ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Trabalho Prático

Processamento de Linguagens — 2020/2021

Licenciatura em Engenharia de Sistemas Informáticos

Versão 0.1 — 8 de Dezembro de 2020— Avaliação Contínua

Instruções

- Estes enunciados correspondem ao segundo trabalho prático de Processamento de Linguagens, regimes diurno e pós-laboral, no ano letivo 2020/2021;
- O trabalho prático será realizado em Grupo com um **máximo de 3 alunos**;
- A data limite para a entrega do Trabalho Prático é o **dia 24 de Janeiro**;
- As apresentações dos trabalhos práticos são realizadas por **todos os elementos do grupo**;
- Para além da implementação em Python + Ply do projeto, cada grupo deverá incluir um conjunto de testes que demonstrem as funcionalidades implementadas;
- Deverá ser preparado um pequeno relatório que explique de que forma o enunciado foi interpretado, e quais as decisões tomadas na sua implementação.
- Qualquer detalhe que não esteja claro no enunciado deve ser extrapolado pelos alunos, optando pela interpretação que lhes parecer mais lógica/funcional.

A. Linguagem Musical

Pretende-se implementar um interpretador para uma linguagem que permita descrever melodias simples, que serão posteriormente transformadas em ficheiros MIDI.

mxm.midifile

Esta secção descreve uma forma muito simples de uso da bibioteca mxm.midifile. O seu uso baseia-se num par de eventos: iniciar a reprodução de uma nota, e terminar a reprodução de uma nota.

O código apresentado na listagem 1 gera um excerto da célebre música "Parabéns" usando esta biblioteca.

Neste exemplo definiram-se variáveis para as durações típicas da figuras rítmicas. Para as notas, utilizou-se diretamente a sua frequência. O dó central tem a frequência 60, e para cada meio tom, incrementa-se ou decrementa-se uma unidade à frequência. Assim, por exemplo, a escala de dó maior será representada pelas seguintes frequências: 60, 62, 64, 65, 67, 69, 71, 72.

Linguagem de Programação Musical

O que se pretende é, então, a criação de uma ferramenta que receba músicas num formato textual, que será descrito de seguida. O seu resultado será a geração de um ficheiro MIDI com a música definida.

Os comandos desta linguagem são contextuais, ou seja, cada comando depende do comando anterior, e portanto, dos valores de frequência ou duração usados na nota anterior.

```python
from mxm.midifile import MidiOutFile

out_file = open('file-generated.mid', 'wb')
midi = MidiOutFile(out_file)

midi.header(format=0, nTracks=1, division=32)
midi.start_of_track()

MIDI_WHOLE = 64           # breve
MIDI_MINIM = 32           # mínima
MIDI_CROTCHET = 16        # semínima
MIDI_QUAVER = 8           # colcheia
MIDI_SEMIQUAVER = 4       # semicolcheia

melody = [ (60, MIDI_QUAVER), (60, MIDI_QUAVER), (62, MIDI_CROTCHET), (60, MIDI_CROTCHET),
           (65, MIDI_CROTCHET), (64, MIDI_MINIM), (60, MIDI_QUAVER), (60, MIDI_QUAVER),
           (62, MIDI_CROTCHET), (60, MIDI_CROTCHET), (67, MIDI_CROTCHET), (65, MIDI_MINIM) ]

midi.update_time(0)
for note, duration in melody:
    midi.note_on(channel=0, note=note)
    midi.update_time(duration)
    midi.note_off(channel=0, note=note)
midi.update_time(0)
midi.end_of_track()
```

Listing 1: Uso simples da biblioteca mxm.midifile.

Considera-se que, ao iniciar a interpretação de uma música, o contexto da primeira nota é definido pela nota dó (frequência 60) e pela frequência de uma semínima. Deste modo, qualquer nota que seja reproduzida sem alterações, será um dó, com a dureção de uma semínima.

Os operadores desta linguagem são descritos de seguida:

- O primeiro operador da linguagem é o ponto (`.`) que representa uma nota, no contexto atual. Assim, a expressão "`...`" corresponde a uma sequência de três semínimas, com a nota dó.
- É possível controlar a frequência do contexto, aumentando-a em meio tom, usando o caracter circunflexo (`^`). Note-se que este comando altera a frequência em meio tom, mas não reproduz qualquer nota.

  Para reproduzir a sequência de notas "dó, dó sustenido, ré", terá de ser utilizada a expressão "`.^.^.`". Note que esta expressão não muda a duração das notas.

  Para reproduzir as noas "dó, ré, mi, fá", poder-se-ia utilizar a seguinte expressão: "`.^^.^^.^.`" (note-se o aumento em dois meios tons entre o dó e o ré, e o ré e o mi, mas apenas de meio tom enre o mi e o fá).
- A frequência pode também ser diminuída, usando para isso o carater sublinado (`_`). O funcionamento é semelhante ao operador circunflexo referido anteriormente.
- Para facilitar o aumento de um grande número de meios tons (por exemplo, para reproduzir uma oitava — 12 meios tons) pode utilizar-se um modificador entre chavetas. Assim, uma oitava pode ser reproduzida com: "`.^{12}.`"
- Do mesmo modo que se altera a frequência, também é possível aumentar ou diminuir a velocidade (duração das notas) utilizando os caracteres *menor que* (`<`) e *maior que* (`>`)

  Considerando o contexto inicial (dó, semínima), a sequência "`.<.`" irá reproduzir um dó com a duração de uma semínima, e um dó com a duração de uma mínima (ou seja, aumentará a duração das notas).

  Do mesmo modo, e considerando o contexto inicial, a sequência "`.>.`" irá reproduzir o segundo dó com a duração de uma colcheia (com duração menor).
- Por uma questão de legibilidade, deverá ser possível usar espaços e mudanças de linha, que deverão ser ignorados.

- Qualquer linha que contenha um por cardinal (`#`) deverá ser ignorada a partir desse cardinal até ao final da linha.
- Tal como o ponto corresponde à reprodução de uma nota, um asterisco (`*`) deve ser considerado uma pausa.
- Na reprodução musical nem sempre se usam, apenas, as durações das figuras base, sendo possível a junção de notas, seja com a mesma duração ou durações diferentes.

  Assim, a expressão "`.~.`" corresponde a duas notas seguidas, sem que a primeira termine (basicamente, uma única nota com a duração atual). Também é possível ligar notas com alterações de velocidade: "`. ~> .`" corresponde a uma semínima e uma colcheia ligadas, ou seja, uma única nota cuja duração é equivalente à soma da duração de uma semínima e de uma colcheia.
- O uso do carater *dois pontos* (`:`) deverá reproduzir um acode maior (três notas simultanemanete: a base, referente ao contexto, uma acima da nota base 4 meios tons, e uma outra, acima da nota base 7 meios tons)
- Deverá ser possível definir funções. Por exemplo, a expressão

  "`MAJORSCALE=[.^^.^^.^.^^.^^.^^.^.]`"

  permite definir uma escala maior (que não é reproduzida mas pode ser usada posteriormente). Assim, poderemos reproduzir as escalas de dó maior e de ré maior, sequencialmente, usando a expressão:

  "`\MAJORSCALE ^{2} \MAJORSCALE`"

  Note que o uso da macro não afeta a frequência nem a velocidade do contexto antes do seu uso.
- Deverá ser também possível o uso da notação de letras para cada nota: c, d, e, f, g, a, b, c (escala de dó a dó). Considere que cada letra se refere, sempre, à nota mais próxima. Assim, `cdb` correspondem a `.^{2}._{3}.` (ou seja, o si encontra-se na oitava abaixo do dó inicial).

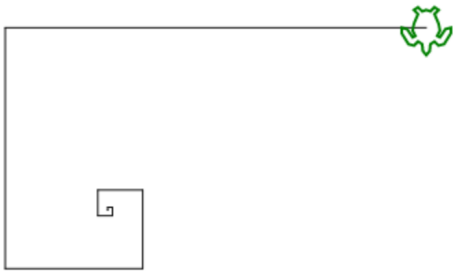
B. Logo

O Logo é uma linguagem de programação educativa, com origens em 1967. Pretende-se implementar um interpretador básico, capaz de simular o maior número possível de comandos da linguagem original. Nesta linguagem o programador controla uma tartaruga, que se vai mexendo através do espaço, e desenhando linhas por onde passa.

Uma vez que o processo de desenho no ecrã não faz parte do conteúdo programático da unidade curricular, o processador implementado deverá reconhecer as instruções apresentadas e gerar a imagem correspondente no formato SVG (Scalable Vector Graphics), que é um standard definido pelo World Wide Web Consortium (w3c). Existe um tutorial disponível em `https://www.w3schools.com/graphics/svg_intro.asp`, que deve ser analisado pelos alunos que optem por este enunciado.

O programa Logo será um ficheiro de texto com uma ou mais linhas, e deve implementar, pelo menos, os seguintes comandos:

- O comando `fd` ou `forward` move a tartaruga n pixeis em frente: `fd 10`
- O comando `bk` ou `back` move a tartaruga n pixeis para trás: `back 20`
- O comando `lt` ou `left` roda a tartaruga, para a esquerda, n graus: `left 90`
- O comando `rt` ou `right` roda a tartaruga, para a direita, n graus: `right 180`
- O comando `setpos`, `setxy`, `setx` e `sety` permitem definir uma posição para a qual a tartaruga se deve movimentar: `setpos [100 100]`, ou `setxy 100 100` ou um eixo apenas, mantendo o outro eixo: `setx -100`
- O comando `home` move a tartaruga para o ponto inicial $(0, 0)$, e roda-a para a orientação original: `home`



- Os comandos `pendown` e `penup`, respetivamente abreviados por `pd` e `pu`, permitem alternar entre o modo de desenho (em que os comandos anteriores desenham uma linha) e o modo de movimentação livre (em que nada é desenhado).
- O comando `setpencolor` permite alterar a cor das linhas para os comandos que se seguirem: `setpencolor [99 0 0]`
- O comando `make` permite definir o valor de uma variável: `make "varname 10`
- O comando `if` e `ifelse` permitem definir estruturas condicionais.
- O comando `repeat` repete um conjunto de comandos.
- O comando `while` permite definir ciclos.
- O comando `to` permite criar funções.

A listagem 2 apresenta um pequeno exemplo de um programa definido em Logo. Note que: ao definir variáveis, os

```
1   make "i 10
2   make "l 2
3   while [ :i > 0 ] [
4       fd :l rt 90
5       make "i :i - 1
6       make "l 1.75 * :l
7   ]
```

Listing 2: Exemplo de programa Logo

seus nomes devem ser precedidos por uma aspa. Ao aceder ao valor de uma variável, o seu nome deve ser precedido por dois pontos. Note também que as mudanças de linha são opcionais, sendo possível escrever um programa Logo, válido, numa única linha.

Em `http://www.calormen.com/jslogo/` existe um interpretador on-line, onde se pode experimentar os vários comandos existentes. Tem, também, uma descrição da sintaxe específica do Logo, pelo que não será aqui detalhada.