

Programação Funcional

Aula 10 — O jogo *Life*

Pedro Vasconcelos
DCC/FCUP

2022

O jogo *Life*

- ▶ Um *autômato celular* inventado em 1970 pelo matemático John H. Conway.
- ▶ O jogo desenrola-se numa grelha bi-dimensional
- ▶ Cada posição está *vazia* ou contém uma *célula*
- ▶ A colónia de células evolui por *gerações*
- ▶ Determinamos uma nova geração pelas seguintes regras:
 - ▶ uma célula com menos do que 2 ou mais do que 3 vizinhos morre
 - ▶ uma célula com 2 ou 3 vizinhos sobrevive
 - ▶ nasce uma nova célula em cada posição vazia com exactamente 3 vizinhos

https://pt.wikipedia.org/wiki/Jogo_da_vida

Objetivo

Um programa que:

- ▶ visualiza a colónia de células no terminal
- ▶ simula a passagem de gerações

Baseado na solução do livro *Programming in Haskell* de Graham Hutton (capítulo 9).

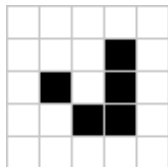
Representação do jogo

Vamos representar a colónia de células por uma **lista de coordenadas**:

```
type Pos = (Int,Int)           -- coluna, linha  
  
type Cells = [Pos]            -- lista de coordenadas
```

Exemplo: um *glider*.

```
glider :: Cells  
glider = [(4,2),(2,3),(4,3),(3,4),(4,4)]
```



Representação do jogo (cont.)

Para facilitar a visualização:

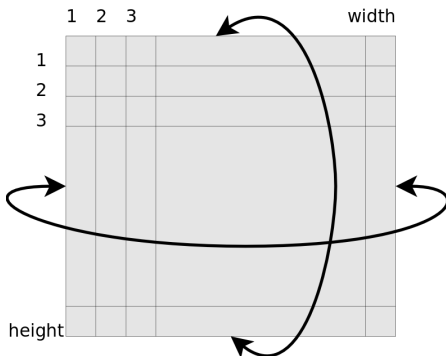
- ▶ largura e altura limitadas

```
width, height :: Int
```

```
width = 80
```

```
height = 24
```

- ▶ lados esquerdo/direito e de topo/baixo são ligados



Algumas funções auxiliares

Testar se uma posição tem uma célula ou está vazia:

```
isAlive, isEmpty :: Cells -> Pos -> Bool
isAlive ps p = elem p ps
isEmpty ps p = not (isAlive ps p)

-- elem :: Eq a => a -> [a] -> Bool
-- testa se um elemento ocorre numa lista
```

Algumas funções auxiliares (cont.)

Obter as 8 posições vizinhas de uma dada coordenada:

```
neighbors :: Pos -> [Pos]
neighbors (x,y) = map wrap [(x-1,y-1), (x,y-1),
                             (x+1,y-1), (x-1,y),
                             (x+1,y)   , (x-1,y+1),
                             (x,y+1)   , (x+1,y+1)]

-- garantir que uma posição está dentro dos limites
wrap :: Pos -> Pos
wrap (x,y) = ((x-1) `mod` width + 1,
              (y-1) `mod` height + 1)
```

Algumas funções auxiliares (cont.)

Contar células vivas entre as posições vizinhas de uma coordenada:

```
liveneighbs :: Cells -> Pos -> Int  
liveneighbs ps = length . filter (isAlive ps) . neighbs
```


Transição entre gerações

A nova geração depende apenas da geração atual. Assim, vamos definir uma **função de transição** entre gerações.

As novas células são as *sobreviventes* mais os *nascimentos*:

```
nextgen :: Cells -> Cells
nextgen ps = survivors ps ++ births ps
```

Falta definir as duas funções auxiliares:

```
survivors, births :: Cells -> Cells
```

Transição entre gerações (cont.)

```
-- as células com 2 ou 3 vizinhos sobrevivem
survivors :: Cells -> Cells
survivors ps
  = [p | p<-ps, elem (liveneighbs ps p) [2,3]]

-- nascem novas células
-- nas posições vazias com 3 vizinhos
-- nub :: Eq a => [a] -> [a] remove repetidos
births :: Cells -> Cells
births ps
  = [p | p<-nub (concat (map neighbs ps)),
      isEmpty ps p,
      liveneighbs ps p == 3]
```

Visualização

Uma função para fazer a animação de n gerações da colónia partindo duma configuração inicial.

```
life :: Cells -> Int -> IO ()
life ps n
  | n>0 = do cls
            printCells ps
            wait 500
            life (nextgen ps) (n-1)
  | otherwise = return ()
```

Esta função não devolve um resultado útil — o objetivo é fazer simulação da passagem de gerações.

Visualização (cont.)

Funções auxiliares:

```
cls :: IO ()           -- limpar o terminal
printCells :: Cells -> IO ()  -- mostrar a colônia
wait :: Int -> IO ()      -- esperar um certo tempo (ms)
```

- ▶ Usamos sequências de escape ANSI para limpar o terminal e posicionar texto
https://en.wikipedia.org/wiki/ANSI_escape_code
- ▶ E a função `usleep` do *standard* POSIX para pausar a execução

Sumário

- ▶ Uma implementação simples do jogo do vida de Conway
- ▶ A separação entre funções de **computação** e **interação** é patente nos tipos

```
liveneighbs :: Cells -> Pos -> Int      -- computação
```

```
nextgen :: Cells -> Cells              -- computação
```

```
printCells :: Cells -> IO ()           -- interação
```

```
life :: Cells -> Int -> IO ()          -- interação
```

- ▶ Facilita a compreensão e extensão do programa

Extras

- ▶ Ler a configuração inicial da entrada padrão
- ▶ Melhorar a visualização usando símbolos Unicode e cores
- ▶ Configuração inicial aleatória usando `randomRIO`
- ▶ Variações das regras (ver https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life)