Python - Jogo da Vida

Orientador: Prof. Bernardo M. Rocha

Bolsista: Agilian Lucas

Objetivos:

- Aprender Python:

    * Como: implementação de um Autômato Celular;

    * Finalidade: Trabalhar com um Software de simulação de células cardíacas escrito em Python;

O que é um Autômato Celular?

- Consiste de uma grelha infinita e regular de *células*, cada uma podendo estar em um número finito de *estados*, que variam de acordo com regras determinísticas. O estado de uma célula no tempo *t* é uma função do estado no tempo *t-1* de um número finito de células na sua *vizinhança*. Cada vez que as regras são aplicadas à grelha completa, uma nova *geração* é produzida.

- Ex.:

Exemplos de autômatos celulares:

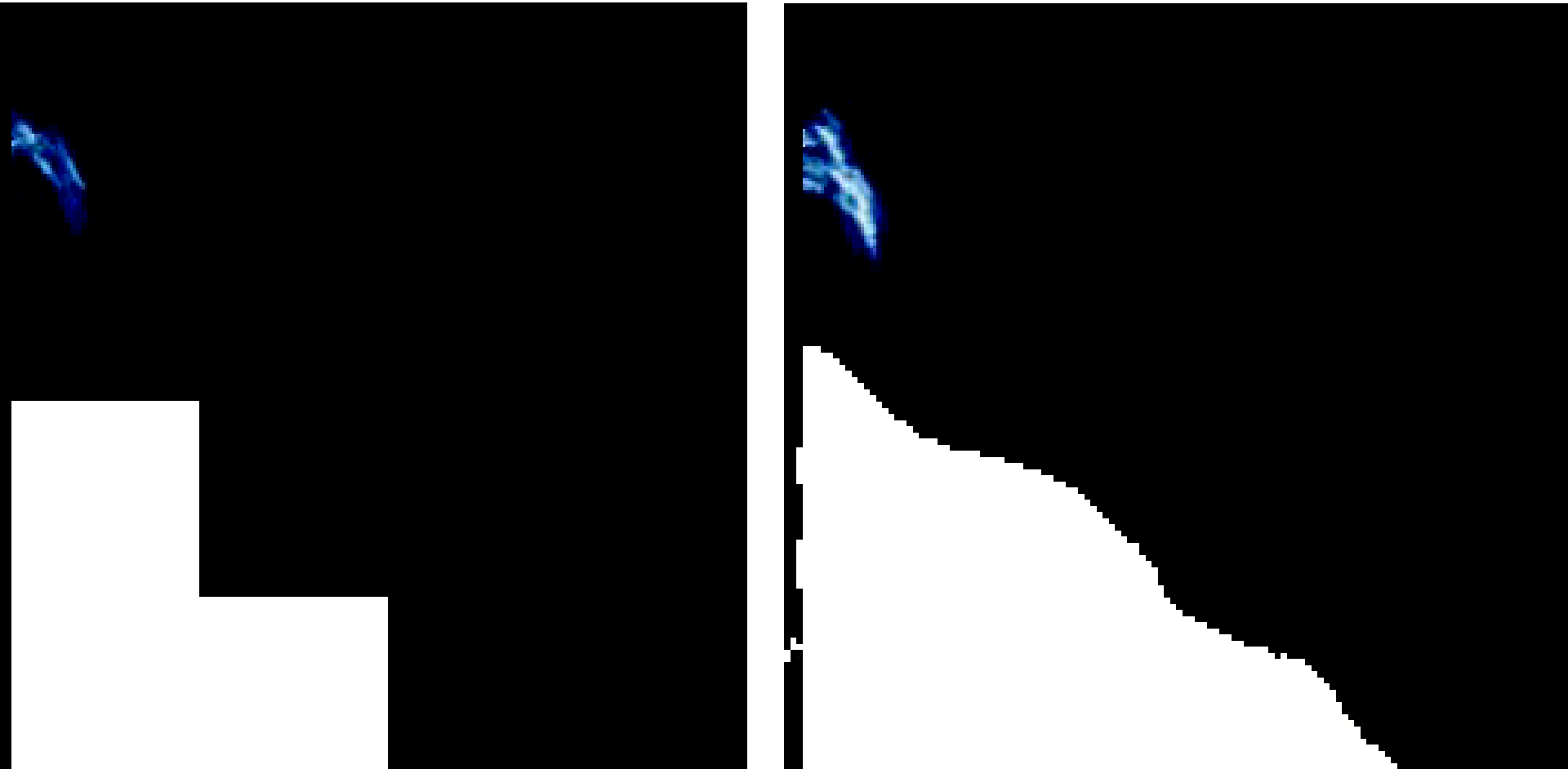
- Simulação de Fogo:

Fogo

Fogo Com Vento

Exemplos de autômatos celulares:

- Simulação de Queda d’água:

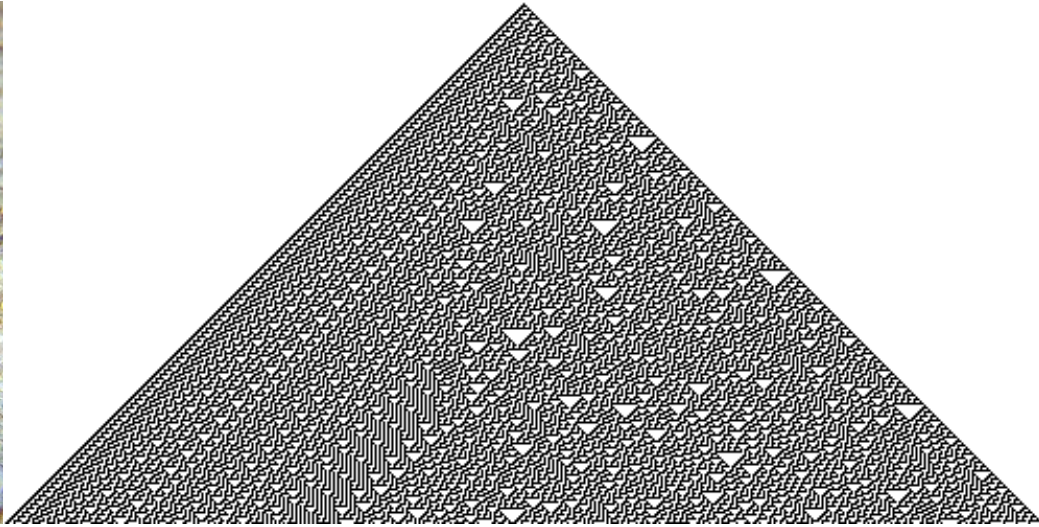


Exemplos de autômatos celulares:

- Desenhos da natureza:

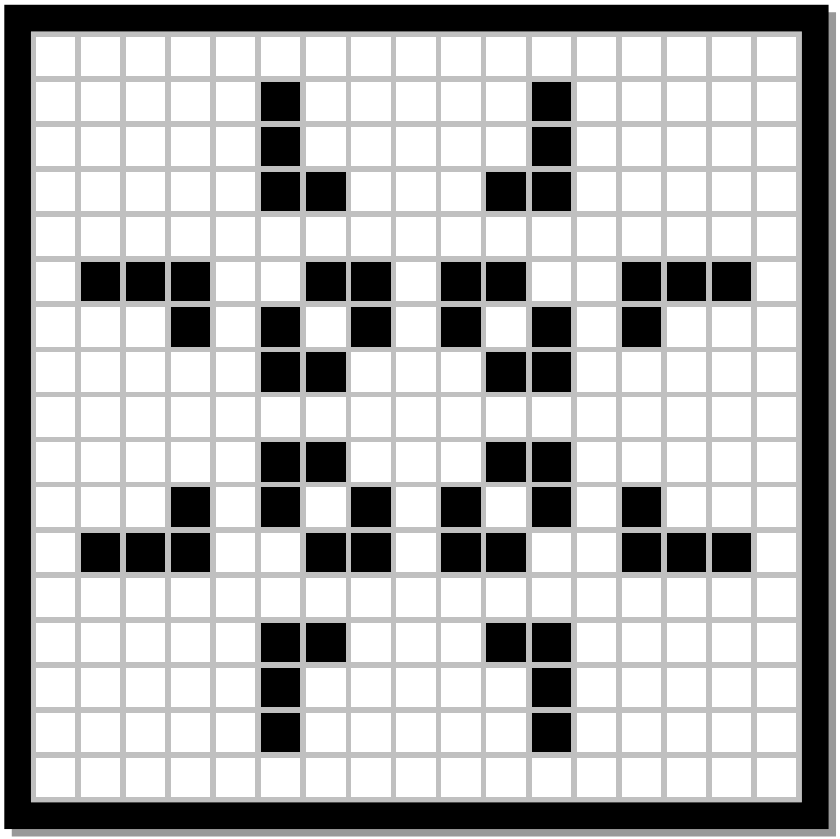
Conus textile:

Rule 30(Regra 30)*:*



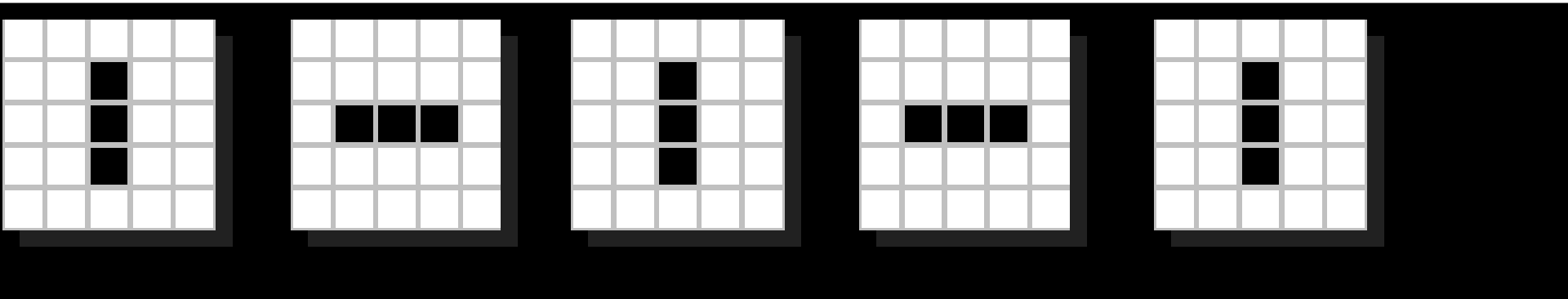
Exemplos de autômatos celulares:

- Jogo da Vida:

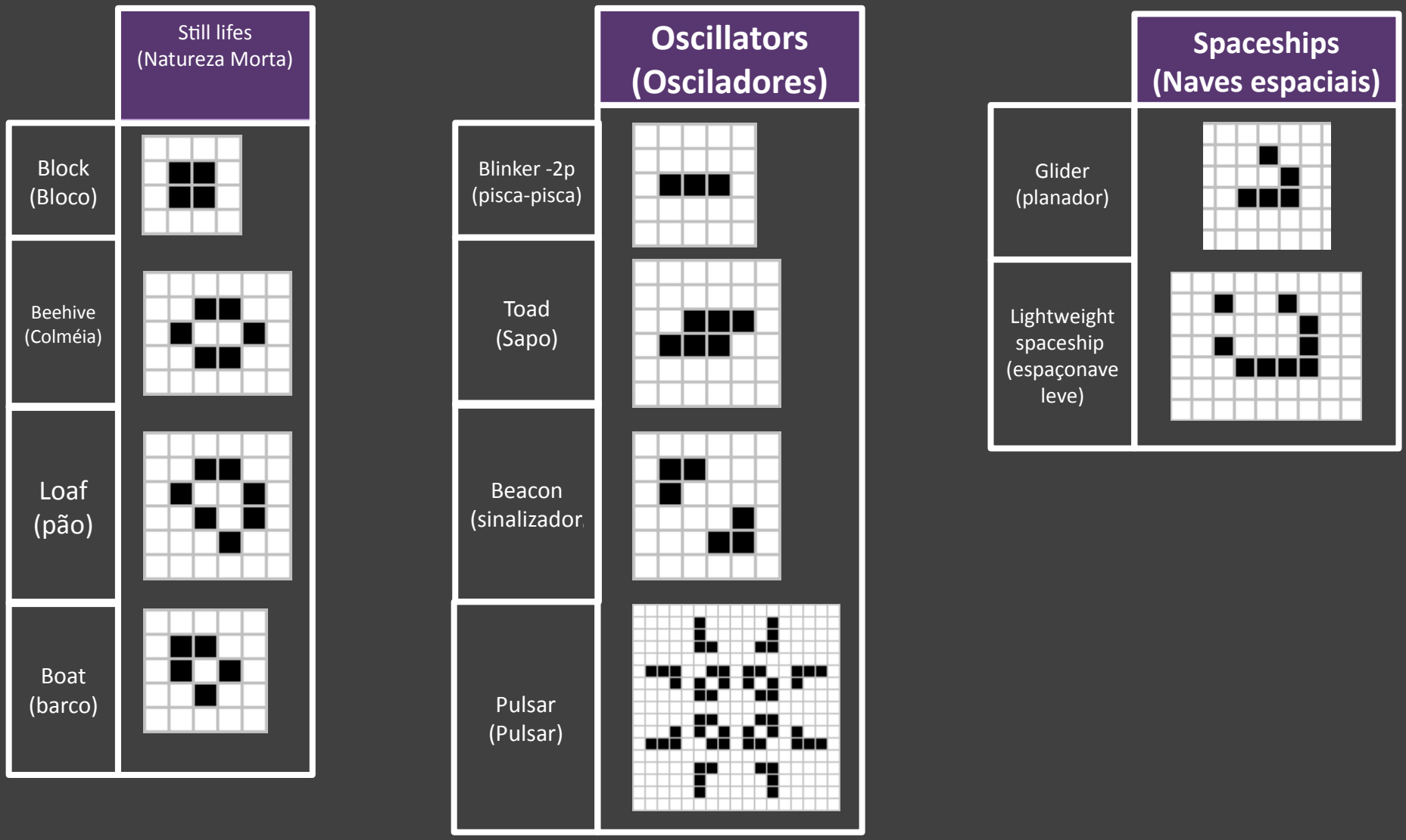


... e o “Jogo da Vida” ?

- É uma matriz em que cada célula admite um único valor: “1” ou “0” (vivo ou morto)
- Esse valor é recalculado através de 4 regras:
  * Qualquer célula viva com menos de 2 vizinhos vivos, morre;
  * Qualquer célula viva com mais de 3 vizinhos vivos, morre;
  * Qualquer célula viva com 2 ou 3 vizinhos vivos, vive na prox geração;
  * Qualquer célula morta com exatamente 3 vizinhos vivos, vive;



Alguns outros itens do Jogo da Vida:



Jogo da Vida em Python

```
# -*- coding: utf-8 -*-
"""
Created on Fri Aug 17 11:28:17 2012
@author: User
"""
################### D O C U M E N T A Ç Ã O ###################################
#Variáveis:
#   i         --> é usado para percorrer as  linhas  de uma matriz
#   j         --> é usado para percorrer as  colunas de uma matriz
#   tx        --> é o tamanho 'x' da matriz principal
#   ty        --> é o tamanho 'y' da matriz principal
#   m         --> é a matriz principal
#   n         --> é uma cópia da matriz principal
#   v         --> é uma matriz que guarda o nº de vizinhos atuais de cada célula
#   aux_cont  --> guarda quantas vezes foi chamada a função
#   cont_ger  --> conta qual o numero da geração atual
#
#Funções:
#   def_var   --> inicializa as variáveis globais
#   def_bts   --> inicializa os botões globais
#   destroy_bt--> destroi os botões para serem reUpados em "def_bts" posteriormente
#   callback  --> é a função a ser executada quando o botão bt1 é clicado
#   callback_2--> é a função a ser executada quando o botão bt2 é clicado
#   geracao           -->
#   monta             -->
###############################################################################

#JOGO DA VIDA:
from Tkinter import *
from functools import partial
import random
import time
import numpy as np

def def_var():
    global cont_ger
    global tx
    global ty
    global m
    global n
    global root
    global canvas
    global cell
    cont_ger = 1
    tx = 40
    ty = 40
    m = np.zeros((tx,ty))
    n = m.copy()
    root = Tk()
    root.title("jogo da vida")
    canvas  = Canvas(root, width=ty*10, height=tx*10, highlightthickness=0, bd=0,
bg='black')
    cell = [[0 for row in range(-1,tx+1)] for col in range(-1,ty+1)]
    for y in range(-1,tx+1):
        for x in range(-1,ty+1):
            cell[x][y] = canvas.create_rectangle((x*10, y*10, x*10+10,
y*10+10),outline="gray",fill="white")

def def_bts(m, n, aux_cont):
    global bt1
    global bt2

    bt1 = Button(root, text="PRÓXIMA GERAÇAO", command=lambda: callback(m, n,
aux_cont))
    bt2 = Button(root, text="PLAY", command=lambda: callback_2(m, n, aux_cont))
    bt1.pack()
    bt2.pack()
    canvas.pack()

def geracao(m, n, aux_cont):
    v = np.zeros((tx,ty))
    for i in range(1,tx-1):
        for j in range(1,ty-1):
            if m[i-1][j-1] == 1:
                v[i][j] = v[i][j] + 1
            if m[i-1][j] == 1:
                v[i][j] = v[i][j] + 1
            if m[i-1][j+1] == 1:
                v[i][j] = v[i][j] + 1
            if m[i][j-1] == 1:
                v[i][j] = v[i][j] + 1
            if m[i][j+1] == 1:
                v[i][j] = v[i][j] + 1
            if m[i+1][j-1] == 1:
                v[i][j] = v[i][j] + 1
            if m[i+1][j] == 1:
                v[i][j] = v[i][j] + 1
            if m[i+1][j+1] == 1:
                v[i][j] = v[i][j] + 1
    for y in range(1,tx-1):
        for x in range(1,ty-1):
            if m[x][y] == 0:
                canvas.itemconfig(cell[x][y], fill="white")
            else:
                canvas.itemconfig(cell[x][y], fill="black")
    canvas.pack()
    for i in range (tx):
        for j in range (ty):
            ########################################CONDIÇOES DO  GAME OF LIFE
            if m[i][j] == 0:
                if v[i][j] == 3:
                    n[i][j] = 1
            elif m[i][j] == 1:
                if v[i][j] > 3:
                    n[i][j] = 0
                if v[i][j] < 2:
                    n[i][j] = 0
                if v[i][j] == 2 or v[i][j] == 3:
                    n[i][j] = 1
            ########################################
    m = n.copy()
    return m

def monta(m, n, aux_cont):
    for y in range(1,tx-1):
        for x in range(1,ty-1):
            if m[x][y] == 0:
                canvas.itemconfig(cell[x][y], fill="white")
            else:
                canvas.itemconfig(cell[x][y], fill="black")
    ###############################################(Re)define os Botões
    global bt1
    global bt2
    bt1 = Button(root, text="PRÓXIMA GERAÇAO", command=lambda: callback(m, n,
aux_cont))
    bt2 = Button(root, text="PLAY", command=lambda: callback_2(m, n, aux_cont))
    bt1.pack()
    bt2.pack()
    canvas.pack()
    ###############################################
    if aux_cont == 0:
        print 'nnnl'
        root.mainloop()
        def_var()
    return

def callback(m, n, aux_cont):
    aux_cont = aux_cont + 1
    print "t = %d" %(aux_cont+1)
    destroy_bt()
    m = geracao(m, n, aux_cont)
    monta(m, n, aux_cont)

def callback_2(m, n, aux_cont):
    aux_cont = aux_cont + 1
    print "t = %d" %(aux_cont+1)
    destroy_bt()
    m = geracao(m, n, aux_cont)
    monta(m, n, aux_cont)
    root.after(100,lambda: callback_2(m, n, aux_cont))

def destroy_bt():
    bt1.destroy()
    bt2.destroy()

###############################################################
##########################   M    A   I   N   ##################
###############################################################
if __name__ == "__main__":
    aux_cont = 0
    def_var()
    ################################################
    ## I N I C I A L I Z A Ç A O   D A  M A T R I Z##
    ################################################


    m = np.zeros((tx,ty))
    m[1][6] = 1
    m[2][6] = 1
    m[1][7] = 1
    m[2][7] = 1
    m[10][6] = 1
    m[11][6] = 1
    m[9][7] = 1
    m[11][7] = 1
    m[9][8] = 1
    m[10][8] = 1
    m[17][8] = 1
    m[18][8] = 1
    m[17][9] = 1
    m[19][9] = 1
    m[17][10] = 1
    m[24][4] = 1
    m[25][4] = 1
    m[23][5] = 1
    m[25][5] = 1
    m[23][6] = 1
    m[24][6] = 1
    m[25][16] = 1
    m[26][16] = 1
    m[27][16] = 1
    m[25][17] = 1
    m[26][18] = 1
    m[35][4] = 1
    m[36][4] = 1
    m[35][5] = 1
    m[36][5] = 1
    m[36][11] = 1
    m[37][11] = 1
    m[36][12] = 1
    m[38][12] = 1
    m[36][13] = 1
    n = m.copy()
    m = geracao(m, n, aux_cont)
    monta(m, n, aux_cont)
###############################################################
###############################################################
```

Próximos Passos:

- limpetGUI

- O programa serve para simular e visualizar os resultados da simulação de equações diferenciais ordinárias que descreve o comportamento eletrofisiológico de células cardíacas;

Próximos Passos: