunto para seu desenvolvimento. O CSound é um software livre, ou seja, qualquer um pode fazer o download na internet e utilizá-lo sem pagar nada. Há um bom tutorial em português disponível no endereço http://www.csounds.com/toots/pt/.

2.2 Python

"Python é uma linguagem de programação orientada a objetos, dinâmica e interativa, que pode ser usada para muitos tipos de desenvolvimento de software. Oferece um forte suporte à integração com outras linguagens e ferramentas, vem com várias bibliotecas por padrão e pode ser aprendida em poucos dias".[2] Além de sua forte tipagem - característica que permite definir o tipo de variável dinamicamente - e interatividade, que permite que as saídas das funções sejam vistas na medida em que se programa, ainda possui a vantagem de funcionar sobre praticamente qualquer plataforma ou sistema operacional, seja um computador com Windows, Linux, MacOS ou mesmo em celulares Nokia e outros hardwares para os quais hajam suporte. Junto com a linguagem, é instalado um interpretador de Python, o Idle. Ele é muito útil para começar a programar, mas depois de

um certo tempo não tem tantas vantagens práticas, motivo pelo qual sugerimos outros editores nesse artigo. Da mesma forma que o CSound, é open-source e possui as mesmas vantagens que ele, podendo também ser obtido gratuitamente. Há vários tutoriais sobre a linguagem na internet, mas os mais recomendados são os que se encontram no site do projeto. Como foge do nosso escopo explicar todas as funções da linguagem, nos limitaremos ao que for necessário para o entendimento deste artigo, tendo sempre em mente que utilizaremos o Python para programar e gerar os arquivos do CSound.

3. Obtendo o Python e o Csound

Tanto o Python quanto o Csound podem ser obtidos nos sites dos projetos. Para fazer o download do Python, acesse www.python.org. Para obter o Csound, acesse www.csounds.com. Ambos os softwares estão constantemente sofrendo atualizações, dessa forma, é recomendável ficar sempre atento a esses dois links.

4. Usando o CSound no Python

Para que se possa abrir, manipular ou criar um arquivo CSD (Arquivo unificado do CSound) no Python, há a necessidade de se ter uma biblioteca que servirá como intermediária na comunicação, tornando mais rápida a programação-composição. Essa biblioteca é chamada *csnd* e vem instalada por definição com o CSound. Para utilizá-la, temos que importá-la. Isso é feito através do comando:

```
import csnd
```

Você pode testar isso no Idle, para se certificar que a biblioteca *csnd* está corretamente instalada. Para ter certeza que a operação foi bem sucedida, use o seguinte comando:

```
print dir(csnd)
```

A saída será uma lista com os comandos que a biblioteca oferece. Agora vamos utilizar a *csnd*.

5. Entendendo o básico da CSND

Alguns comandos básicos da biblioteca podem nos ser muito úteis. São comandos básicos

como abrir e compilar um arquivo csd. Antes de tudo, é preciso criar uma instância de um objeto da *csnd*, o *CppSound,* que aqui chamaremos de *var*:

```
var = csnd.CppSound()
```

Agora nossa variável *var* possui todos as funções aplicáveis (métodos) a *CppSound*. Um teste simples para verificar se tudo correu bem é fazer isso no Idle e em seguida digitar "*var.*" e apertar Ctrl+Espaço. Isso fará com que o Idle mostre uma lista com todos os métodos de *var*.

Nosso primeiro passo será abrir e compilar um arquivo csd. Ao compilar, estamos transformando as estradas de texto em uma saída sonora, no caso do CSound. Recomendamos a partir de agora escrever em um editor de texto e depois testá-lo, não mais utilizando o Idle. Um editor muito recomendado é o Notepad++, que possui a mesma funcionalidade do Idle em destacar comandos em cores. Para fazer o download, acesse http://notepad-plus.sourceforge.net/uk/download.php. Vamos criar o seguinte arquivo csd:

```
1       <CsoundSynthesizer>
2
3        <CsOptions>
4         -o ex1.wav
5        </CsOptions>
6
7       <CsInstruments>
8
9       sr = 44100
10      kr = 4410
11      ksmps = 10
12      nchnls = 1
13
14      instr 1
15
16        kamp = 30000
17        kcps = p4
18        ifn = 1
19
20        a1 oscil kamp, kcps, ifn
21
22       out a1
23
24      endin
25
```

```
26      </CsInstruments>
27
28      <CsScore>
29
30      f 1 0 16384 10 1
31
32      i 1 0 1 7.00
33      i 1 . 1 7.01
34      i 1 . 1 7.02
35      i 1 . 1 7.03
36      i 1 . 1 7.04
37      i 1 . 1 7.05
38      i 1 . 1 7.06
39      i 1 . 1.7.07
40      i 1 . 1 7.08
41      i 1 . 1 7.09
42      i 1 . 1 7.10
43      i 1 . 1 7.11
44      e
45
46      </CsScore>
47
48      </CsoundSynthesizer>
```

Comentando o arquivo, teremos:

Linha 1: abre o arquivo como um CSound Unified File (Arquivo do CSound unificado)

Linha 3: inicia a seção com as opções de compilação

Linha 4: configura o arquivo de saída para ex1.wav (arquivos wave são arquivos básicos de áudio digital)

Linha 5: finaliza a seção com as opções de compilação

Linha 7: inicia a seção onde serão escritos os timbres

Linha 9: define a taxa de amostragem como 44100 amostras por segundo.

Linha 10: define a taxa de controle como 4410 amostras por segundo

Linha 11: indica a razão entre a tava de amostragem e a taxa de controle (sr/kr)

Linha 12: define o número de canais

Linha 14: define o instrumento 1

Linha 16: define amplitude como 30000. Para nosso exemplo usaremos esse valor fixo, mas ele poderia ser dado por um parâmetro qualquer, como fizemos na linha seguinte com a altura.

Linha 17: define a altura sendo passada como referência no parâmetro quatro da seção CsScore. O CSound usa o formato [oitava].[nota] para alturas do sistema temperado, sendo

o número “7” a oitava do Dó central e 0 correspondente à nota Dó. Assim 7.02 seria o Ré, 7.11 Si, na mesma oitava e assim sucessivamente.

Linha 18: define que o instrumento será tocado na ftable 1

Linha 20: cria um oscilador em a1

Linha 22: envia para a saída (o arquivo .wav)

Linha 24: finaliza o instrumento 1

Linha 26: finaliza a seção de instrumentos (timbres)

Linha 28: inicia a seção da partitura (score)

Linha 30: cria a ftable número 1

Linhas 32-43: toca, uma escala cromática no instrumento 1

Linha 44: finaliza a partitura

Linha 46: finaliza a seção score

Linha 48: finaliza o arquivo CSound Unifield File

O mesmo arquivo csd poderia ter sido criado dinamicamente (sem criar o arquivo csd no disco) utilizando-se Python, da seguinte forma:

```
1       import csnd
2       var=csnd.CppSound()
3       var.SetPythonMessageCallback()
4       orc='''
5       sr=44100
6       kr=4410
7       ksmps=10
8       nchnls=1
9
10      instr1
11
12        kamp=30000
13        kcps=p4
14        ifn=1
15        a1 oscil kamp, kcps, ifn
16       out a1
17      endin
18      '''
19      sco='''
20      f 1 0 16384 10 1
21      i 1 0 1 7.00
22      i 1 . 1 7.01
23      i 1 . 1 7.02
24      i 1 . 1 7.03
```

```
25      i 1 . 1 7.04
26      i 1 . 1 7.05
27      i 1 . 1 7.06
28      i 1 . 1.7.07
29      i 1 . 1 7.08
30      i 1 . 1 7.09
31      i 1 . 1 7.10
32      i 1 . 1 7.11
33      e
34      '''
35      var.setOrchestra(orc)
36      var.setScore(sco)
37      var.setCommand('csound - o ex1.  wav')
38      print var.getCSD()
39      var.save('ex1.csd')
40      var.exportForPerformance()
41      var.Perform()
```

Comentando, teríamos:

Linha 1: importa a biblioteca *csnd*

Linha 2: cria uma variável de nome *var* do tipo CppSound

Linha 3: permite que as saídas do CSound durante a compilação sejam visualizadas no Python

Linhas 4-18: Define a orquestra na variável orc. As aspas triplas são usadas para editar variáveis do tipo string de mais de uma linha.

Linhas 19-34: Define a score na variável sco

Linha 35: Define a seção CsInstruments do arquivo csd como o conteúdo da variável orc

Linha 36: Define a seção CsScore do arquivo csd como o conteúdo da variável sco

Linha 37: Define as opções da saída para um arquivo wave chamado ex1.wav

Linha 38: imprime o conteúdo do arquivo csd criado pelo Python na tela

Linha 39: Salva o arquivo com o nome de "ex1.csd"

Linha 40: Prepara o arquivo para ser compilado

Linha 41: Compila o arquivo

Se tudo tiver corrido bem, o arquivo ex1.wav deve ter sido criado na mesma pasta que os arquivos csd e py.

6. Python, o luthier: gerando um instrumento (timbre)

Além da possibilidade de se criar e compilar um arquivo csd a partir do Python temos

algumas possibilidades de aumentar as opções de programação aproveitando as facilidades que a linguagem nos oferece. Implementações utilizando recursividade, redes neurais ou ainda utilizando a intervenção do usuário podem ser desenvolvidas aproveitando o poder da linguagem e convertendo isso em um arquivo do CSound. Apresentaremos aqui um breve panorama de tal aplicação, utilizando valores randômicos, o que resultará em saídas completamente imprevistas, no entanto pode-se utilizar vários algoritmos para que se possa ter um controle maior sobre a saída, conforme for do interesse do programador. Juntamente com o Professor Ms. Marcelo Mello, desenvolvi um projeto no Núcleo de Música da Universidade do Sagrado Coração que fazia uso de algumas desses princípios que agora expomos. Logicamente, quanto maior for o conhecimento em programação e em música, maior será o rol de possibilidades de programação, seja no sentido de controlar os resultados ou de deixá-los livres, ao acaso, por assim dizer. As saídas dos algoritmos que a partir de agora propomos são, propositalmente, diferentes a cada execução, tanto em timbres, notas e duração geral. Para iniciar, trabalharemos a possibilidade de se criar timbres diferentes, utilizando recursividade. O código segue abaixo (As funções do CSound não estão mais comentadas):

```
1       def create_instr(self, nharm):
2       from random import randint
3         self.orc='''
4         sr = 44100
5         kr = 4410
6         ksmps = 10
7         nchnls = 1
8
9         instr 1
10          kamp = 30000
11          kcps = p4
12          ifn = 1
13          a1 oscil kamp, kcps,ifn
14         out a1
15        endin
16        '''
17        self.ftable = 'f 1 0 16384 10 1'
18        while nharm>0:
19          self.ftable = self.ftable  + ' ' + str(randint(0,100)/100)
20          nharm = nharm - 1
```

Linha 1: passa como padrão a variável *self,* que é obrigatório para o Python e a variável *nharm* que define o número de harmônicos que esse instrumento terá

Linha 2: importa a função randint da biblioteca random, que nos possibilitará utilizar funções randômicas

Linhas 3-16: cria o básico da seção CsInstruments na variável orc. O *self* é utilizado para definir a variável como pertencente a essa função

Linha 17: cria a ftable do instrumento 1, onde serão definidos os harmônicos

Linha 19: define a amplitude de cada harmônico e adiciona à ftable

7. Python, o compositor: gerando uma score

Já criamos um instrumento e mostramos como o Python pode ser nosso "luthier". Agora vamos torná-lo um "compositor", fazendo com que escolha sozinho as notas (consideraremos aqui somente alturas do sistema temperado) e as durações que serão tocadas. Para gerar nossa score, optamos por deixar o tamanho flexível, podendo o usuário definir ou deixar por conta do Python, utilizando uma função randômica. Neste exemplo, utilizamos uma função recursiva para gerar as notas. O código abaixo dá um exemplo disso (novamente retiramos os comentário do CSound):

```
1        def create_score(self, tamanho):
2           from random import randint
3           if tamanho==0:
4              self.sco = self.sco +'/n  e'
5           else:
6              nota = randint(4,13) +   (randint(0,11))/100.0
7              dur = (randint(1,50))/10.0
8              instrumento = randint(1, self.som.getInstrumentCount())
9              self.sco =('/n i ' +     str(instrumento) + ' . ' +      str(dur) + ' ' + str(nota))
10      self.create_score(tamanho-1)
```

Linha 1**:** passa como padrão a variável *self,* que é obrigatório para o Python e a variável *tamanho* que define o número de notas que essa partitura terá.

Linha 2: importa a função randint da biblioteca random

Linha 3-4: se "tamanho" for zero, finaliza a score

Linha 6: define a nota randomicamente

Linha 7: define a duração randomicamente

Linha 8: escolhe o instrumento randomicamente dentre os existentes

Linha 9: adiciona a linha à score

Linha 10: chama a função create score novamente

8. Colocando tudo para funcionar

Com os dois códigos criados, poderemos concentrar todas as funções de criação de instrumento e de score. Na classe abaixo, incluímos também a opção de salvar o arquivo CSD antes da compilação, além da opção de definir as opções.

```
1       class csRandom():
2         import csnd
3         from random import randint
4         var = csnd.CppSound()
5         orc = ' '
6         sco = ' '
7         ftable = ' '
8         def create_instr(self, nharm):
9           from random import randint
10          self.orc='''
11              sr = 44100
12              kr = 4410
13              ksmps = 10
14              nchnls = 1
15
16              instr 1
17              kamp = 30000
18              kcps = p4
19              ifn = 1
20              a1 oscil kamp, kcps,ifn
21              out a1
22             endin
23             '''
24          self.ftable = 'f 1 0 16384 10 1'
25          while nharm>0:
26          self.ftable = self. ftable + ' ' + str(randint(0,   100)/100)
27          nharm = nharm - 1
28
29      def create_score(self, tamanho):
30        if tamanho==0:
31           self.sco = self.sco    +'/n e'
32        else:
33           nota = randint(4,13) + (randint(0,11))/100.0
34           dur = (randint(1,50))  /10.0
35           instrumento = randint(1, self.som.getInstrumentCount())
36           self.sco =('/n i ' +     str(instrumento) + ' . ' +      str(dur) + ' ' + str(nota))
37           self.create_score(tamanho-1)
38
39      def Compile():
40        self.var.setOrchestra(self.orc)
41        self.var.setScore(self.sco)
42        if raw_input('Deseja salvar? ') == 's' or 'S':
```

```
43              local = raw_input('Digite o caminho: ')
44              self.var.save(local)
45
46          op = raw_input('Digite as opções para compilação: ')
47          self.var.setCommand('csound' + op)
48          self.var.exportForPerformance()
49          self.var.Perform()
```

Linha 1: cria uma classe chamada csRandom onde agruparemos todas as funções que utilizaremos para gerar o arquivo CSound

Linha 2: importa a biblioteca *csnd*

Linha 3: importa a função para trabalhar com números randômicos

Linhas 4-7: cria as variáveis de uso global (as que serão usadas por todas as funções)

Linhas 8-27: cria a função *create_instr* que já discutimos anteriormente

Linhas 29-37: cria a função *create_score* também já discutida anteriormente

Linha 39: cria a função *Compile,* que utilizaremos para compilar e ajustar as opções de compilação do arquivo gerado.

Linha 40: define o conteúdo da variável *orc* como a seção CsInstruments

Linha 41: define o conteúdo da variável *sco* como a seção CsScore

Linha 42-44: verifica se o usuário quer salvar o arquivo CSD gerado. Caso a resposta seja afirmativa, pergunta o local e salva o arquivo.

Linha 46: salva na variável *op* as opções de compilação, que serão utilizadas na seção CsOptions

Linha 47: aplica as opções de compilação

Linha 48: prepara o arquivo para compilação

Linha 49: compila o arquivo

Após compilar esse código já poderemos utilizar. Se quisermos criar um novo arquivo, basta criarmos uma variável do tipo csRandom e todas funções que escrevemos estarão disponíveis. Para criar um som completamente randômico, teríamos que escrever o seguinte código:

```
1       from random import randint
2       var = csRandom()
3       var.create_instr(randint(1,100))
4       var.create_score(randint(1,100))
5       var.Compile()
```

Linha 1: importa a função randint da biblioteca random
Linha 2: cria uma variável de nome *var* do tipo *csRandom*
Linha 3: chama a função *create_instr*, passando um número randômico entre 1 e 100 para a variável nharm
Linha 4: chama a função *create_score*, passando um número randômico entre 1 e 100 para a variável *tamanho*
Linha 5: chama a função *Compile*, que salvará e compilará nosso arquivo

Por ter uma característica de ser completamente randômico, cada vez que este algoritmo for executado produzirá um arquivo de saída completamente diferente.

9. Conclusões

Muito pode ser feito com CSound e muito pode ser feito com Python. Ao unir o poder de ambos, pode-se compor coisas que fogem de nossa imaginação, mas nunca de nosso controle. Essas duas ferramentas, quando aliadas, só tendem a facilitar a tarefa do compositor-programador. Pode-se utilizar todos os meios de programação como recursividade, redes neurais, intervenção do usuário, além de meios matemáticos para se gerar música. Por ser extremamente versátil, o Python permite que controlemos parâmetros complexos que teríamos relativa dificuldade em controlar com CSound sozinho. Por ser uma linguagem de alto nível, ou seja, com um alto nível de abstração, Python permite que o gerenciamento de dados complexos se torne simples, além de permitir que uma série de interações complexas entre os dados. Logicamente, quanto mais o programador conhecer a linguagem e as técnicas de programação, mais controle terá sobre a composição. Ainda há muito a pesquisar e muitos projetos a desenvolver em diversas áreas, seja na composição de música aleatória, pesquisa de timbres, desenvolvimentos de linguagens para *real-time* ou ainda integração de outros sistemas físicos, como uma série de sensores, com algoritmos que gerem música.

10. Referências bibliográficas

[1] www.csounds.com/toot/pt/
[2] www.python.org