TP2 - Trabalho 2

14 de novembro de 2022

André Oliveira Barbosa - A91684 Francisco Antonio Borges Paulino - A91666

Caso de estudo

O objetivo deste problema é modificar as regras do autómato do Conway's Game of Life da seguinte forma:

1. O espaço de estados é finito definido por uma grelha de células booleanas (morta=0/viva=1) de dimensão $N \times N$ (com $N > 3$) identificadas por índices $(i, j) \in \{1..N\}$. Estas N^2 células são aqui referidas como "normais".
2. No estado inicial todas as células normais estão mortas excepto um quadrado 3×3, designado por "centro", aleatoriamente posicionado formado apenas por células vivas.
3. Adicionalmente existem $2\,N + 1$ "células da borda" que correspondem a um dos índices, i ou j, ser zero. As células da borda têm valores constantes que, no estado inicial, são gerados aleatoriamente com uma probabilidade ρ de estarem vivas.
4. As células normais o autómato modificam o estado de acordo com a regra "B3/S23": i.e. a célula nasce (passa de 0 a 1) se tem exatamente 3 vizinhos vivos e sobrevive (mantém-se viva) se o número de vizinhos vivos é 2 ou 3, caso contrário morre ou continua morta.

Análise do Problema

Procura-se:

1. Construir uma máquina de estados finita que represente este autómato; são parâmetros do problema os parâmetros N,p e a posição do "centro".
2. Verificar se se conseguem provar as seguintes propriedades: a) Todos os estados acessíveis contém pelo menos uma célula viva. b) Toda a célula normal está viva pelo menos uma vez em algum estado acessível.

Inicialização

In [30]:

```
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import clear_output
from random import randint
import random
```

Implementação

Geração de Matriz bidimensional

In [31]:

```
Mlinhas = 100

Mcolunas = 100

matriz = np.zeros([Mlinhas, Mcolunas])
```

Geração das condições iniciais

In [32]:

```
np.random.seed(100)


NC_vivas = 9 #quadrado 3x3

# selecionar linhas e colunas aleatoriamente
linha_seeds = np.random.randint(0, matriz.shape[0])
col_seeds = np.random.randint(0, matriz.shape[1])
print(linha_seeds)
print(col_seeds)

l_inicial=linha_seeds
c_inicial=col_seeds
c= NC_vivas
comp_linha=3

#formar o quadrado 3x3 com celulas vivas
while (c):
    if(comp_linha>0):
        matriz[l_inicial,c_inicial] = 1
        l_inicial = l_inicial + 1
        comp_linha= comp_linha - 1
        c=c-1
    else:
        l_inicial=linha_seeds
        c_inicial=c_inicial + 1
        comp_linha = 3
```

```
8
24
```

Células da Borda

Probabilidade = p = 60%

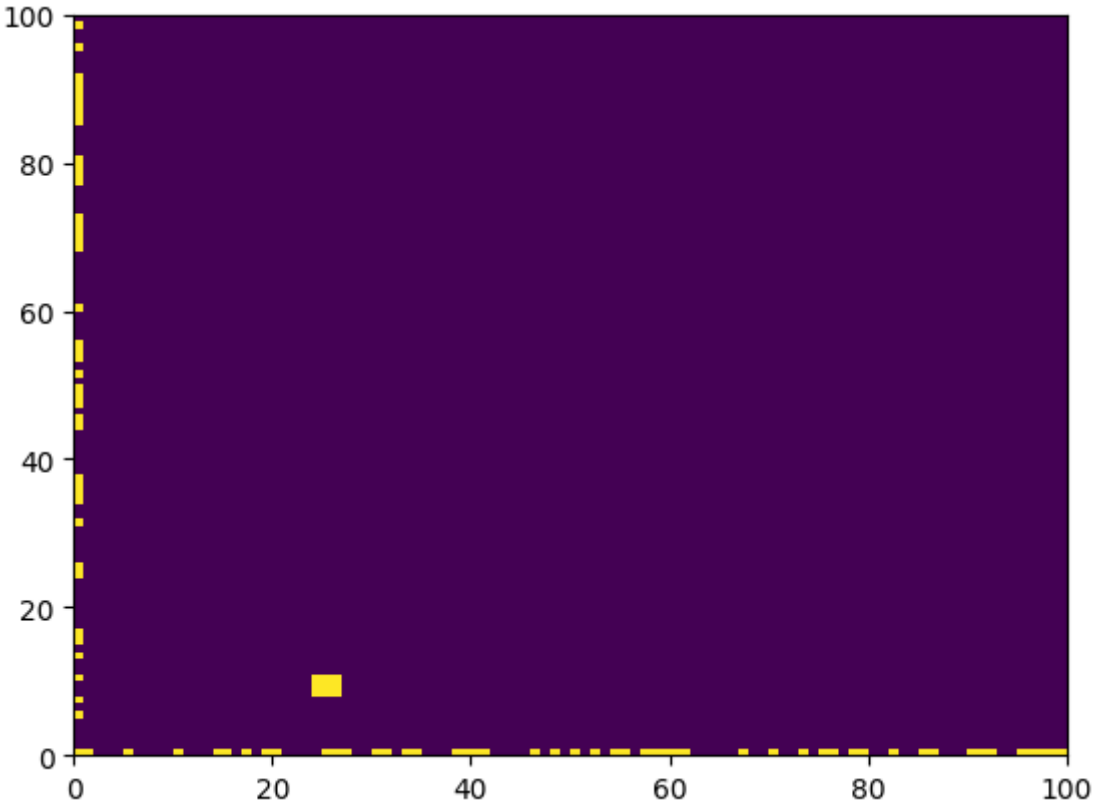
In [33]:

```
n_Cborda = 2*100+1 # 2N+1 células da borda
prob_ro=60
ctr=0
while (n_Cborda>0):
    i = random.randint(0, 1)
    if (i==0):
        j=random.randint(0, matriz.shape[1]-1)
    else:
        j=0
        i=random.randint(0, matriz.shape[1]-1)

    if random.randint(0,100) < prob_ro:
        matriz[i,j] = 1
        print("-  ", i,j)
        ctr=ctr+1
    n_Cborda=n_Cborda-1
print("\nNúmero de celulas da borda vivas:", ctr)
```

```
-   0 54
-   35 0
-   37 0
-   87 0
-   0 33
-   90 0
-   0 34
-   0 97
-   88 0
-   98 0
-   0 41
-   7 0
-   48 0
-   0 61
-   0 26
-   47 0

Número de celulas da borda vivas: 125
```

In [34]:

```
#verificação das condiçoes iniciais
plt.pcolormesh(matriz)
plt.axis('on')
plt.show()
```



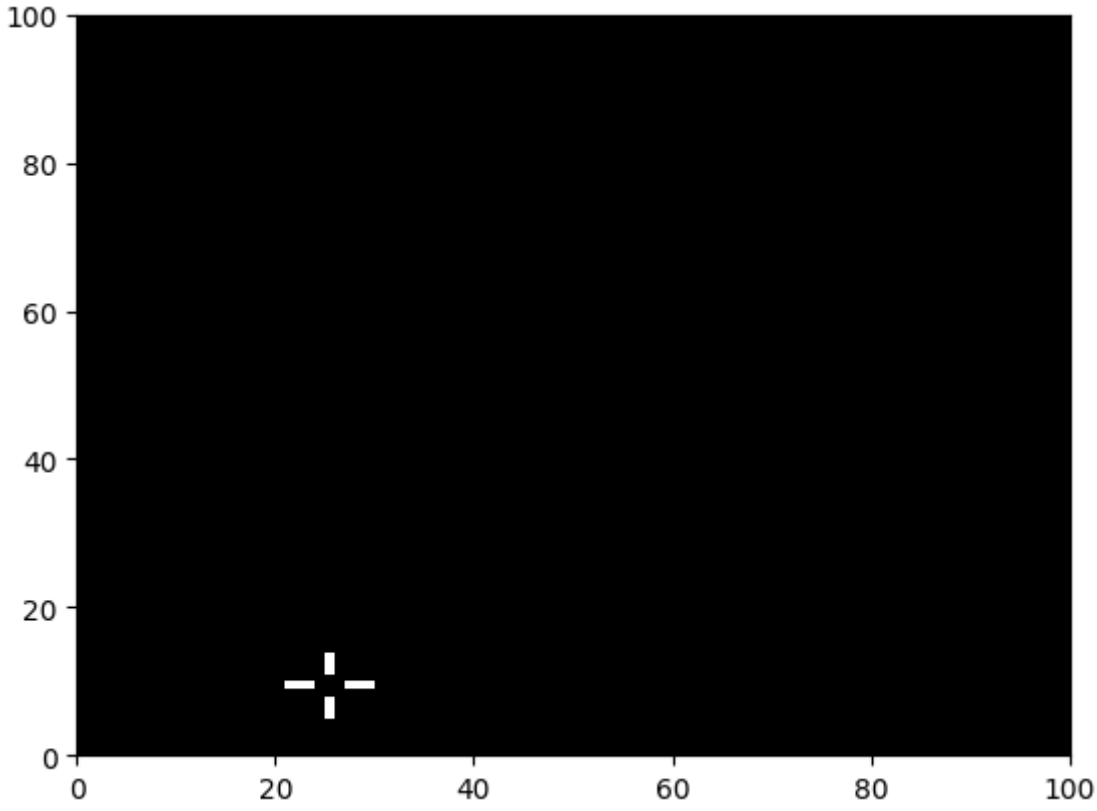
In [35]:

```
#células vizinhas
viz_linhas = np.array([-1, -1, -1, 0, 0, 1, 1, 1])
viz_cols = np.array([-1, 0, 1, -1, 1, -1, 0, 1])
```

Definição das regras

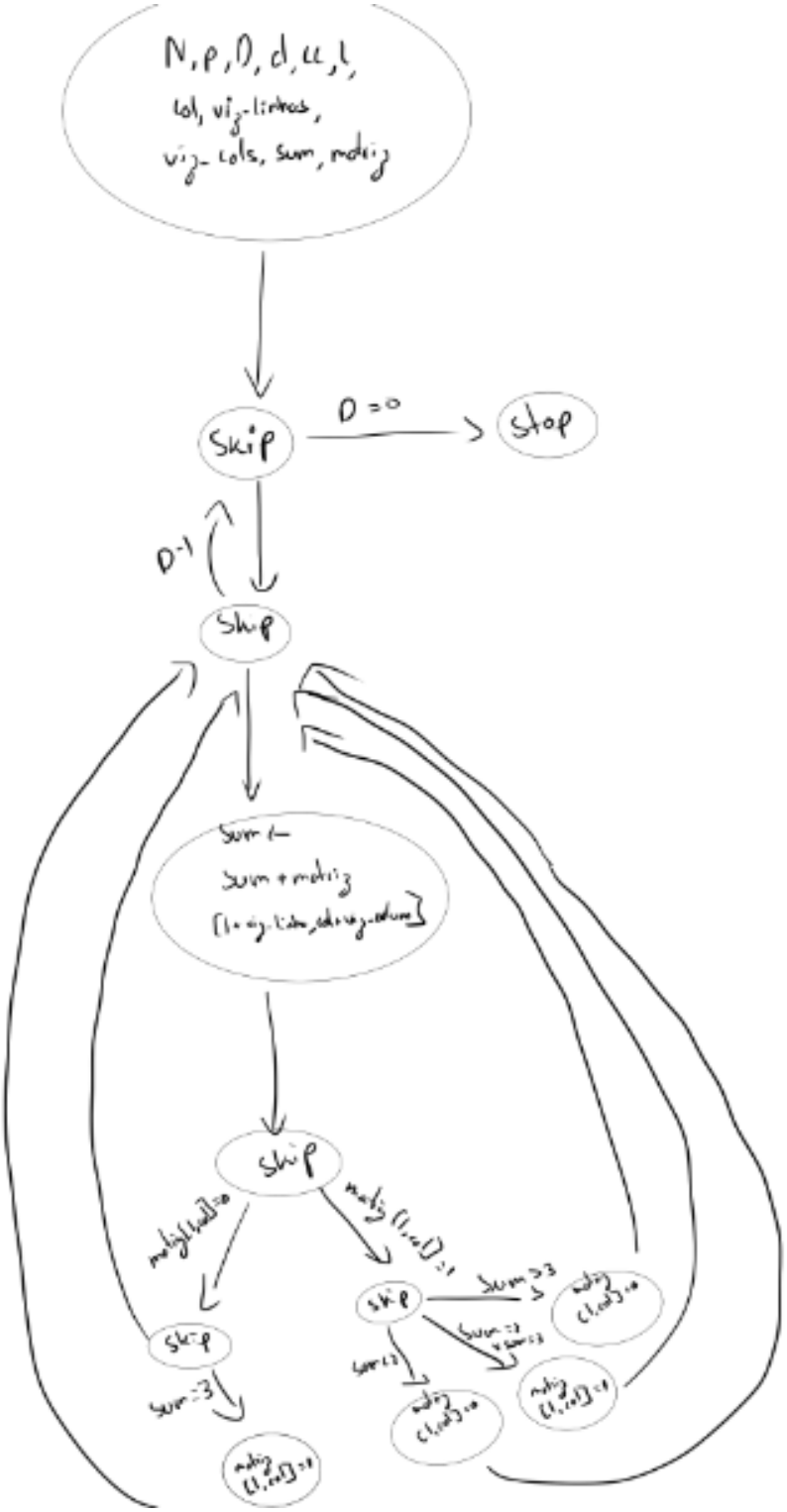
In [*]:

```
dias = 100


while(dias): #dias>0

    clear_output(wait=True)
    plt.pcolormesh(matriz, cmap='gray')
    plt.show()

    M_nova = np.zeros([Mlinhas,Mcolunas]) #guarda os resultados para a geração seguinte

    for l in range(1, matriz.shape[0]-1):
        for col in range(1, matriz.shape[1]-1):

            #soma dos vizinhos de forma a conseguir verificar a geração seguinte
            soma = matriz[l + viz_linhas, col + viz_cols].sum()

            # célula atual vive ou morre?
            if matriz[l,col] == 1:

                # menos de 2 vizinhos = morte
                if  soma < 2:
                    M_nova[l,col] = 0

                #2 ou 3 vizinhos = vive
                elif soma == 2 or soma == 3:
                    M_nova[l,col] = 1

                #mais de 3 vizinhos= morte
                elif soma > 3:
                    M_nova[l,col] = 0
            else:

                # 3 vizinhos torna-se viva
                if soma == 3:
                    M_nova[l,col] = 1
    #diminuir dias no loop
    dias=dias-1

    #proxima geração
    matriz = M_nova
    #print(dias)
```



Criação do automato de Conway's Game of Life

O autómato:



In [*]:

```
from pysmt.shortcuts import *
from pysmt.typing import INT
```

O estado dos FOTS sera um conjunto de inteiros contendo o valor pc, o segundo o valor da variavel x, o terceiro o valor da variavel y e o quarto do valor da variavel z. O estado inicial é caracterizado pelo seguinte perdicado:

$$pc = 0 \wedge N = 100 \wedge \rho = 60\% \wedge D = dias \wedge cl = linha\_seeds \wedge cc = col\_seeds \wedge l = 1 \wedge col = 1 \wedge vi$$
$$\wedge viz\_cols = [-1, 0, 1, -1, 1, -1, 0, 1] \wedge sum = 0 \wedge matriz = [N, N$$

As transições possiveis no FOTS para o programa em questão são:

$$(pc = 0 \wedge pc' = 1 \wedge N' = N \wedge \rho' = \rho \wedge D' = D \wedge cl' = cl \wedge cc' = cc \wedge l' = l \wedge col' = col \wedge viz\_linha$$
$$\wedge sum' = sum \wedge matriz' = matriz)$$
$$\vee$$
$$(pc = 1 \wedge D! = 0 \wedge pc' = 2 \wedge D' = D \wedge cl' = cl \wedge cc' = cc \wedge l' = l \wedge col' = col \wedge viz\_linhas' = viz\_l$$
$$= sum \wedge matriz' = matriz)$$
$$\vee$$
$$(pc = 1 \wedge D = 0 \wedge pc' = 11 \wedge N' = N \wedge \rho' = \rho \wedge D' = D \wedge cl' = cl \wedge cc' = cc \wedge l' = l \wedge col' = col \wedge$$
$$= viz\_cols \wedge sum' = sum \wedge matriz' = matriz)$$
$$\vee$$
$$(pc = 2 \wedge pc' = 3 \wedge D' = D \wedge cl' = cl \wedge cc' = cc \wedge l' = l \wedge col' = col \wedge viz\_linhas' = viz\_linhas \wedge v$$
$$\wedge matriz' = matriz)$$
$$\vee$$
$$(pc = 3 \wedge pc' = 4 \wedge D' = D \wedge cl' = cl \wedge cc' = cc \wedge l' = l \wedge col' = col \wedge viz\_linhas' = viz\_linhas \wedge v$$
$$+ matriz[l + viz\_linhas, col + viz\_cols] \wedge matriz' = matriz)$$
$$\vee$$
$$(pc = 4 \wedge matriz[l, col] = 1 \wedge pc' = 5 \wedge D' = D \wedge cl' = cl \wedge cc' = cc \wedge l' = l \wedge col' = col \wedge viz\_linha$$
$$\wedge sum' = sum \wedge matriz' = matriz)$$
$$\vee$$
$$(pc = 4 \wedge matriz[l, col] = 0 \wedge pc' = 9 \wedge D' = D \wedge cl' = cl \wedge cc' = cc \wedge l' = l \wedge col' = col \wedge viz\_linha$$
$$\wedge sum' = sum \wedge matriz' = matriz)$$
$$\vee$$
$$(pc = 5 \wedge sum < 2 \wedge pc' = 6 \wedge D' = D \wedge cl' = cl \wedge cc' = cc \wedge l' = l \wedge col' = col \wedge viz\_linhas' = viz\_$$
$$= sum \wedge matriz' = matriz[l, col] = 0)$$
$$\vee$$
$$(pc = 5 \wedge (sum = 2 \vee sum = 3) \wedge pc' = 7 \wedge D' = D \wedge cl' = cl \wedge cc' = cc \wedge l' = l \wedge col' = col \wedge viz\_li$$
$$= viz\_cols \wedge sum' = sum \wedge matriz' = matriz[l, col] = 1)$$
$$\vee$$
$$(pc = 5 \wedge sum > 32 \wedge pc' = 8 \wedge D' = D \wedge cl' = cl \wedge cc' = cc \wedge l' = l \wedge col' = col \wedge viz\_linhas' = viz$$
$$= sum \wedge matriz' = matriz[l, col] = 0)$$
$$\vee$$
$$(pc = 6 \wedge pc' = 2 \wedge D' = D \wedge cl' = cl \wedge cc' = cc \wedge l' = l + 1 \wedge col' = col + 1 \wedge viz\_linhas' = viz\_lin$$
$$\wedge matriz' = matriz)$$
$$\vee$$
$$(pc = 7 \wedge pc' = 2 \wedge D' = D \wedge cl' = cl \wedge cc' = cc \wedge l' = l + 1 \wedge col' = col + 1 \wedge viz\_linhas' = viz\_lin$$
$$\wedge matriz' = matriz)$$
$$\vee$$
$$(pc = 8 \wedge pc' = 2 \wedge D' = D \wedge cl' = cl \wedge cc' = cc \wedge l' = l + 1 \wedge col' = col + 1 \wedge viz\_linhas' = viz\_lin$$
$$\wedge matriz' = matriz)$$
$$\vee$$
$$(pc = 9 \wedge sum < 3 \wedge pc' = 2 \wedge D' = D \wedge cl' = cl \wedge cc' = cc \wedge l' = l + 1 \wedge col' = col + 1 \wedge viz\_linhas$$
$$\wedge sum' = 0 \wedge matriz' = matriz)$$
$$\vee$$
$$(pc = 9 \wedge sum = 3 \wedge pc' = 10 \wedge D' = D \wedge cl' = cl \wedge cc' = cc \wedge l' = l \wedge col' = col \wedge viz\_linhas' = viz$$
$$= sum \wedge matriz' = matriz[l, col] = 1)$$
$$\vee$$
$$(pc = 10 \wedge pc' = 2 \wedge D' = D \wedge cl' = cl \wedge cc' = cc \wedge l' = l + 1 \wedge col' = col + 1 \wedge viz\_linhas' = viz\_l$$
$$\wedge matriz' = matriz)$$
$$\vee$$

$$(pc = 2 \wedge (l > 100 \vee col > 100) \wedge pc' = 1 \wedge D' = D - 1 \wedge cl' = cl \wedge cc' = cc \wedge l' = l \wedge col' = col \wedge v$$
$$= viz\_cols \wedge sum' = sum \wedge matriz' = matriz)$$
$$\vee$$
$$(pc = 11 \wedge pc' = 11 \wedge N' = N \wedge \rho' = \rho \wedge D' = D \wedge cl' = cl \wedge cc' = cc \wedge l' = l \wedge col' = col \wedge viz\_lin$$
$$= viz\_cols \wedge sum' = sum \wedge matriz' = matriz)$$
$$\vee$$

In [*]:

```
def declare(i):
    state = {}
    state['pc'] = Symbol('pc'+str(i),INT)
    state['N'] = Symbol('N'+str(i),INT)
    state['ρ'] = Symbol('ρ'+str(i),INT)
    state['D'] = Symbol('D'+str(i),INT)
    state['cl'] = Symbol('cl'+str(i),INT)
    state['cc'] = Symbol('cc'+str(i),INT)
    state['l'] = Symbol('l'+str(i),INT)
    state['col'] = Symbol('col'+str(i),INT)
    state['viz_linhas'] = Symbol('viz_linhas'+str(i),INT)
    state['viz_cols'] = Symbol('viz_cols'+str(i),INT)
    state['sum'] = Symbol('sum'+str(i),INT)
    state['matriz'] = Symbol('matriz'+str(i),Array)

    return state
```

In [*]:

```
def init(state):
    return And(Equals(state['pc'], Int(0)), Equals(state['N'], Int(100)),
               Equals(state['ρ'], Int(60)), Equals(state['D'], Int(100)),
               Equals(state['cl'], Int(8)),Equals(state['cc'], Int(24)),
               Equals(state['l'], Int(1)),Equals(state['col'], Int(1)),
               Equals(state['viz_linhas'], Array([-1, -1, -1, 0, 0, 1, 1, 1])),
               Equals(state['viz_cols'], Array([-1, 0, 1, -1, 1, -1, 0, 1])), Equals(state[
               Equals(state['matriz'],Array([100,100])))
```

In [ ]: