MAC0110 — Quarto EP - Data da Entrega: 9/6/2016

Roberto Hirata Jr.

25 de maio de 2016

1 Introdução

Neste quarto EP, o objetivo será implementar, em Python, funções que computam a evolução de um autômato celular e que produzem alguns desenhos de estados desse autômato.

Estamos falando aqui do Jogo da Vida (Conway's Game of Life) proposto inicialmente pelo matemático britânico John Horton Conway. Este jogo, é considerado um jogo para zero jogadores, pois a única interação que terá com o jogo é determinar o seu estado inicial. A partir daí o jogo evolui sozinho durante uma quantidade indeterminada de iterações.

Na Seção 2 explicamos o funcionamento do jogo e o que deverão fazer. Na Seção 3 deixamos as especificações de cada função que o programa de vocês deve ter. Adicionalmente vocês poderão também realizar algumas tarefas extras que valerão alguns pontos a mais na média de EPs. Tais tarefas estão descritas na Seção 5.

2 Jogo da vida

O jogo da vida consiste na simulação de um autômato celular (`http://mathworld.wolfram.com/CellularAutomaton.html`). Este autômato é, em geral, composto de uma grade bidimensional, de dimensões infinitas, na qual cada célula pode assumir um conjunto finito de estados e um conjunto de regras para a mudança de estado. No caso do autômato que vocês irão construir, vamos considerar duas diferentes variações.

2.1 Versão clássica

Inicialmente vocês devem implementar a variante original do jogo da vida, na qual cada célula da grade possui oito células vizinhas (células são quadradas e seus vizinhos são as células verticalmente, horizontalmente e diagonalmente adjacentes) e dois estados possíveis: vivo ou morto. A cada iteração, o estado de uma célula pode mudar de acordo com as seguintes regras:

- Uma célula nasce (seu estado muda de morta para viva) se ela possui três vizinhos vivos (dizemos que as células se reproduzem).

- Uma célula viva sobrevive (mantém seu estado de viva) se possui dois ou três vizinhos vivos.
- Nos demais casos as células vivas morrem e as mortas mantém-se mortas.

Essas regras são representadas pelo código `N3S23` (**N**ascem se 3 vizinhos estão vivos, **S**obrevive se 2 ou 3 vizinhos estão vivos).

2.2 Versão hexagonal

Nessa versão temos duas mudanças em relação à versão clássica. As células são hexagonais, cada célula é um hexágono e, logo, possui seis vizinhos. As regras para esse caso devem ser: `N35S2`.

3 Especificação das funções

Vocês devem implementar as seguintes funções:

- `leEntrada(nome)`: Esta função deve receber como entrada o nome de um arquivo que conterá uma configuração inicial para uma simulação do jogo da vida, ler o arquivo e retornar os seguintes itens (nesta ordem):
  - um número indicando o tipo de grade (0 = grade quadrada, 1 = grade hexagonal)
  - uma lista de posições (par ordenado) de células vivas.
- `simulaQuad(n,m,lista,t)`: Esta função deve receber como entrada o número n de linhas e m de colunas da grade, uma lista com posições (pares ordenados) de células inicialmente vivas e uma quantidade t de iterações a serem realizadas pela simulação do jogo da vida com células quadradas. Esta função deve retornar uma lista de posições de células vivas após t iterações da simulação.
- `simulaHex(n,m,lista,t)`: Esta função deve receber como entrada o número n de linhas e m de colunas da grade, uma lista com posições (pares ordenados) de células inicialmente vivas e uma quantidade t de iterações a serem realizadas pela simulação do jogo da vida com células hexagonais. Esta função deve retornar uma lista de posições de células vivas após t iterações da simulação.
- `desenhaQuad(n,m,lista,figura)`: Esta função deve receber como entrada o número n de linhas e m de colunas da grade, uma lista com posições (pares ordenados) de células vivas e o nome $figura$ de um arquivo e deve gerar o desenho de um estado do jogo da vida clássico com as posições vivas informadas e salvá-lo no arquivo de nome $figura$, no formato PNG.

- `desenhaHex(n,m,lista,figura)`: Esta função deve receber como entrada o número n de linhas e m de colunas da grade, uma lista com posições (pares ordenados) de células vivas e o nome $figura$ de um arquivo e deve gerar o desenho de um estado do jogo da vida hexagonal com as posições vivas informadas e salvá-lo no arquivo de nome $figura$, no formato PNG.

4 Precisões

Explicarei aqui alguns detalhes importantes que não foram mencionados antes.

Embora as grades nesses jogos serem teoricamente infinitas vocês implementarão grades de dimensões finitas. Porém essas grades devem ser de tal forma que as células nas bordas terão como vizinhos células nas bordas opostas. Por exemplo, digamos que temos uma grade quadrada de tamanho 10×10. Os vizinhos da posição $[0, 0]$ serão as células $[1, 0]$, $[0, 1]$, $[1, 1]$, $[0, 9]$, $[1, 9]$, $[9, 0]$ e $[9, 9]$. No caso hexagonal identificar os vizinhos das bordas é menos trivial. Nos casos em que m é ímpar deve-se supor que existe uma coluna a mais, invisível que faz a ponte entre as extremidades esquerda e direita.

O arquivo de entrada deve ser um arquivo de texto com a primeira linha sendo um caractere 'Q' para identificar grades quadradas e 'H' para identificar grades hexagonais e nas demais linhas, dois números inteiros por linha, separados por uma vírgula, representando as células vivas.

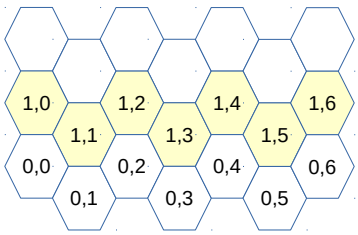
```
Q
3,4
3,3
3,5
4,4
```

Para evitar confusões, observem a Figura 1 ilustrando a numeração das posições em grades quadradas e hexagonais.

Finalmente, se fizerem uma busca online ou olharem o artigo do Wikipedia sobre o assunto, notaram que há alguns algoritmos desenvolvidos para simular o jogo da vida de forma bastante eficiente. Aqui o importante é que tenham uma versão feita por vocês que funcione, independentemente da quantidade de memória ou processamento que ela possa consumir.

5 Extras

Deixo aqui duas tarefas extras que valerão 2 e 1 pontos extras cada, respectivamente. Lembro que essas tarefas são completamente opcionais.



1,0	1,1	1,2	1,3	1,4	1,5	1,6
0,0	0,1	0,2	0,3	0,4	0,5	0,6

Figura 1: Ilustração de como numerar as células no caso clássico e hexagonal.

5.1 Regras genéricas

A primeira tarefa extra é a de implementar funções que recebem como parâmetro as regras para as simulações de ambas versões do jogo da vida tratadas neste exercício programa. As funções devem ser as seguintes:

- `simulaQuadGenerica(n,m,lista,t,b,s)`: Esta função deve receber como entrada o número n de linhas e m de colunas da grade, uma lista com posições (pares ordenados) de células inicialmente vivas e uma quantidade t de iterações a serem realizadas pela simulação do jogo da vida com células **quadradas** (clássico). Recebe também os valores b e s que definem a regra `NbSs` para a simulação. Esta função deve retornar uma lista de posições de células vivas após t iterações da simulação.

- `simulaHexGenerica(n,m,lista,t,b,s)`: Similar a anterior, mas considerando a simulação do jogo da vida com células **hexagonais**.

Notem que se decidirem implementar essas funções, elas podem perfeitamente ser utilizadas para implementar as funções `simulaQuad` e `simulaHex`.

5.2 Identificando repetições

O termo oscilador, no contexto de autômatos celulares, caracteriza padrões que retornam ao seu estado inicial. A tarefa aqui seria um pouco mais simples, mas relacionada com o conceito de osciladores. Vocês devem aqui escrever uma função que identifica, no caso do jogo da vida

clássico, se algum padrão se repete pelo menos uma vez em uma quantidade fixa de iterações. Vocês devem escrever a seguinte função:

- `haRepeticoes(n,m,lista,t)`: Esta função deve receber como entrada o número n de linhas e m de colunas da grade, uma lista com posições (pares ordenados) de células inicialmente vivas e uma quantidade t de iterações a serem realizadas pela simulação do jogo da vida com células quadradas. Esta função deve retornar `True` se algum padrão (incluindo o inicial) gerado durante as t repetições se repete pelo menos uma vez ou `False` em caso contrário.

6 Entrega

A entrega consiste de um arquivo comprimido em formato `zip` com os arquivos `.py`

7 Tutorial básico

Abaixo segue um tutorial simples de como abrir e ler arquivos em python. Para mais detalhes vejam o link a respeito na seção 9.

```
# Forma tradicional de ler um arquivo
f = open("entrada","r")
for linha in f :
    print(linha)

f.close()

# Aqui, usando  "with"
with open("entrada","r") as f:
    for linha in f :
        print(linha)
```

8 Plágio

Plágio é a cópia/modificação não autorizada e/ou sem o conhecimento do autor original. O plágio é um problema grave que pode levar até a expulsão do aluno da universidade. Para quaisquer dúvidas, consulte o texto que disponibilizamos para uma disciplina irmã desta (Plagio).

9 Referências

Alguns links que podem ser úteis/interessantes:

- **Documentação de como lidar com arquivos em Python (EN):** https://docs.python.org/3/tutorial/inputoutput.html#reading-and-writing-files
- **Artigo sobre Conway's Game of Life no Wikipedia (EN):** https://en.wikipedia.org/wiki/Conway's_Game_of_Life
- **Applet de um jogo da vida clássico:** http://www.bitstorm.org/gameoflife.
- **Artigo Wikipedia sobre osciladores:** https://en.wikipedia.org/wiki/Oscillator_%28cellular_automaton%29