La morphogenèse modélisée par les structures de Turing

François Deligand

Année : 2017-2018

Problématique

Comment simuler numériquement les structures de Turing ?

1- Morphogenèse et structures de Turing

2- Modélisation informatique

3- Quelques résultats

Morphogenèse et structures de Turing

Embryon

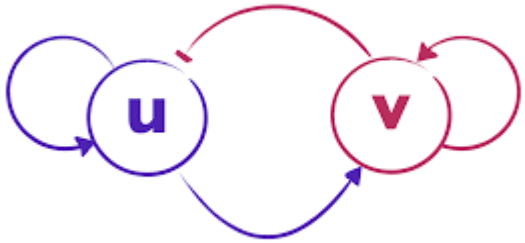
Formes et motifs

Structures

Morphogenèse et structures de Turing

Une idée de modélisation : deux espèces

Activateur et inhibiteur

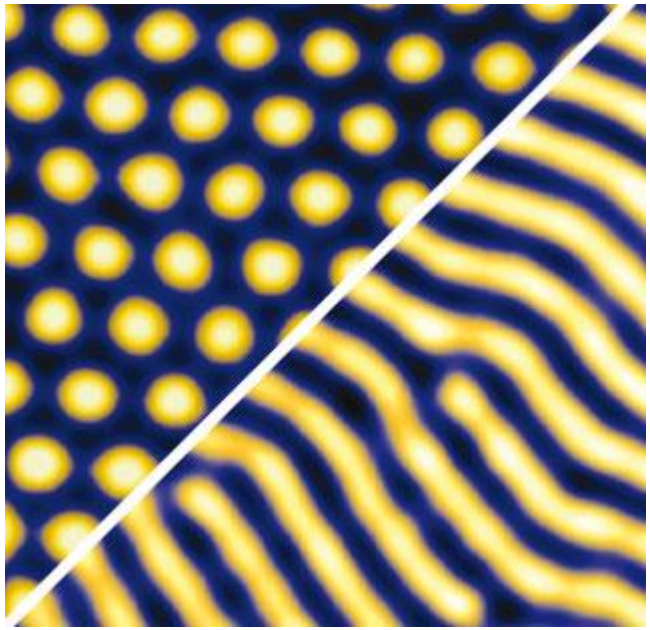


Source : *http://math.univ-lyon1.fr/~pujo/TURING-IXXI.pdf*

Système de réaction-diffusion :

$$\begin{cases} \dfrac{\partial u}{\partial t} = D_u \Delta u + f(u,v) \\ \dfrac{\partial v}{\partial t} = D_v \Delta v + g(u,v) \end{cases}$$

Modèle observé en 1990 :



Source : https://interstices.info/jcms/int_71868/alan-turing-les-motifs-et-les-structures-du-vivant

Morphogenèse et structures de Turing

Résolution approximative

Linéarisation autour d'états d'équilibre

Stabilité du système

Solution dépendante :

- Initialisation

- Coefficients de diffusion

- Espace d’application

Morphogenèse et structures de Turing

Modèle Gray-Scott

- Réaction : $U + 2V \rightarrow 3V$

Cinétique chimique : $\begin{cases} \frac{\partial u}{\partial t} = -\alpha u v^2 \\ \frac{\partial v}{\partial t} = +\alpha u v^2 \end{cases}$

- Diffusion : $\begin{cases} \frac{\partial u}{\partial t} = D_1 \Delta u \\ \frac{\partial v}{\partial t} = D_2 \Delta v \end{cases}$

- Extérieur

Modèle Gray-Scott

Après normalisation :

$$\begin{cases} \frac{\partial u}{\partial t} = D_u \Delta u - uv^2 + f(1-u) \\ \frac{\partial v}{\partial t} = D_v \Delta v + uv^2 - (f+k)v \end{cases}$$

f : coefficient de réapprovisionnement

k : coefficient d'élimination

Modèle Gray-Scott

Solutions stationnaires uniformes :

- 3 si $f > 4(k+f)^2$
- 1 seule sinon (toujours valable)

Modélisation informatique

Représentation matricielle :

- Matrices de concentrations U et V
- 1 pixel = 1 concentration

Modélisation informatique

Discrétisation du phénomène :

- Laplacien :

$$\Delta u = \frac{u(x-1,y) + u(x+1,y) + u(x,y-1) + u(x,y+1) - 4u(x,y)}{dx^2}$$

- Méthode d'Euler :

$$\begin{cases} u(t+dt) = u(t) + dt(D_u \Delta u + f\big(u(t), v(t)\big)) \\ v(t+dt) = v(t) + dt(D_v \Delta v + g\big(u(t), v(t)\big)) \end{cases}$$

Modélisation informatique

Affichage :

Matrice 3D de couleurs

```
def rend_affichable(Z, couleur0, couleur1):
    """Crée une matrice 3D affichable dans tkinter à partir de la matrice Z qui
    contient des flottants dans [0 ; 1]. Le résultat est un dégradé entre la
    couleur0 et la couleur1.
    0 -> couleur0 et 1 -> couleur1"""
    # on décompose les deux couleurs tkinter
    r0, v0, b0 = décompose(couleur0)
    r1, v1, b1 = décompose(couleur1)
    # On crée trois matrices (une par composante RVB)
    # Chaque matrice contient donc une composante plus ou moins prononcée par un
    # calcul de barycentre
    # Le "astype" permet d'obtenir des entiers entre 0 et 255
    R = (Z*r1 + (1 - Z)*r0).astype(np.uint8)
    V = (Z*v1 + (1 - Z)*v0).astype(np.uint8)
    B = (Z*b1 + (1 - Z)*b0).astype(np.uint8)
    # on recompose le tout dans une matrice 3D (instruction numpy)
    M = np.stack((R, V, B), axis=2)
    return M
```

Modélisation informatique

Création de l'interface tkinter :

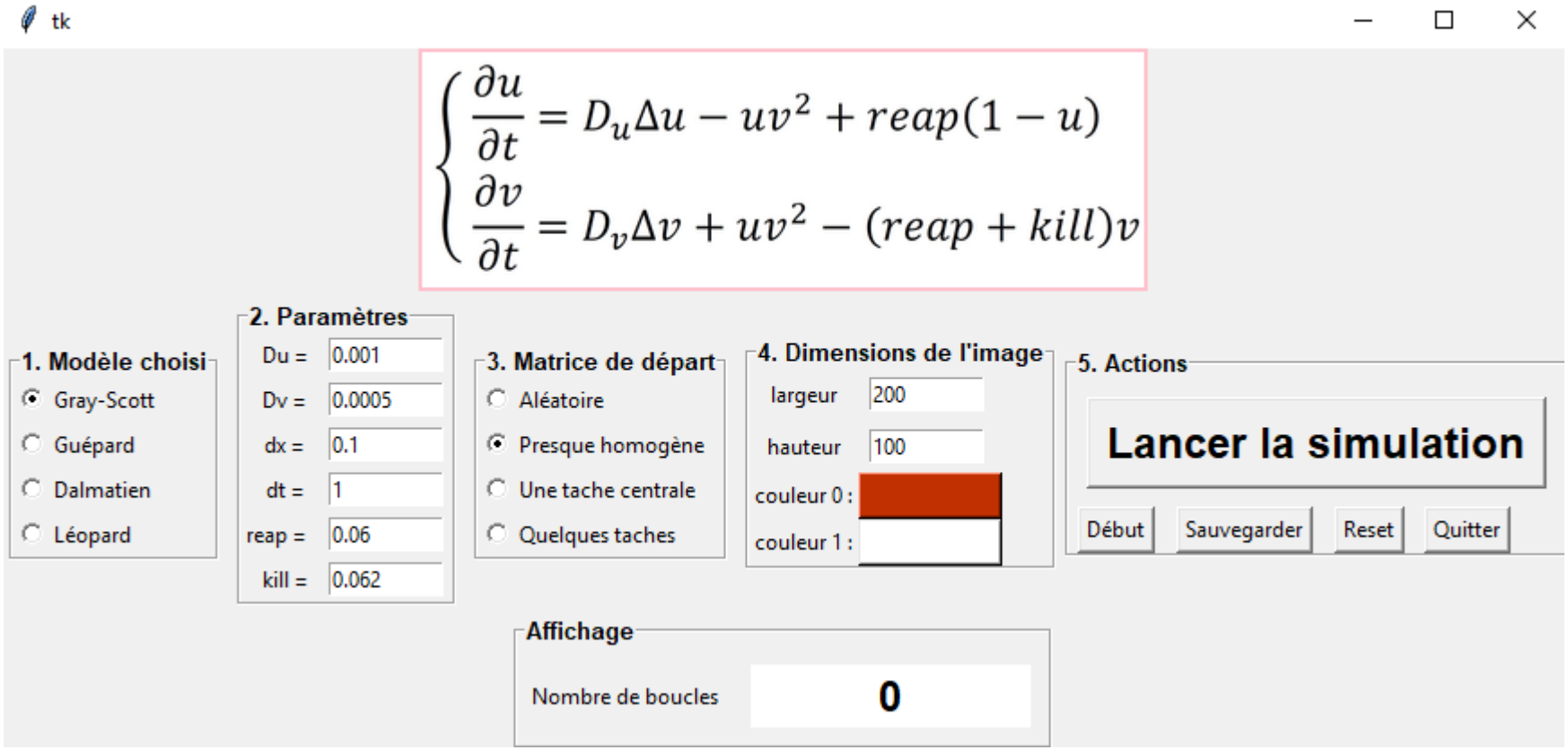
- Boucle événementielle

```
ma_fenetre.after_idle(boucle_principale)

ma_fenetre.mainloop()
```

- Voir l'évolution du système

Modélisation informatique



Modélisation informatique

Problèmes de stabilité :

- Conditions d'initialisation
- Conditions aux bords

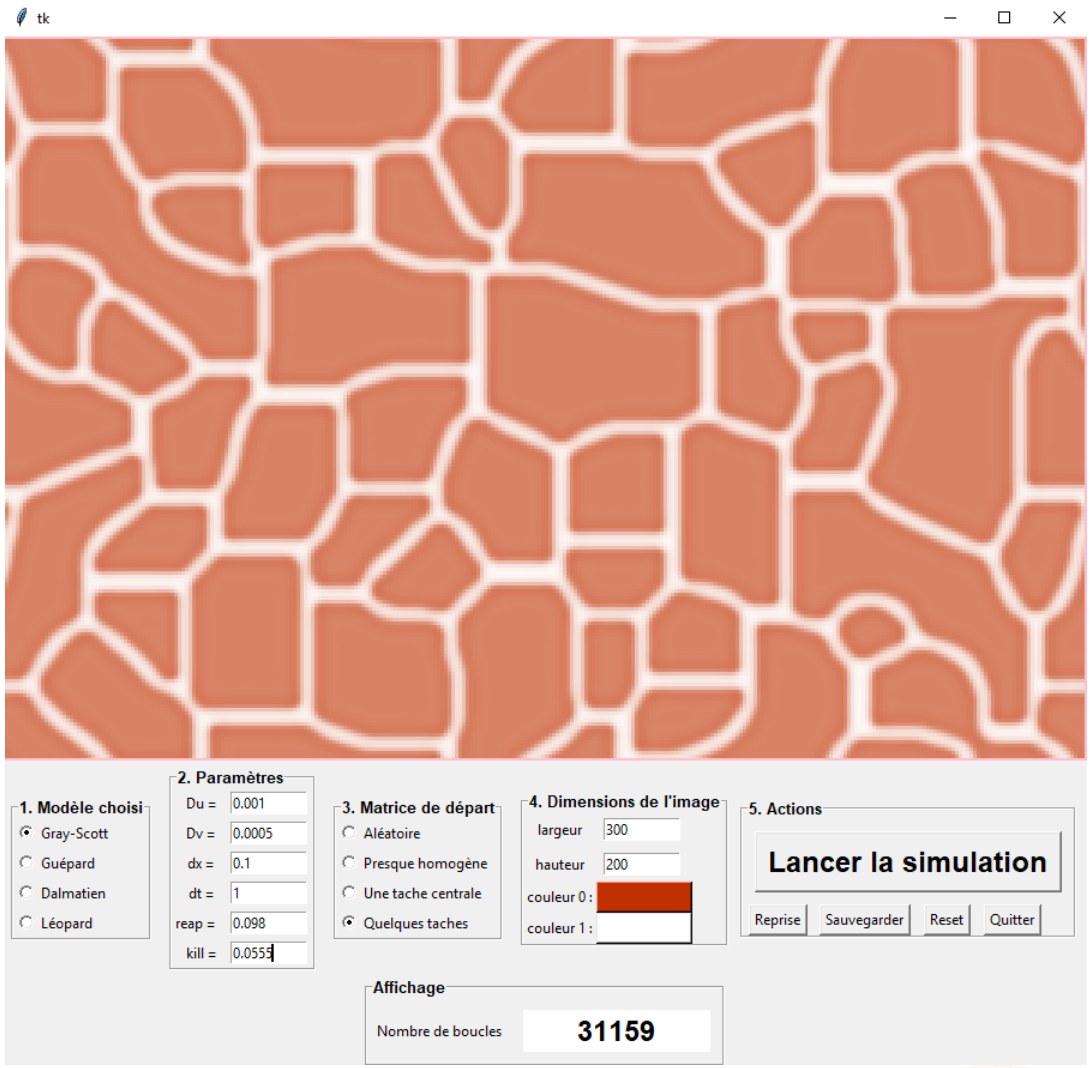
```
# on rattache les bords (haut-bas, droite-gauche)
for matrice in (U, V):
    matrice[0, :] = matrice[-2, :]
    matrice[-1, :] = matrice[1, :]
    matrice[:, 0] = matrice[:, -2]
    matrice[:, -1] = matrice[:, 1]
```

- Conditions CFL : $\max\{D_u; D_v\}\, dt \leq dx^2$
- Erreur numérique

```
U[U<0], V[V<0] = 0, 0
U[U>1], V[V>1] = 1, 1
```

Quelques résultats

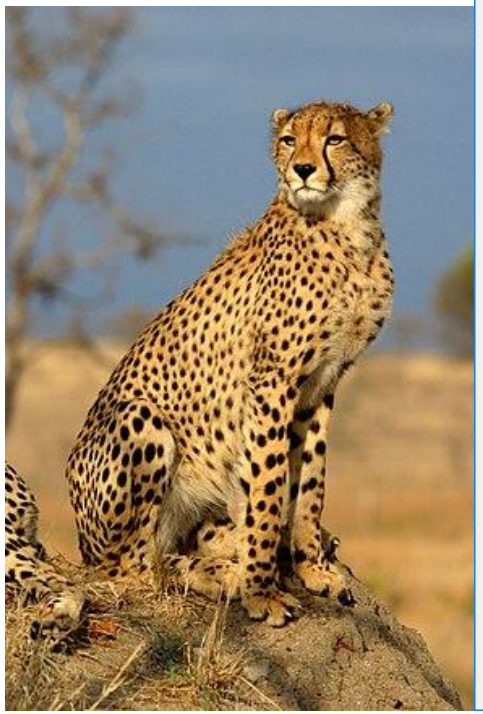
- Girafe



Source : *http://photos.linternaute.com/photo/636987/7543908954/864/girafe-reticulee/*

Quelques résultats

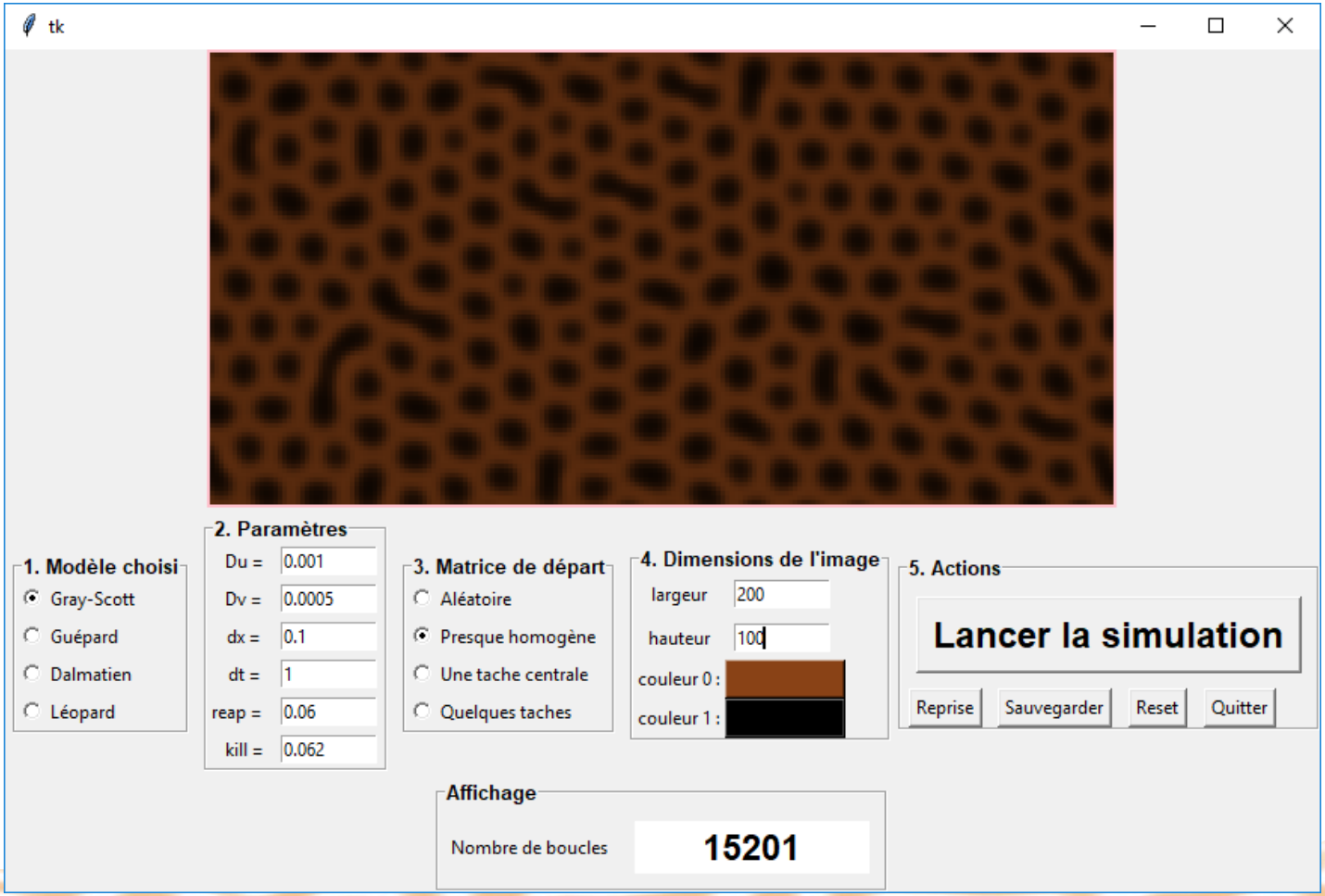
- Guépard



Source : *https://fr.wikipedia.org/wiki/Gu%C3%A9pard*

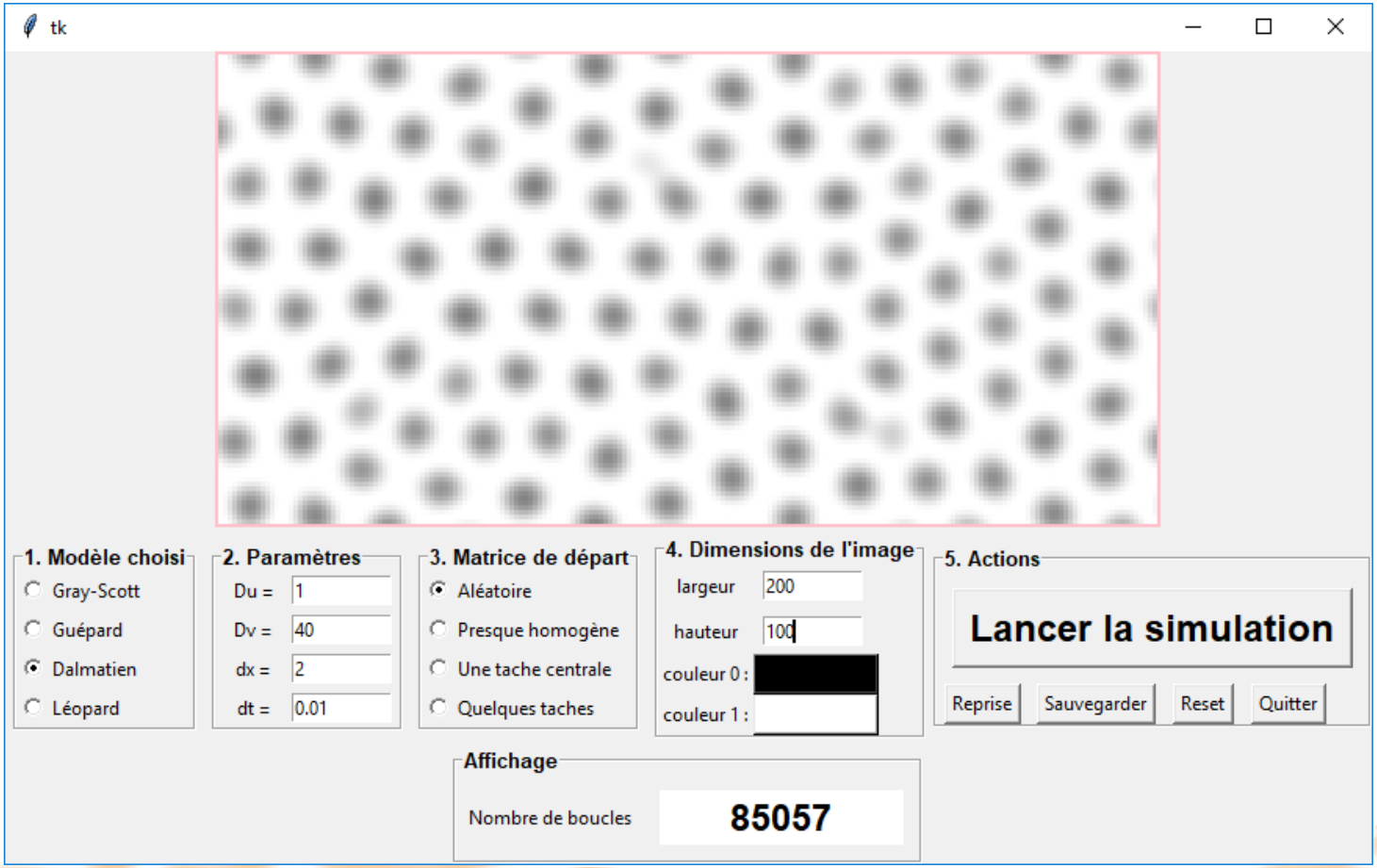
Quelques résultats

- Pois



Quelques résultats

- Pois

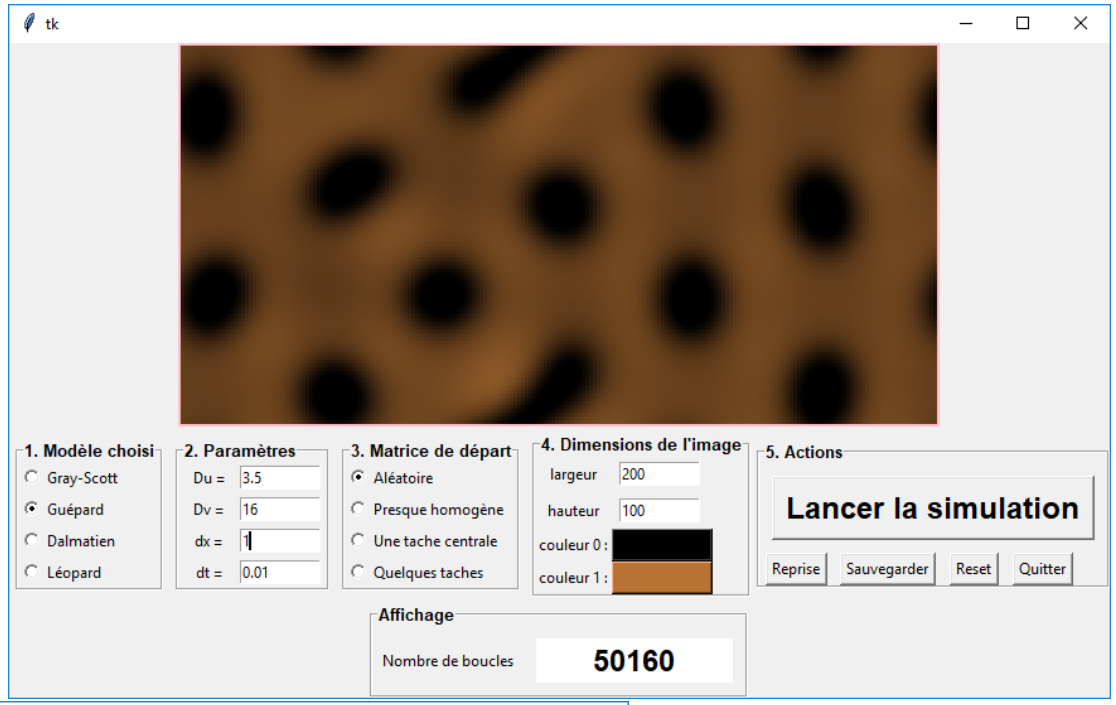


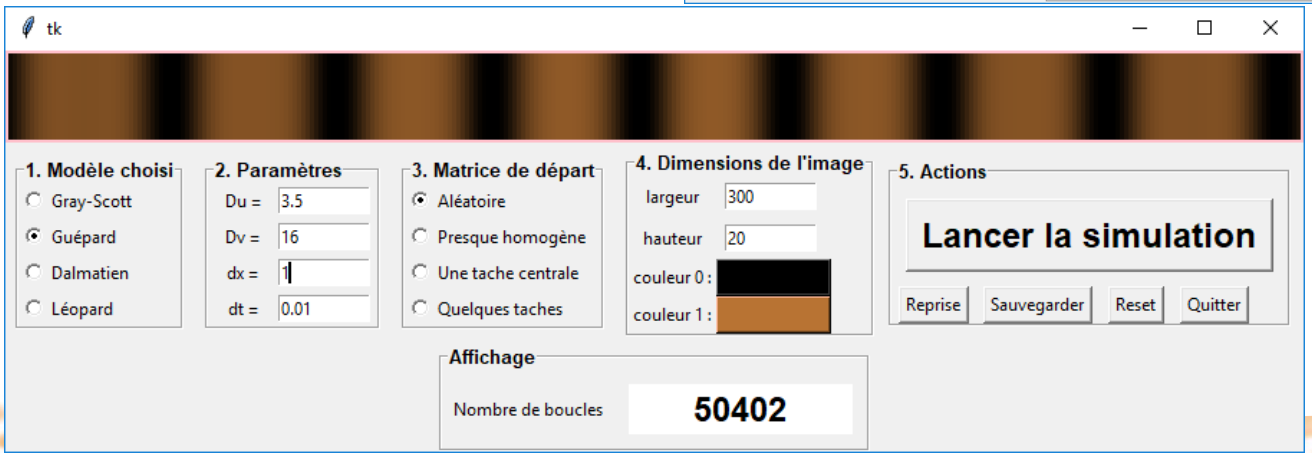
Quelques résultats

Variation d'espace

200×100 ⟶

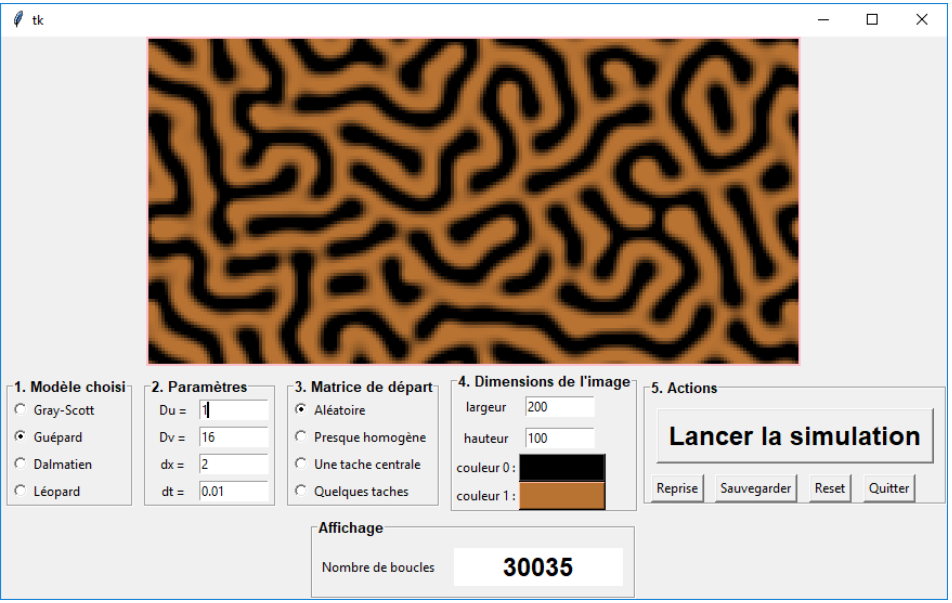
300×20 ↓



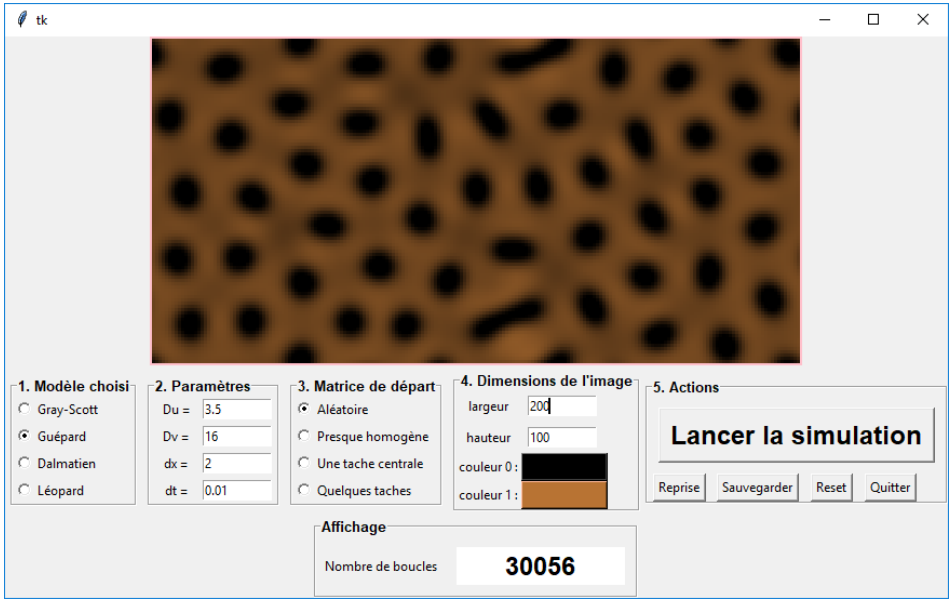


Quelques résultats

Variation des coefficients de diffusion



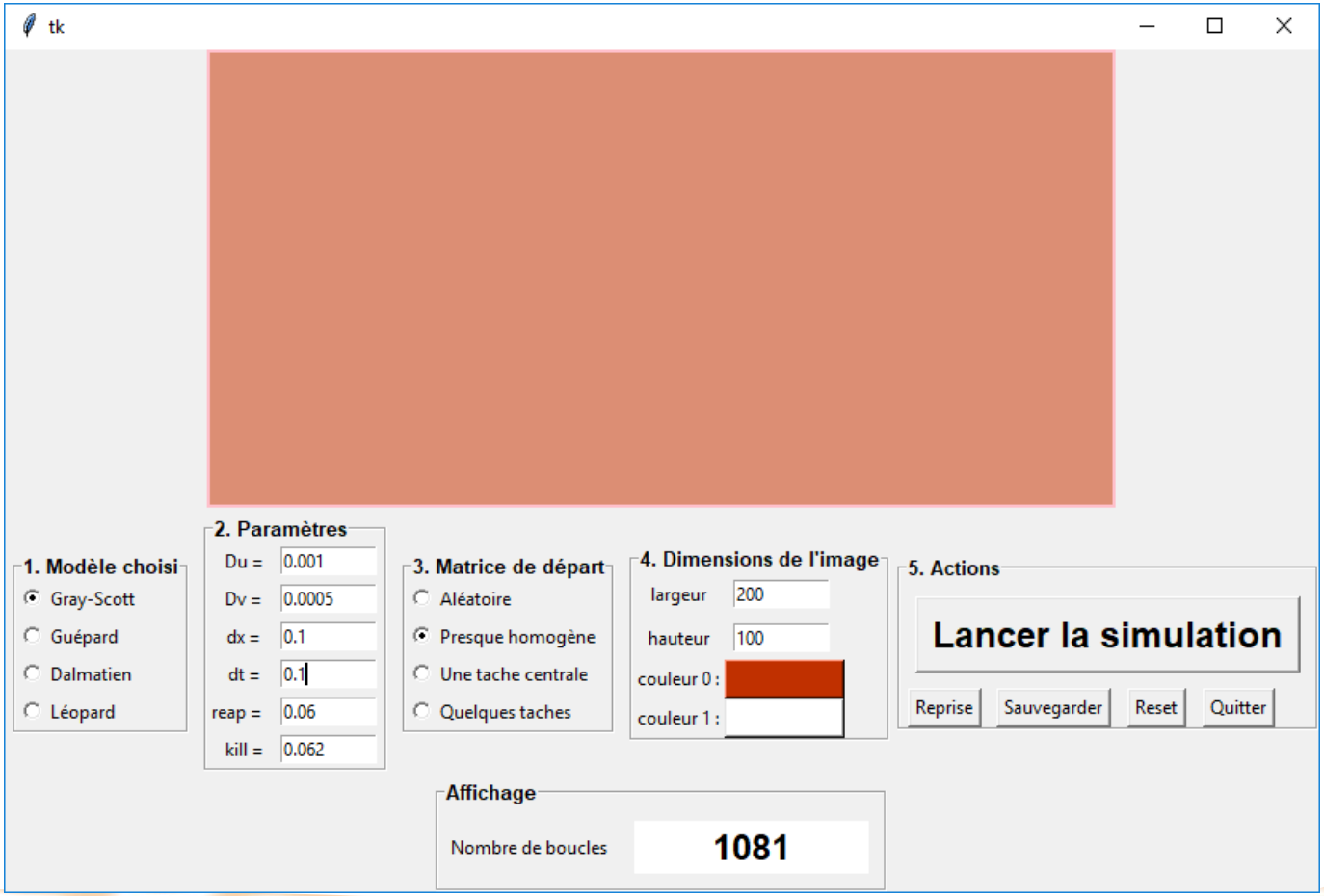
$D_u = 1$



$D_u = 3{,}5$

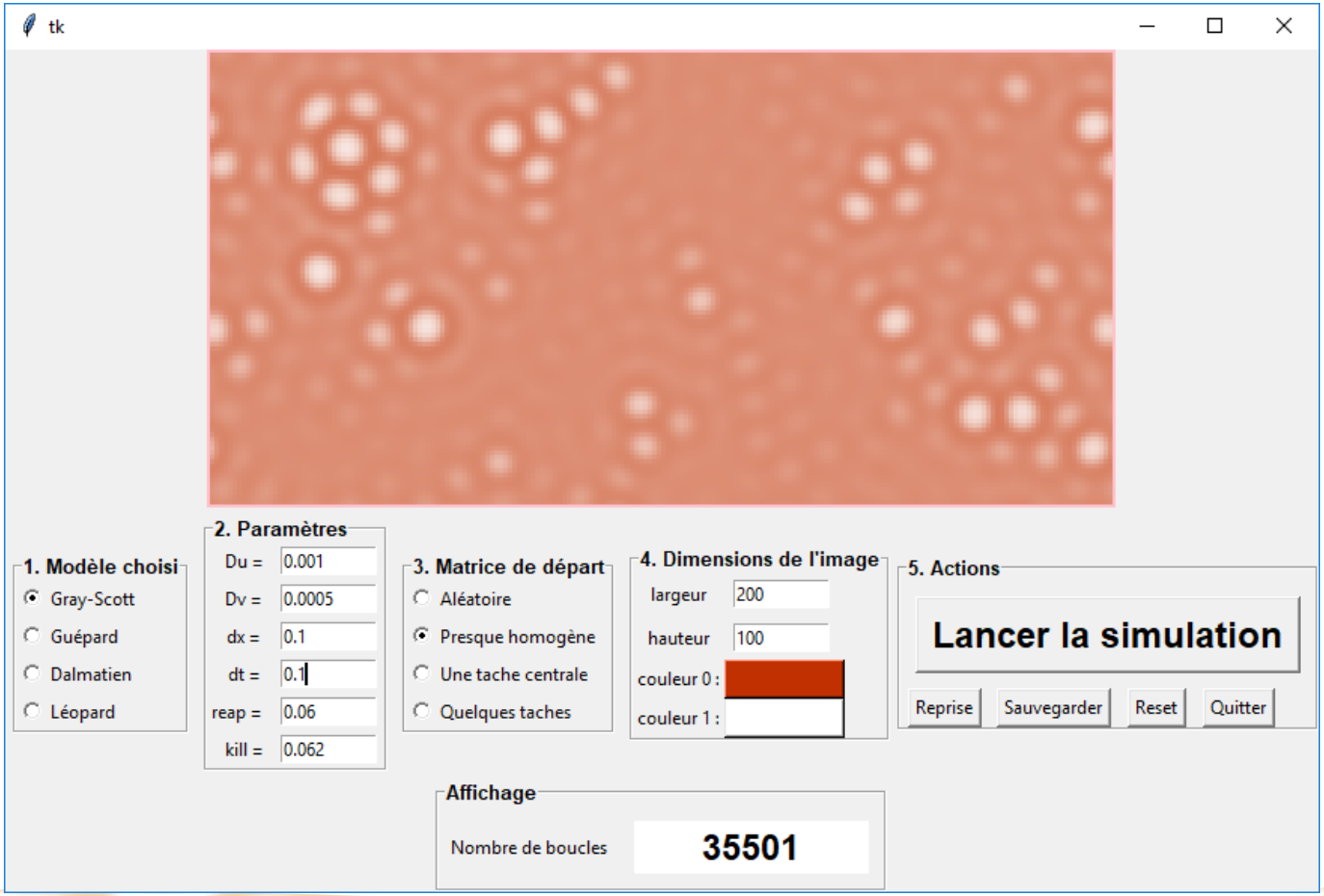
Quelques résultats

Un modèle acceptable



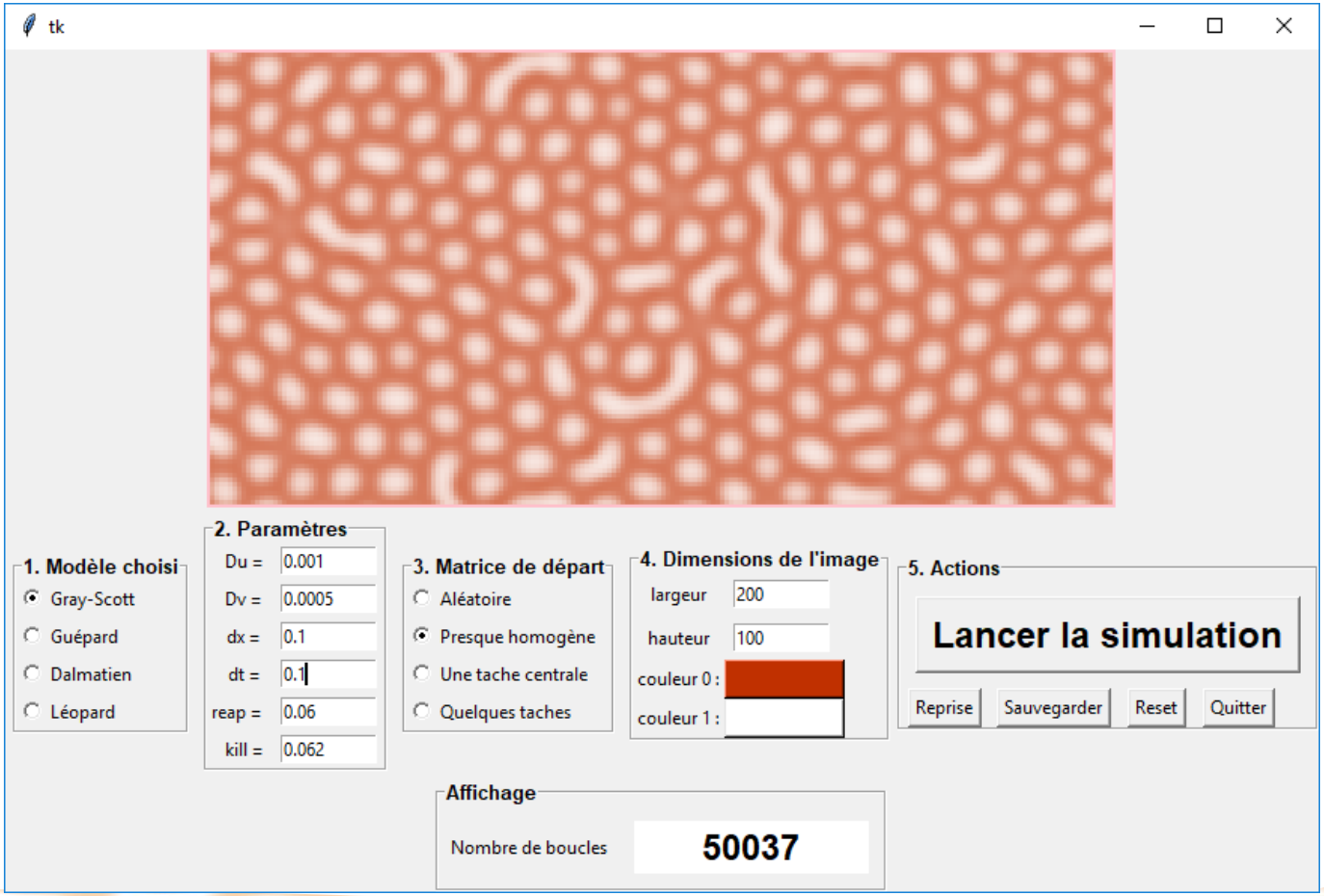
Quelques résultats

Un modèle acceptable



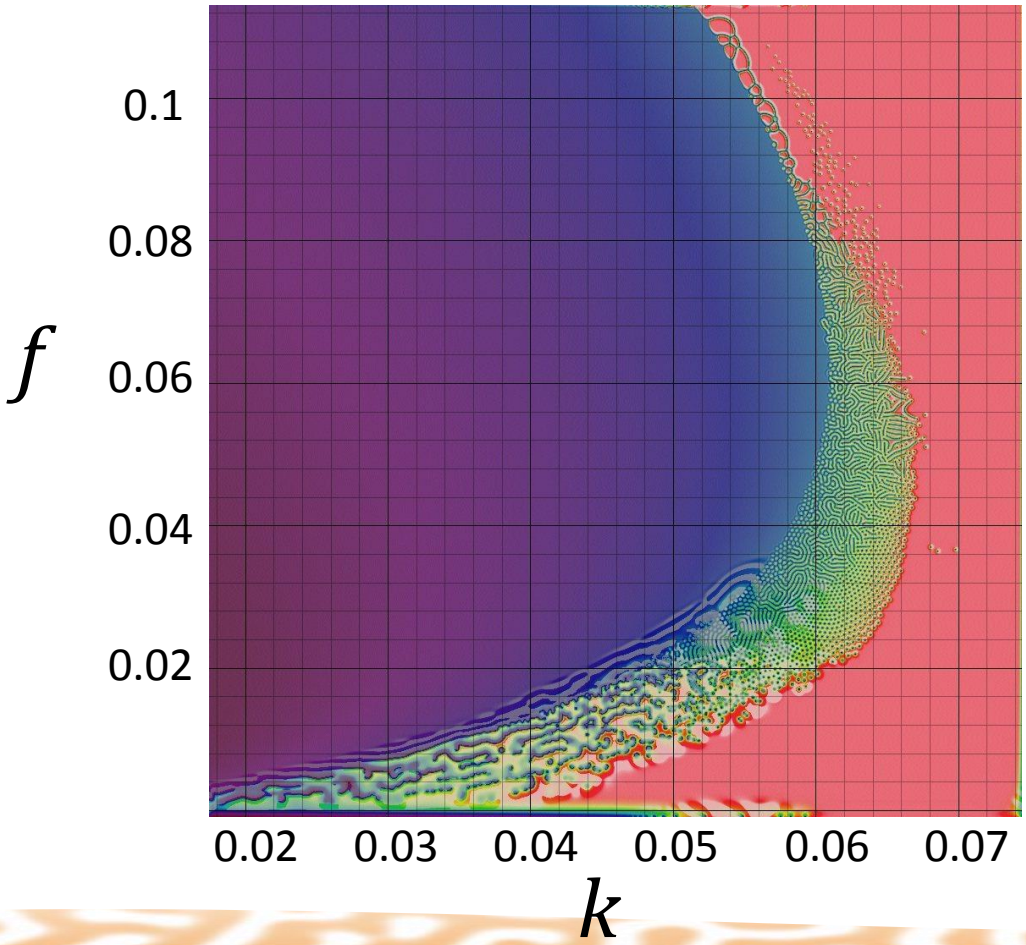
Quelques résultats

Un modèle acceptable



Quelques résultats

Sélection des paramètres du modèle Gray-Scott



Source : *http://mrob.com/pub/comp/xmorphia/*

Conclusion

Modèle parfois approximatif…

…mais des résultats satisfaisants et variés

Annexe : programme

```python
# -*- coding: utf-8 -*-

# Ce fichier modélise les équations de réaction-diffusion de Turing
# Il rend compte de l'évolution d'un système de réaction-diffusion à l'aide
# d'une interface tkinter
# Deux composants U et V interagissent, U est activateur et V inhibiteur

import numpy as np
import random
import tkinter as tk
from tkinter.colorchooser import *
from tkinter.filedialog import *
from PIL import Image, ImageTk

__author__ = 'François Deligand'


#################################################################
#\\\\      Fonctions sur l'affichage et les couleurs      /////#
#################################################################

# une fonction pour décomposer les couleurs
def décompose(couleur_tkinter='#b87333'):
    """ Prend une couleur donnée par une chaîne tkinter et la convertit en un
    triplet d'entiers entre 0 et 255"""
    # la chaîne tkinter est codée en hexadécimal et commence par #
    # la couleur par défaut est "cuivre" : '#b87333'
    rouge = int(couleur_tkinter[1:3], 16)
    vert = int(couleur_tkinter[3:5], 16)
    bleu = int(couleur_tkinter[5:], 16)
    return (rouge, vert, bleu)

def rend_affichable(Z, couleur0, couleur1):
    """Crée une matrice 3D affichable dans tkinter à partir de la matrice Z qui
    contient des flottants dans [0 ; 1]. Le résultat est un dégradé entre la
    couleur0 et la couleur1.
    0 -> couleur0 et 1 -> couleur1"""
    # on décompose les deux couleurs tkinter
    r0, v0, b0 = décompose(couleur0)
    r1, v1, b1 = décompose(couleur1)
    # On crée trois matrices (une par composante RVB)
    # Chaque matrice contient donc une composante plus ou moins prononcée par un
    # calcul de barycentre
    # Le "astype" permet d'obtenir des entiers entre 0 et 255
    R = (Z*r1 + (1 - Z)*r0).astype(np.uint8)
    V = (Z*v1 + (1 - Z)*v0).astype(np.uint8)
    B = (Z*b1 + (1 - Z)*b0).astype(np.uint8)
    # on recompose le tout dans une matrice 3D (instruction numpy)
    M = np.stack((R, V, B), axis=2)
    return M
```

```python
#################################################################
#\\\\      Initialisations en fonction du modèle      /////#
#################################################################

# les fonctions intervenant dans le système de réaction-diffusion

#########################    Modèle Gray-Scott    ###########################
####                                                                    ####
####  f(u,v) = -u*v**2 + reap(1-u)  ###  g(u,v) = u*v**2 - (kill + reap)*v  ####
####                                                                    ####
###############################################################################
def f1(X, Y):
    """ Première fonction du modèle Gray-Scott """
    # elle renvoie une matrice en fonction de deux autres
    # reap est le coefficient de réapprovisionnement
    return - X * Y**2 + reap.get() * (1 - X)

def g1(X, Y):
    """ Deuxième fonction du modèle Gray-Scott """
    # elle renvoie une matrice en fonction de deux autres
    # reap est le coefficient de réapprovisionnement
    # kill est le coefficient d'élimination
    return X * Y**2 - (kill.get() + reap.get()) * Y

def init1():
    """ Initialise les paramètres du modèle et affiche les équations """
    # création de l'image des équations du modèle
    image1 = Image.open("equa_gray_scott.png")
    equa1 = ImageTk.PhotoImage(image1)
    # affichage de l'équation
    motif.image = equa1 # pour éviter le garbage collector
    motif.config(image=equa1)
    # activation des paramètres supplémentaires
    reap_label.grid()
    reap_entree.grid()
    kill_label.grid()
    kill_entree.grid()
    # initialisation par défaut des paramètres du modèle
    Du.set(0.001)
    Dv.set(0.0005)
    dx.set(0.1)
    dt.set(1)
    reap.set(0.06)
    kill.set(0.062)
    couleur0.set('#c03000')
    bouton_couleur0.config(bg=couleur0.get())
    couleur1.set('#ffffff')
    bouton_couleur1.config(bg=couleur1.get())
    # matrice de départ
    mat_départ.set("2")
```

Annexe : programme

```
############################      Modèle guépard      ############################
####                                                                        ####
####   f(u,v) = u*(v-0.1) - 0.12   ############   g(u,v) = -u*v + 0.16      ####
####                                                                        ####
################################################################################
def f2(X, Y):
    """ Première fonction du modèle "guépard" """
    # renvoie une matrice en fonction de deux autres
    return X * (Y - 0.1) - 0.12

def g2(X, Y):
    """ Deuxième fonction du modèle "guépard" """
    # renvoie une matrice en fonction de deux autres
    return - X * Y + 0.16

def init2():
    """ Initialise les paramètres du modèle et affiche les équations """
    # création de l'image des équations du modèle
    image2 = Image.open("equa_guepard.png")
    equa2 = ImageTk.PhotoImage(image2)
    # affichage de l'équation
    motif.image = equa2 # pour éviter le garbage collector
    motif.config(image=equa2)
    # désactivation des paramètres en trop
    reap_label.grid_remove()
    reap_entree.grid_remove()
    kill_label.grid_remove()
    kill_entree.grid_remove()
    # initialisation par défaut des paramètres du modèle
    Du.set(3.5)
    Dv.set(16)
    dx.set(2)
    dt.set(0.01)
    couleur0.set('#000000')
    bouton_couleur0.config(bg=couleur0.get())
    couleur1.set('#b87333')
    bouton_couleur1.config(bg=couleur1.get())
    # matrice de départ
    mat_départ.set("1")
```

```
############################      Modèle dalmatien      ############################
####                                                                        ####
####  f(u,v) = u**2*v - u/6 + 1/360  ############  g(u,v) = -u**2*v + 3/20  ####
####                                                                        ####
################################################################################
def f3(X, Y):
    """ Première fonction du modèle "dalmatien" """
    # renvoie une matrice en fonction de deux autres
    return X**2 * Y - X/6 + 1/360

def g3(X, Y):
    """ Deuxième fonction du modèle "dalmatien" """
    # renvoie une matrice en fonction de deux autres
    return - X**2 * Y + 3/20

def init3():
    """ Initialise les paramètres du modèle et affiche les équations """
    # création de l'image des équations du modèle
    image3 = Image.open("equa_dalmatien.png")
    equa3 = ImageTk.PhotoImage(image3)
    # affichage de l'équation
    motif.image = equa3 # pour éviter le garbage collector
    motif.config(image=equa3)
    # désactivation des paramètres en trop
    reap_label.grid_remove()
    reap_entree.grid_remove()
    kill_label.grid_remove()
    kill_entree.grid_remove()
    # initialisation par défaut des paramètres du modèle
    Du.set(1)
    Dv.set(40)
    dx.set(2)
    dt.set(0.01)
    couleur0.set('#000000')
    bouton_couleur0.config(bg=couleur0.get())
    couleur1.set('#ffffff')
    bouton_couleur1.config(bg=couleur1.get())
    # matrice de départ
    mat_départ.set("1")
```

Annexe : programme

```
183 ##############################     Modèle léopard     ##########################
184 ####                                                                        ####
185 ####  f(u,v) = -u**3 + u - v - 0.005   ############   g(u,v) = 10(u-v)      ####
186 ####                                                                        ####
187 ###############################################################################
188 def f4(X, Y):
189     """ Première fonction du modèle "léopard" """
190     # renvoie une matrice en fonction de deux autres
191     return - X**3 + X - Y - 0.005
192 
193 def g4(X, Y):
194     """ Deuxième fonction du modèle "léopard" """
195     # renvoie une matrice en fonction de deux autres
196     return 10 * (X - Y)
197 
198 def init4():
199     """ Initialise les paramètres du modèle et affiche les équations """
200     # création de l'image des équations du modèle
201     image4 = Image.open("equa_leopard.png")
202     equa4 = ImageTk.PhotoImage(image4)
203     # affichage de l'équation
204     motif.image = equa4 # pour éviter le garbage collector
205     motif.config(image=equa4)
206     # désactivation des paramètres en trop
207     reap_label.grid_remove()
208     reap_entree.grid_remove()
209     kill_label.grid_remove()
210     kill_entree.grid_remove()
211     # initialisation par défaut des paramètres du modèle
212     Du.set(0.001)
213     Dv.set(0.05)
214     dx.set(0.02)
215     dt.set(0.0018)
216     couleur0.set('#000000')
217     bouton_couleur0.config(bg=couleur0.get())
218     couleur1.set('#b87333')
219     bouton_couleur1.config(bg=couleur1.get())
220     # matrice de départ
221     mat_départ.set("1")
```

```
223 # une fonction qui initialise les paramètres selon le modèle choisi
224 # l'initialisation est une étape très importante !
225 def init_param():
226     """ Initialisation des paramètres avant de lancer l'algorithme """
227     # les paramètres dépendent du modèle choisi
228     global f, g # pour qu'elles existent dans la boucle principale
229     if text_pause.get() == "Début":
230         # on réactive le bouton de lancement de la simulation
231         bouton_lancer.configure(state='normal')
232         if modèle.get() == "1":
233             init1()
234             # affectation des deux fonctions
235             f = f1
236             g = g1
237         elif modèle.get() == "2":
238             init2()
239             # affectation des deux fonctions
240             f = f2
241             g = g2
242         elif modèle.get() == "3":
243             init3()
244             # affectation des deux fonctions
245             f = f3
246             g = g3
247         elif modèle.get() == "4":
248             init4()
249             # affectation des deux fonctions
250             f = f4
251             g = g4
```

Annexe : programme

```
253 # une fonction qui initialise les matrices selon le modèle choisi
254 def init_matrice():
255     """ Création des matrices U et V de départ. U représente la concentration de
256     l'activateur en chaque point de l'espace, V celle de l'inhibiteur """
257     global U, V # pour qu'elles existent dans la boucle principale
258     # l'initialisation dépend aussi du modèle choisi
259     if mat_départ.get() == "1": # concentrations partout aléatoires
260         U = np.random.rand(hauteur.get(), largeur.get())
261         V = np.random.rand(hauteur.get(), largeur.get())
262     elif mat_départ.get() == "2": # concentrations presque homogènes
263         U = np.random.rand(hauteur.get(), largeur.get())
264         V = np.random.rand(hauteur.get(), largeur.get())
265         U = U * 0.4559 + (1 - U) * 0.4562
266         V = V * 0.2674 + (1 - V) * 0.2676
267     elif mat_départ.get() == "3": # une tache centrale
268         U = np.ones((hauteur.get(), largeur.get())) # nb_lignes, nb_colonnes
269         V = np.zeros((hauteur.get(), largeur.get())) # idem
270         # création d'un carré de forte concentration de V au centre
271         l_centre = hauteur.get()//2 # ligne du centre
272         c_centre = largeur.get()//2 # colonne du centre
273         for i in range(10):
274             for j in range(10):
275                 U[l_centre + i , c_centre + j] = 0
276                 V[l_centre + i , c_centre + j] = 1
277     elif mat_départ.get() == "4": # quelques taches au hasard
278         U = np.ones((hauteur.get(), largeur.get())) # nb_lignes, nb_colonnes
279         V = np.zeros((hauteur.get(), largeur.get())) # idem
280         for zone in range(100):
281             # on place 100 carrés de taille 10*10 aléatoirement
282             l_coin = random.randint(0, hauteur.get() - 10) # ligne du coin
283             c_coin = random.randint(0, largeur.get() - 10) # colonne du coin
284             for i in range(10):
285                 for j in range(10):
286                     U[l_coin + i , c_coin + j] = 0
287                     V[l_coin + i , c_coin + j] = 1
288
289 ################################################################
290 #\\\\          Fonctions de calcul du système          /////#
291 ################################################################
292
293 # on définit l'opérateur laplacien pour les matrices (discret)
294 def laplacien(Z):
295     """ Calcule le laplacien de la matrice en entrée """
296     # on effectue du calcul matriciel pour réduire la complexité temporelle
297     Zh = Z[0:-2,1:-1]      # les coefficients du haut
298     Zg = Z[1:-1,0:-2]      # les coefficients à gauche
299     Zb = Z[2:,1:-1]        # les coefficients du bas
300     Zd = Z[1:-1,2:]        # les coefficients à droite
301     Zc = Z[1:-1,1:-1]      # les coefficients au centre
302     # on applique la méthode d'Euler pour la dérivée seconde
303     # même élément différentiel pour les deux dimensions d'espace
304     return (Zh + Zg + Zb + Zd - 4 * Zc) / dx.get()**2
```

```
306 # la matrice qui permet de grossir les pixels (agrandir la matrice U)
307 forme = np.ones((3, 3))   # sera utilisée dans un produit de Kronecker avec U
308
309 def boucle_principale():
310     """ Fonction auto-appelante qui met à jour les matrices U et V du système et
311     affiche l'image correspondante """
312     if text_pause.get() == "Début":
313         init_matrice()
314         text_pause.set("Pause")
315         ma_fenetre.after_idle(boucle_principale)
316     elif text_pause.get() == "Pause":
317         # on affiche l'état de la matrice U toutes les 100 boucles
318         if nb_boucles.get() % 100 == 0:
319             #print(U.min(), U.max()) # pour tester la stabilité
320             M = rend_affichable(np.kron(U, forme),
321                                 couleur0.get(), couleur1.get())
322             im = Image.fromarray(M)
323             motif.sauvegarde = im # pour pouvoir sauvegarder l'image
324             photo = ImageTk.PhotoImage(im)
325             motif.image = photo # indispensable pour éviter le garbage collector
326             motif.config(image=photo)
327         # on calcule les laplaciens des deux matrices
328         deltaU = laplacien(U)
329         deltaV = laplacien(V)
330         # on travaille sur l'intérieur (centre) de la matrice
331         # ie on retire les bords
332         Uint = U[1:-1,1:-1]
333         Vint = V[1:-1,1:-1]
334         # mise à jour en suivant les équations du système
335         # on applique la méthode d'Euler en faisant du calcul matriciel pour
336         # réduire la complexité temporelle
337         U[1:-1,1:-1], V[1:-1,1:-1] = \
338             Uint + dt.get() * (Du.get() * deltaU + f(Uint, Vint)), \
339             Vint + dt.get() * (Dv.get() * deltaV + g(Uint, Vint))
340         U[U<0], V[V<0] = 0, 0 # pour corriger une éventuelle erreur numérique
341         U[U>1], V[V>1] = 1, 1 # pour corriger une éventuelle erreur numérique
342 ##         # on simule les conditions de Neumann aux bords des matrices
343 ##         for matrice in (U, V):
344 ##             matrice[0, :] = matrice[1, :]
345 ##             matrice[-1, :] = matrice[-2, :]
346 ##             matrice[:, 0] = matrice[:, 1]
347 ##             matrice[:, -1] = matrice[:, -2]
348         # on rattache les bords (haut-bas, droite-gauche)
349         for matrice in (U, V):
350             matrice[0, :] = matrice[-2, :]
351             matrice[-1, :] = matrice[1, :]
352             matrice[:, 0] = matrice[:, -2]
353             matrice[:, -1] = matrice[:, 1]
354         # on incrémente le nombre de boucles une fois la boucle terminée
355         nb_boucles.set(nb_boucles.get() + 1)
356         ma_fenetre.after_idle(boucle_principale)
```

Annexe : programme

```
358 ##############################################################
359 #\\\\   Fonctions-commandes des boutons de l'interface   /////#
360 ##############################################################
361
362 def lancer():
363     """ Ce qui se passe quand on appuie sur le bouton "LANCER" """
364     boucle_principale()
365
366 def pause():
367     """ Ce qui se passe quand on appuie sur le bouton "PAUSE" """
368     if text_pause.get() == "Pause":
369         text_pause.set("Reprise")
370     elif text_pause.get() == "Reprise":
371         text_pause.set("Pause")
372         ma_fenetre.after_idle(boucle_principale)
373
374 def sauvegarder():
375     """ Permet de sauvegarder l'image quand on appuie sur le bouton
376     "SAUVEGARDER" """
377     # on ne peut sauvegarder que si le système est en pause
378     if text_pause.get() == "Reprise":
379         nom = asksaveasfilename()
380         motif.sauvegarde.save(nom)
381
382 def reset():
383     """ Permet de réinitialiser tout le système quand on appuie sur le bouton
384     "RESET" """
385     # on ne peut réinitialiser que si le système est en pause
386     if text_pause.get() == "Reprise":
387         text_pause.set("Début")
388         nb_boucles.set(0)
389         init_param()
390         init_matrice()
391
392 def quitter():
393     """ Ce qui se passe quand on appuie sur le bouton "QUITTER" """
394     ma_fenetre.destroy()
```

```
396 # on définit les fonctions pour le choix des couleurs dans l'image
397 def choix_couleur0():
398     """ Permet le choix de la couleur quand U = 0 en utilisant la palette de
399     couleurs du système. On réactualise le bouton par la même occasion """
400     choix = askcolor()
401     couleur0.set(choix[1]) # car choix est un 2-uplet tkinter
402     bouton_couleur0.config(bg=couleur0.get())
403
404 def choix_couleur1():
405     """ Permet le choix de la couleur quand U = 1 en utilisant la palette de
406     couleurs du système. On réactualise le bouton par la même occasion """
407     choix = askcolor()
408     couleur1.set(choix[1]) # car choix est un 2-uplet tkinter
409     bouton_couleur1.config(bg=couleur1.get())
410
411
412 ##############################################################
413 #\\\\   Création de l'interface graphique Tkinter   /////#
414 ##############################################################
415
416 # création d'une fenêtre tkinter
417 ma_fenetre = tk.Tk()
418
419
420 # création d'une étiquette aux bonnes dimensions (contient l'image)
421 motif = tk.Label(ma_fenetre, bg='pink')
422 motif.grid(row=1, column=1, columnspan=6, sticky='N')
423
424 # création du cadre "MODELE"
425 cadre_modele = tk.LabelFrame(ma_fenetre, text='1. Modèle choisi',
426                              font=('Arial','10','bold'))
427 # Rappel : bold = en gras
428 cadre_modele.grid(row=2, column=1, padx=5, pady=5)
429 valeurs = ["1", "2", "3", "4"]
430 modeles = ["Gray-Scott", "Guépard", "Dalmatien", "Léopard"]
431 modèle = tk.StringVar()
432 for i in range(len(modeles)):
433     b = tk.Radiobutton(cadre_modele, variable=modèle, text=modeles[i],
434                        value=valeurs[i], command=init_param)
435     b.grid(row=i, column=1, sticky='W')
```

Annexe : programme

```
# création du cadre "PARAMETRES"
cadre_param = tk.LabelFrame(ma_fenetre, text='2. Paramètres',
                            font=('Arial','10','bold'))
cadre_param.grid(row=2, column=2, padx=5, pady=5)
# création du paramètre de diffusion pour u
Du = tk.DoubleVar()
Du.set(0.001)  # valeur par défaut
Du_label = tk.Label(cadre_param, text='   Du = ')
Du_label.grid(row=1, column=1)
Du_entree = tk.Entry(cadre_param, width=10, textvariable=Du)
Du_entree.grid(row=1, column=2, padx=5, pady=3)
# création du paramètre de diffusion pour v
Dv = tk.DoubleVar()
Dv.set(0.0005)  # valeur par défaut
Dv_label = tk.Label(cadre_param, text='   Dv = ')
Dv_label.grid(row=2, column=1)
Dv_entree = tk.Entry(cadre_param, width=10, textvariable=Dv)
Dv_entree.grid(row=2, column=2, padx=5, pady=3)
# création de l'élément différentiel d'espace
dx = tk.DoubleVar()
dx.set(0.1)  # valeur par défaut
dx_label = tk.Label(cadre_param, text='   dx = ')
dx_label.grid(row=3, column=1)
dx_entree = tk.Entry(cadre_param, width=10, textvariable=dx)
dx_entree.grid(row=3, column=2, padx=5, pady=3)
# création de l'élément différentiel de temps
dt = tk.DoubleVar()
dt.set(1)  # valeur par défaut
dt_label = tk.Label(cadre_param, text='   dt = ')
dt_label.grid(row=4, column=1)
dt_entree = tk.Entry(cadre_param, width=10, textvariable=dt)
dt_entree.grid(row=4, column=2, padx=5, pady=3)
# création du coefficient de réapprovisionnement
reap = tk.DoubleVar()
reap.set(0.03)  # valeur par défaut
reap_label = tk.Label(cadre_param, text='reap = ')
reap_label.grid(row=5, column=1)
reap_entree = tk.Entry(cadre_param, width=10, textvariable=reap)
reap_entree.grid(row=5, column=2, padx=5, pady=3)
# création du coefficient d'élimination
kill = tk.DoubleVar()
kill.set(0.063)  # valeur par défaut
kill_label = tk.Label(cadre_param, text='   kill = ')
kill_label.grid(row=6, column=1)
kill_entree = tk.Entry(cadre_param, width=10, textvariable=kill)
kill_entree.grid(row=6, column=2, padx=5, pady=3)
```

```
# création du cadre "MATRICE DE DEPART"
cadre_init = tk.LabelFrame(ma_fenetre, text='3. Matrice de départ',
                           font=('Arial','10','bold'))
cadre_init.grid(row=2, column=3, padx=5, pady=5)
possibilités = ["1", "2", "3", "4"]
choix_matrice = ["Aléatoire", "Presque homogène", "Une tache centrale",
                 "Quelques taches"]
mat_départ = tk.StringVar()
for i in range(len(choix_matrice)):
    b = tk.Radiobutton(cadre_init, variable=mat_départ, text=choix_matrice[i],
                       value=possibilités[i], command=init_matrice)
    b.grid(row=i, column=1, sticky='W')

# création du cadre "IMAGE"
cadre_im = tk.LabelFrame(ma_fenetre, text='4. Dimensions de l\'image',
                         font=('Arial','10','bold'))
cadre_im.grid(row=2, column=4, padx=5, pady=5)
# création de la largeur (nb de colonnes des matrices U et V)
largeur = tk.IntVar()
largeur.set(200)  # valeur par défaut
largeur_label = tk.Label(cadre_im, text='largeur')
largeur_label.grid(row=1, column=1)
largeur_entree = tk.Entry(cadre_im, width=10, textvariable=largeur)
largeur_entree.grid(row=1, column=2, padx=5, pady=5)
# création de la hauteur (nb de lignes des matrices U et V)
hauteur = tk.IntVar()
hauteur.set(100)  # valeur par défaut
hauteur_label = tk.Label(cadre_im, text='hauteur')
hauteur_label.grid(row=2, column=1)
hauteur_entree = tk.Entry(cadre_im, width=10, textvariable=hauteur)
hauteur_entree.grid(row=2, column=2, padx=5, pady=5)
# création de la couleur 0 utilisée pour U = 0
couleur0 = StringVar()
couleur0.set('#000000')
couleur0.label = tk.Label(cadre_im, text='couleur 0 :')
couleur0.label.grid(row=3, column=1)
bouton_couleur0 = tk.Button(cadre_im, bg=couleur0.get(), width=10,
                            command=choix_couleur0)
bouton_couleur0.grid(row=3, column=2, columnspan=2)
# création de la couleur 1 utilisée pour U = 1
couleur1 = StringVar()
couleur1.set('#b87333')
couleur1.label = tk.Label(cadre_im, text='couleur 1 :')
couleur1.label.grid(row=4, column=1)
bouton_couleur1 = tk.Button(cadre_im, bg=couleur1.get(), width=10,
                            command=choix_couleur1)
bouton_couleur1.grid(row=4, column=2, columnspan=2)
```

Annexe : programme

```
532 # création du cadre "ACTION"
533 cadre_actions = tk.LabelFrame(ma_fenetre, text="5. Actions",
534                                font=('Arial','10','bold'))
535 cadre_actions.grid(row=2, column=5)
536 # création du bouton "LANCER"
537 bouton_lancer = tk.Button(cadre_actions, text="Lancer la simulation",
538                           font=('Arial', '18', 'bold'), command=lancer,
539                           state='disabled')
540 # le bouton est désactivé tant qu'on a pas choisi un modèle (bouton radio) et
541 # donc lancé la commande init_param()
542 bouton_lancer.grid(row=1, column=1, columnspan=6, padx=10, pady=10)
543 # création du bouton "PAUSE"
544 text_pause = tk.StringVar()
545 text_pause.set("Début")
546 bouton_pause = tk.Button(cadre_actions, textvariable=text_pause, command=pause)
547 bouton_pause.grid(row=2, column=1)
548 # création du bouton "SAUVEGARDER"
549 bouton_sauvegarder = tk.Button(cadre_actions, text="Sauvegarder",
550                                command=sauvegarder)
551 bouton_sauvegarder.grid(row=2, column=2)
552 # création du bouton "RESET"
553 bouton_reset = tk.Button(cadre_actions, text="Reset", command=reset)
554 bouton_reset.grid(row=2, column=3)
555 # création du bouton "QUITTER"
556 bouton_quitter = tk.Button(cadre_actions, text="Quitter", command=quitter)
557 bouton_quitter.grid(row=2, column=4)
558
559 # création du cadre "AFFICHAGE"
560 cadre_affichage = tk.LabelFrame(ma_fenetre, text="Affichage",
561                                 font=('Arial','10','bold'))
562 cadre_affichage.grid(row=3, column=1, columnspan=6)
563 # création du nombre de boucles effectuées par la fonction auto-appelante
564 nb_boucles_label = tk.Label(cadre_affichage, text="Nombre de boucles")
565 nb_boucles = tk.IntVar()
566 nb_boucles_affich = tk.Label(cadre_affichage, textvariable=nb_boucles, width=10,
567                              bg='white', font=('Arial', '18', 'bold'))
568 nb_boucles_label.grid(row=1, column=1, padx=5, pady=5)
569 nb_boucles_affich.grid(row=1, column=2, padx=10, pady=10)
570
571
572
573 ##############################################################
574 #\\\\       Lancement de la fenêtre graphique        /////#
575 ##############################################################
576
577 ma_fenetre.mainloop()
```