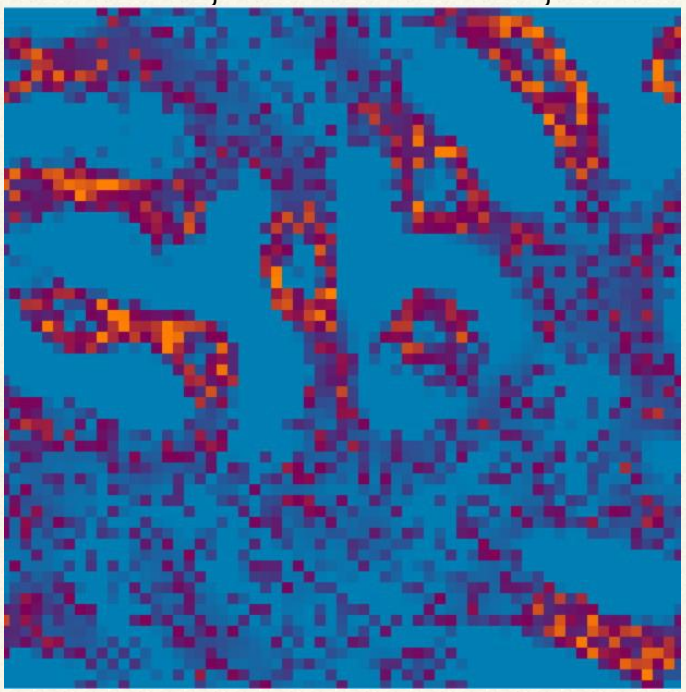
Lenia,
un automate cellulaire continu

TIPE 2024-2025
Transition, transformation, conversion



- Introduction
- I) Du Jeu de la vie à Lenia
- II) Implémentation
- III)Observations
- Conclusion
- Annexe

Introduction

Le Jeu de la vie

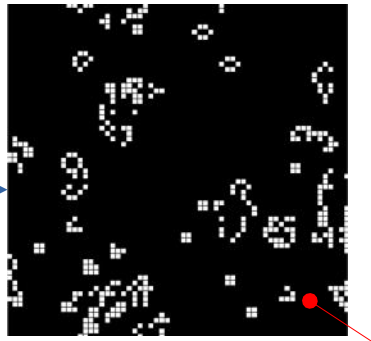
Une cellule vivante (vaut 1)

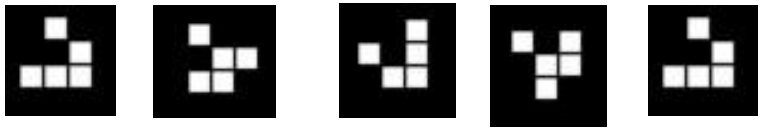
règles d'évolution

Une cellule morte (vaut 0)

grille composée de cellules

État de la grille au bout d'une centaine d'itérations





Le célèbre « glider », motif du Jeu de la vie se répétant au bout de 4 itérations, se déplaçant en diagonale

Introduction

Un autre automate cellulaire, Lenia :

Jusque là, approche essentiellement discrète (états, voisinage, découpe du temps...)

Comment passer à une approche continue ?

I. Du Jeu de la vie à Lenia

Définition d'un automate cellulaire :

$$(R, Q, V, \delta)$$

R : réseau

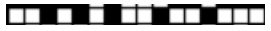
Q : alphabet

$V \subseteq R$: voisinage

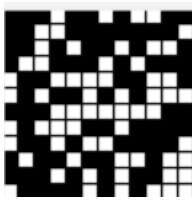
$\delta : Q^{|V|} \to Q$: règle locale

exemples :

$\mathrm{R} = \mathbb{Z}$



$\mathrm{R} = \mathbb{Z}^2$

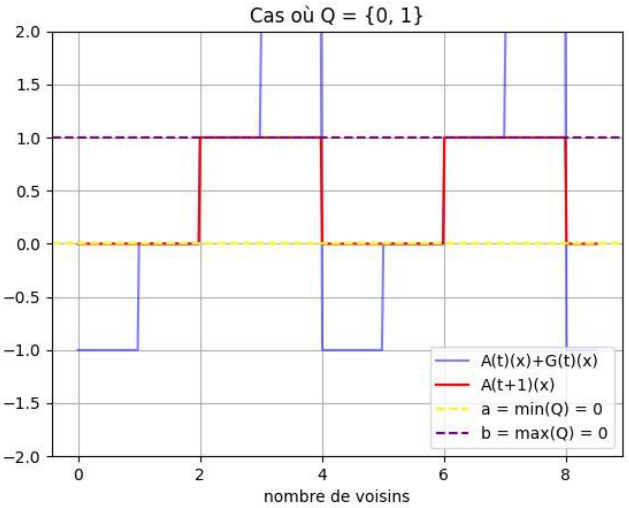


$A_t : R \to Q$ configuration

G_t : fonction de croissance vérifiant

$$A_{t+1}(x) = max(a, min(b, A_t(x) + G_t(x)))$$

avec $a = min(Q)$ et $b = max(Q)$



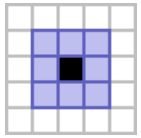
W_t : somme des états voisins

(y compris le centre)

Jeu de la vie

Règles :
Une cellule morte possédant exactement trois cellules voisines vivantes devient vivante (elle naît)
Une cellule vivante ne possédant pas exactement deux ou trois cellules voisines vivantes meurt.

Voisinage du Jeu de la vie
(illustration:https://conwaylife.com/wiki/Larger_than_Life)



$$(\mathbb{Z}^2, \{0,1\}, V, \delta) \text{ avec } V = \{v \in \mathbb{Z}^2, ||v||_\infty \leq 1\}$$

$$V_x = \{x + v, v \in V\} \text{ le voisinage de la cellule } x \in R$$

$$\text{et } \delta(V_x) = A_{t+1}(x) = \begin{cases} 1 & \text{si } A_t(x) = 0 \text{ et } W_t(x) = 3 \\ 1 & \text{si } A_t(x) = 1 \text{ et } W_t(x) \in \{3,4\} \\ 0 & \text{sinon} \end{cases}$$

Un nouvel automate cellulaire : Primordia

Un plus grand nombre d'états

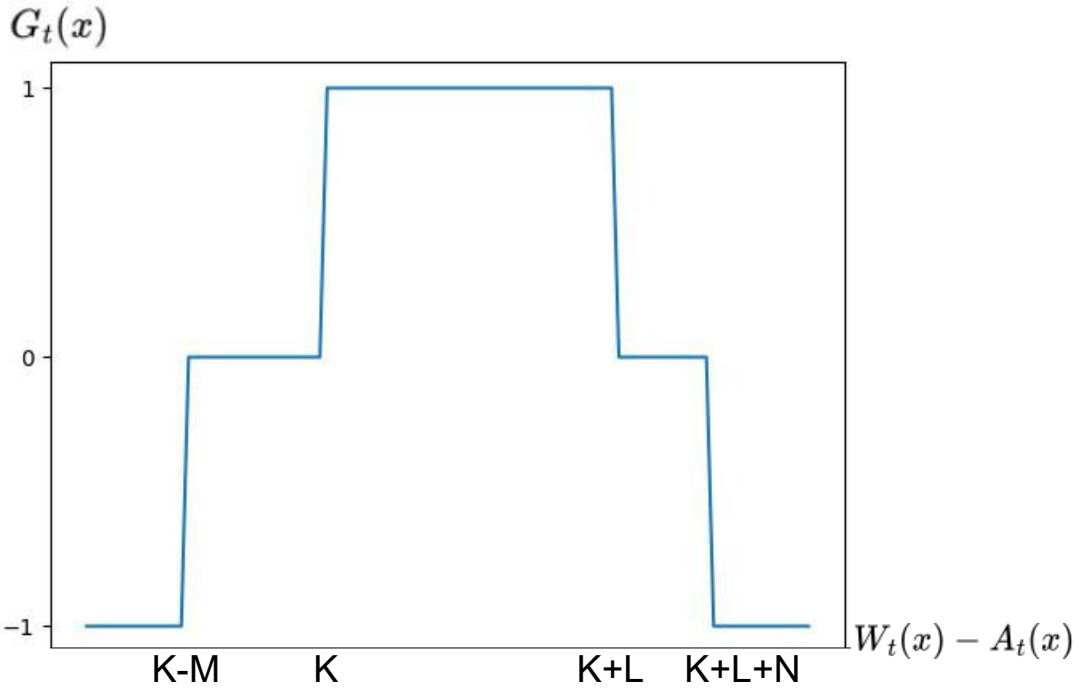
Notation propre à Primordia : M, N, K, L, n+1 états → n/K+L, M, N

$$(\mathbb{Z}^2, [|0,n|], V, \delta)$$

$$Q = \{0,1\} \longrightarrow Q = [|0,n|]$$

$$G_t(x) = \begin{cases} 1 & \text{si } W_t(x) - A_t(x) \in [|K, K+L|] \\ 0 & \text{si } W_t(x) - A_t(x) \in [|K-M, K+L+N|] \setminus [|K, K+L|] \\ -1 & \text{sinon} \end{cases}$$

Jeu de la vie dans Primordia : 1/3+0, 1, 0



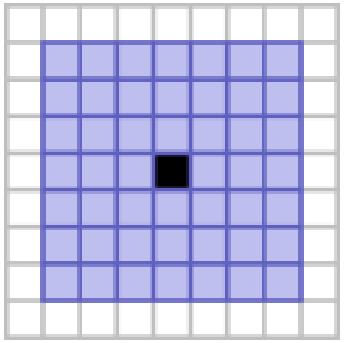
Larger-than-Life (LtL)

Un voisinage plus grand

$$(\mathbb{Z}^2, [|0, n|], V_R, \delta)$$

Notation propre à LtL :
R (rayon du voisinage),
C (nombre n+1 d'états),
S (intervalle de survie),
B (intervalle de naissance)

Voisinage pour R = 3
(https://conwaylife.com/wiki/Larger_than_Life)



$$V = \{v \in \mathbb{Z}^2, ||v||_\infty \leq 1\} \longrightarrow V_R = \{v \in \mathbb{Z}^2, ||v||_\infty \leq R\}$$

$$G_t(x) = \begin{cases} 1 & \text{si } W_t(x) - A_t(x) \in B \\ 0 & \text{si } W_t(x) - A_t(x) \in S \backslash \mathrm{S} \cap B \\ -1 & \text{sinon} \end{cases}$$

Jeu de la vie dans LtL : R1,C2,S2-3,B3

Lenia

Une nouvelle manière de découper le temps

$$(R, T, Q, V, \delta)$$

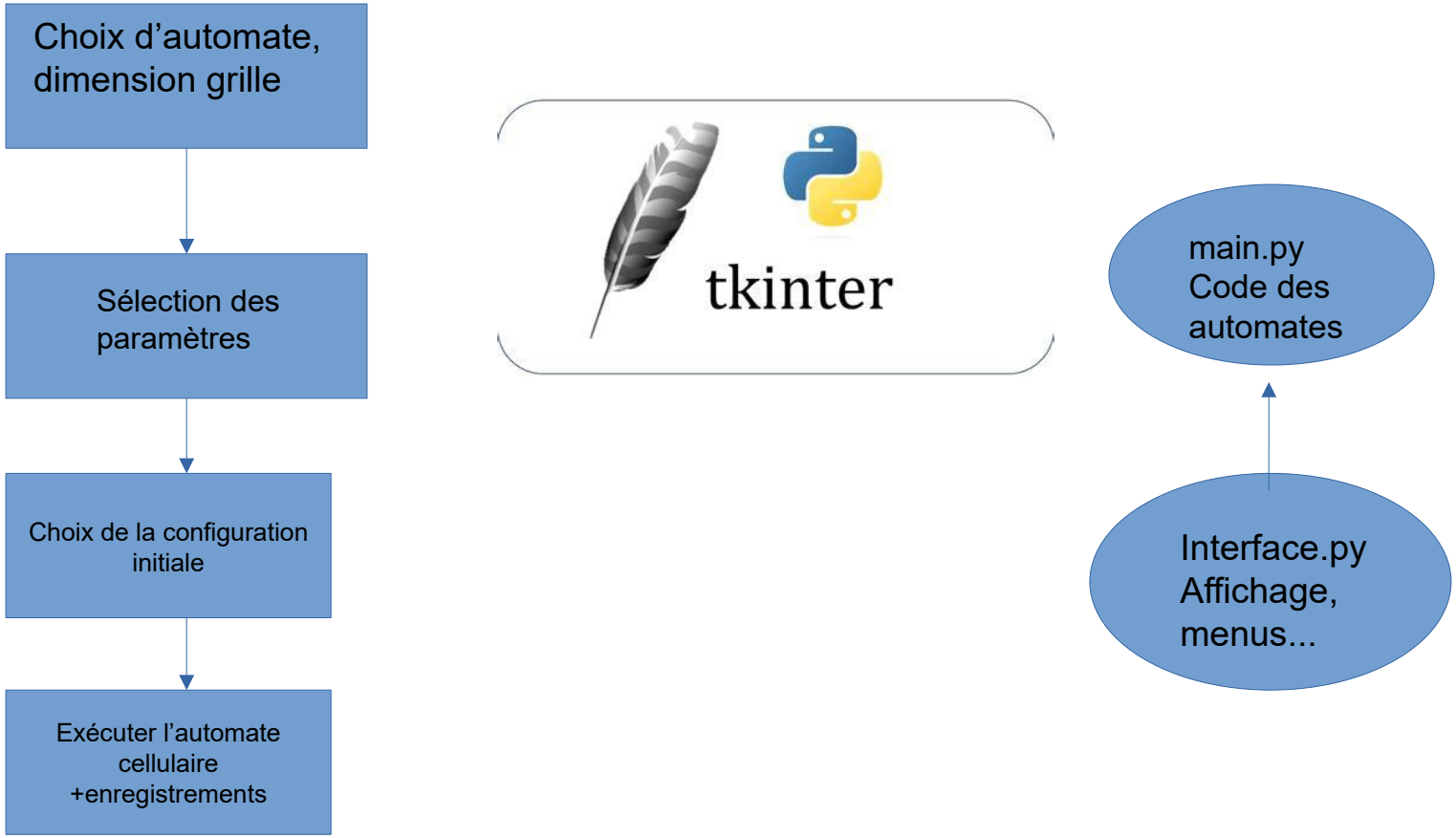
précédemment, $T = \mathbb{N}$ $\longrightarrow$ $T = \Delta t\mathbb{N}$ (avec $\Delta t \in]0, 1]$),

$R = \mathbb{R}^2$ (en pratique $R = \mathbb{Z}^2$), $Q = [0, 1]$

$$A_{t+\Delta t}(x) = max(0, min(1, A_t(x) + \Delta t G_t(x)))$$

II. Implémentation

Organisation des fichiers (version 1), de l'interface graphique et choix du module de l'interface graphique



Méthodes d’affichage

0 1 (ou n, pour n est le plus grand état possible)

En noir et blanc

« Dégradé du bleu vers l’orange »

$$fcolor : \begin{array}{ccc} [0,1] & \longrightarrow & [|0,255|]^3 \\ x & \mapsto & (255x, 255|x - \frac{1}{2}|, 255x) \end{array}$$

Détail technique important concernant le voisinage : la convolution de matrices

$$A = K * M$$

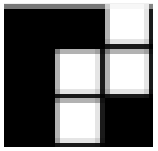
$$A_{i,j} = \sum_{x=-R}^{R} \sum_{y=-R}^{R} K_{x,y} M_{i-x,j-y}$$

Exemple simple avec K de rayon 1 et M donné

$\text{rayon R} = 1$

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} =$$



$\text{rayon R} = 3$

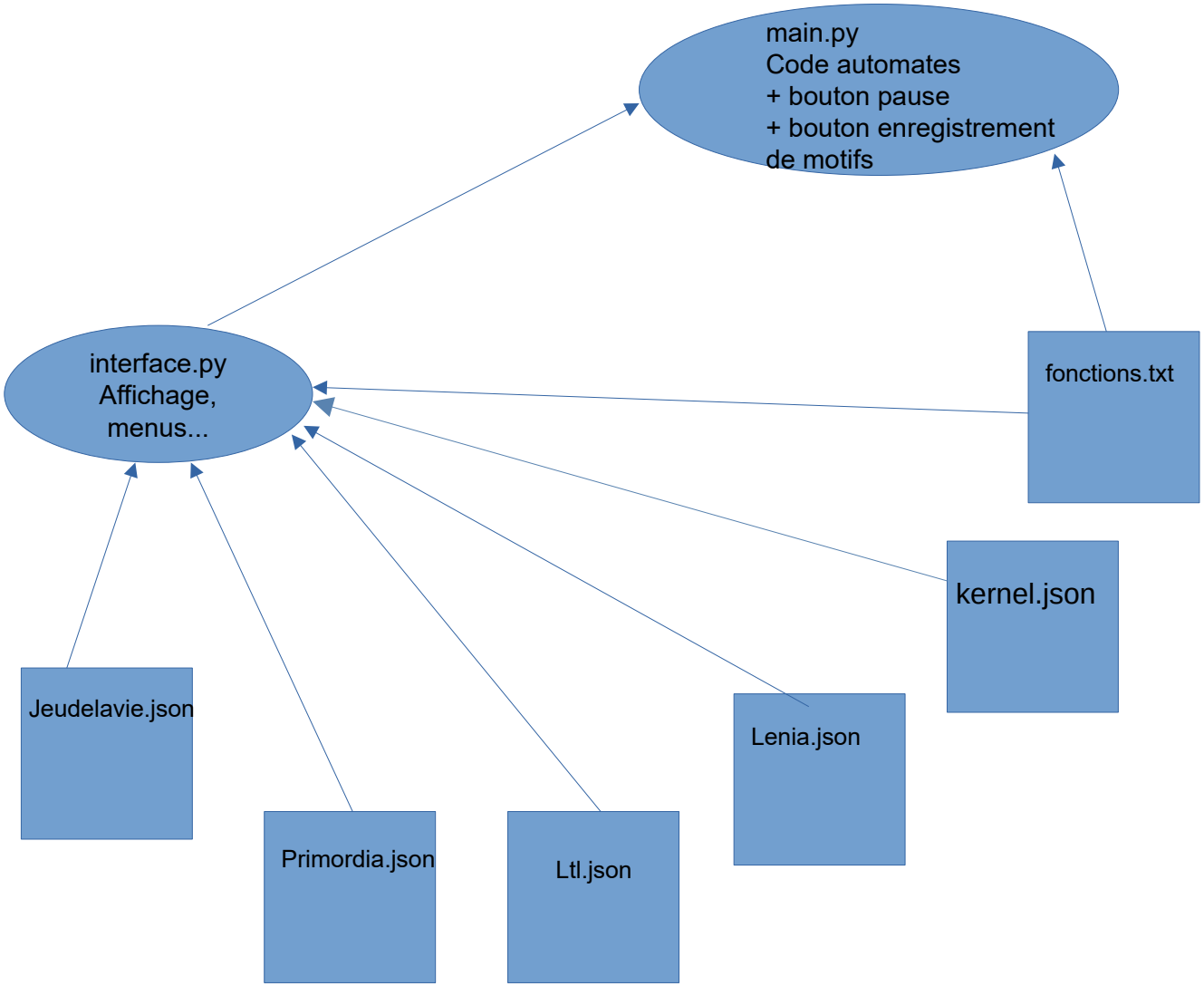
$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Pour (1, 1) la cellule centrale :

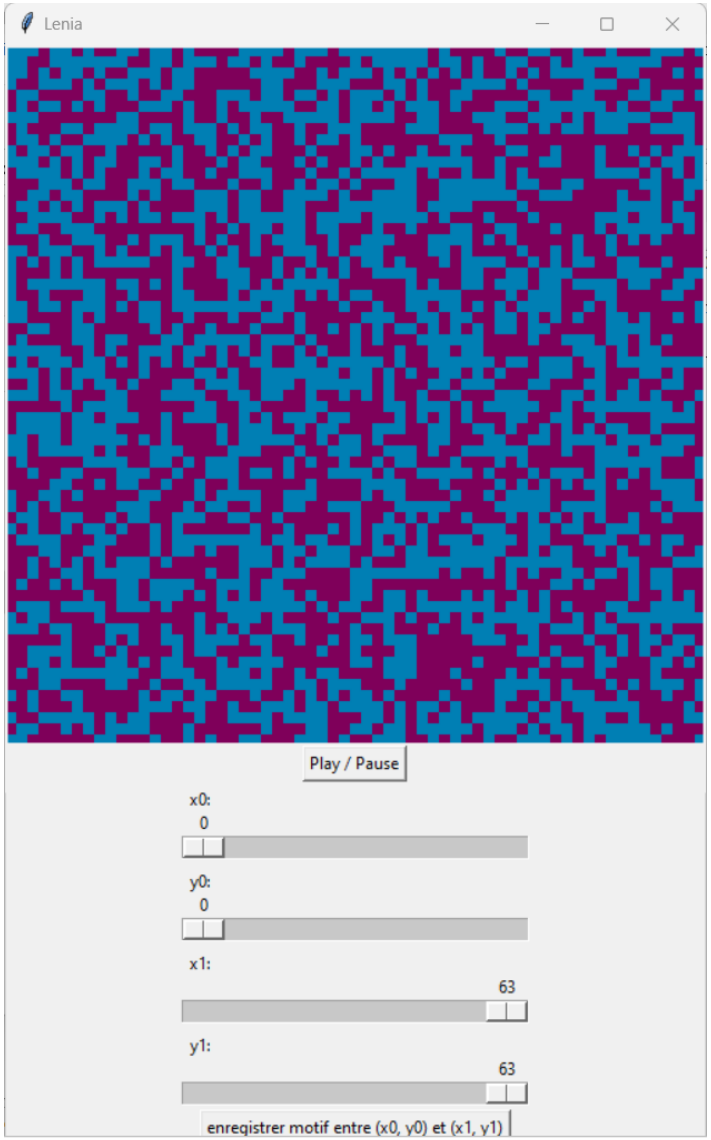
$$\begin{array}{llll} 1 \times 0 & +1 \times 0 & + & 1 \times 1 \\ 1 \times 0 & +0 \times 1 & + & 1 \times 1 \\ 1 \times 0 & +1 \times 1 & + & 1 \times 0 \end{array} = 3$$

voisins (sans compter la cellule (1, 1))

Organisation finale des fichiers



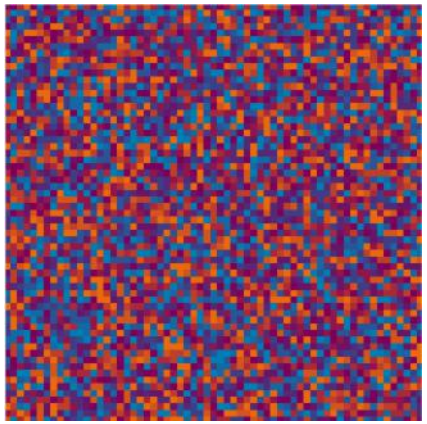
Aperçu de l’interface graphique

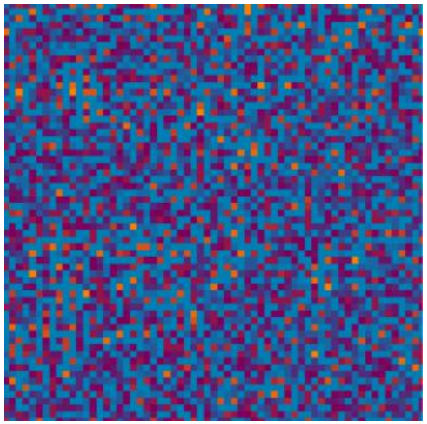


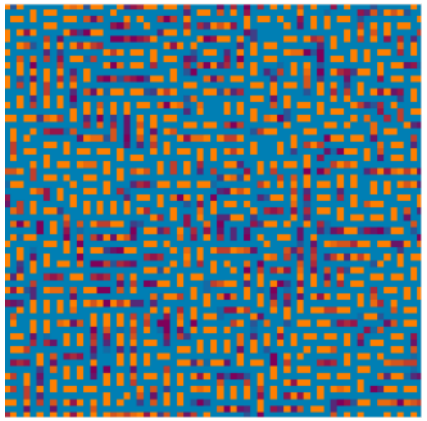
III. Résultats et observations

Premières observations pour des configurations initiales aléatoires

Primordia, 11/10+6, 1, 2



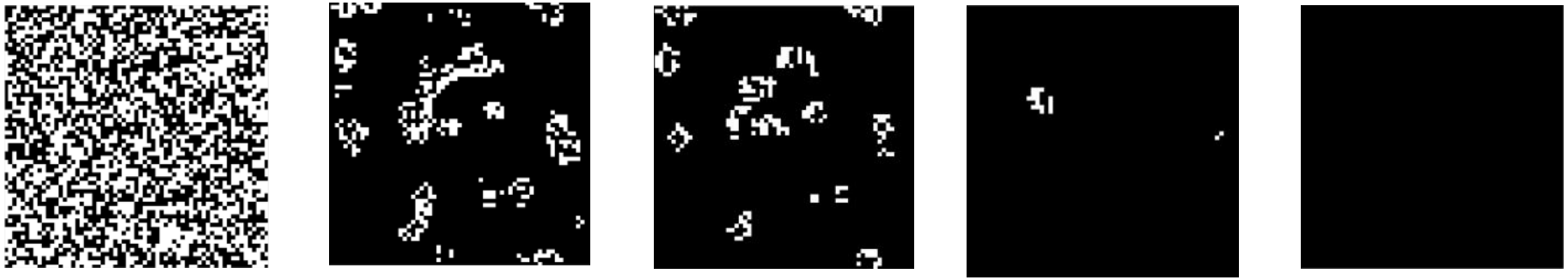




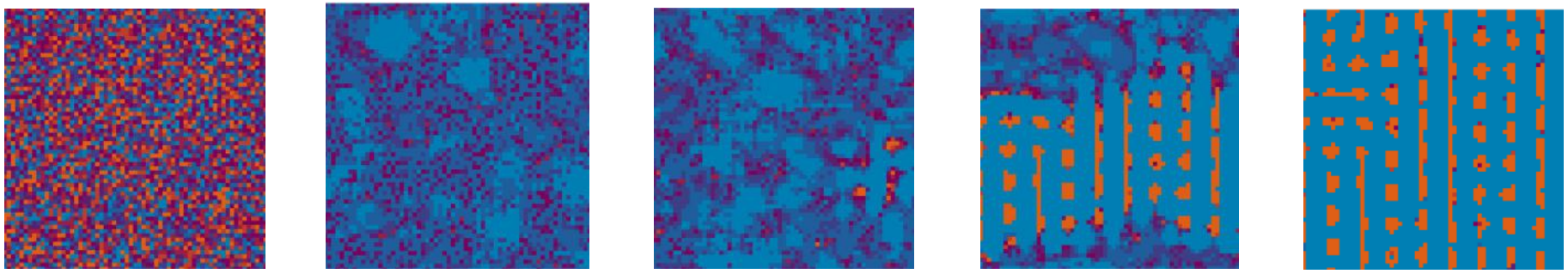
Larger-than-Life

Règle de Bosco : R5,C2,S33-57,B34-45

R2, C2, S7-10, B7-8



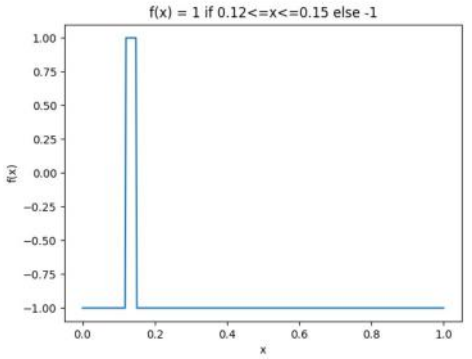
R5, C8, S83-129, B68-115

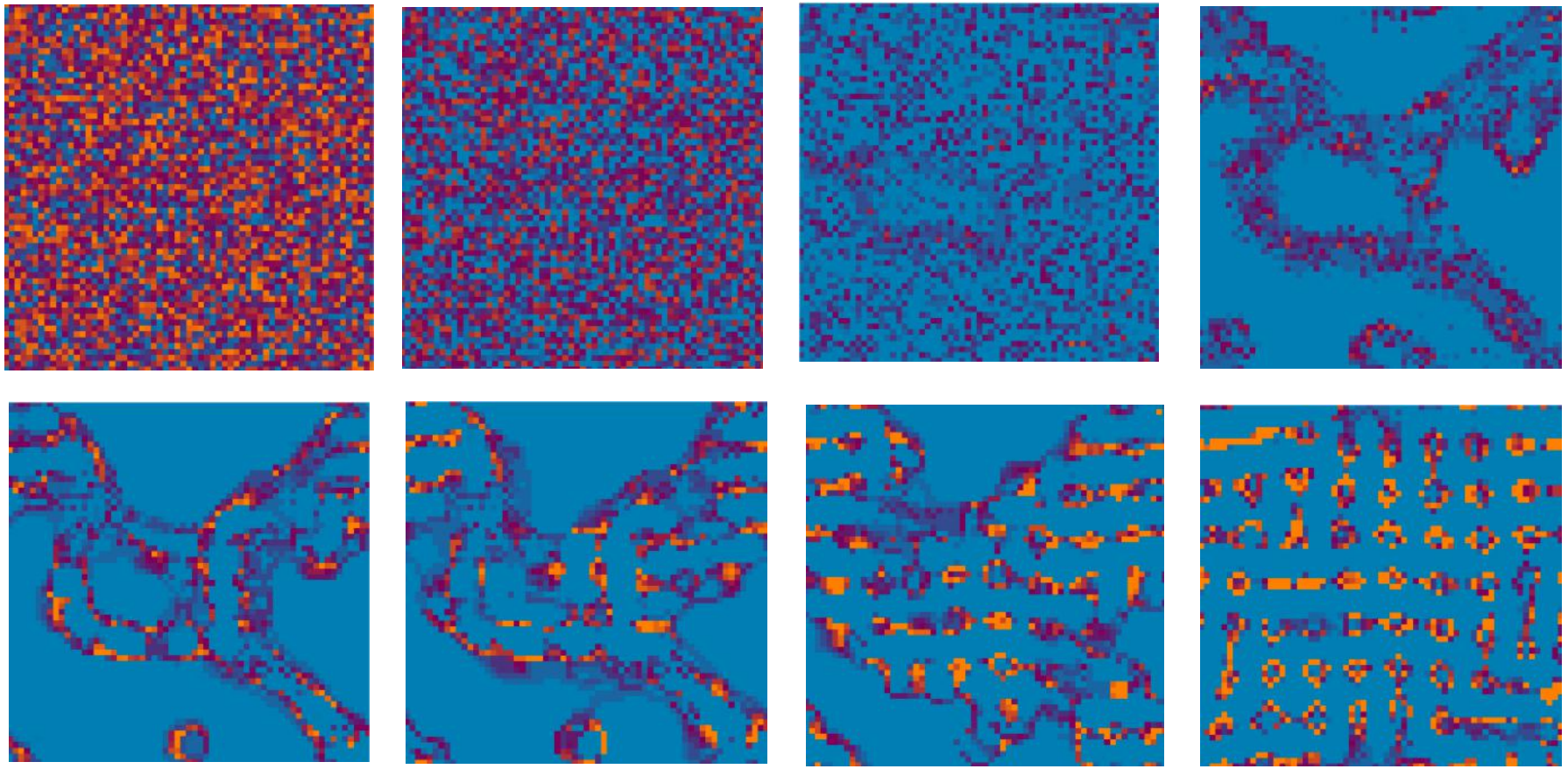


Lenia

Paramètres initiaux : matrice noyau et fonction de croissance

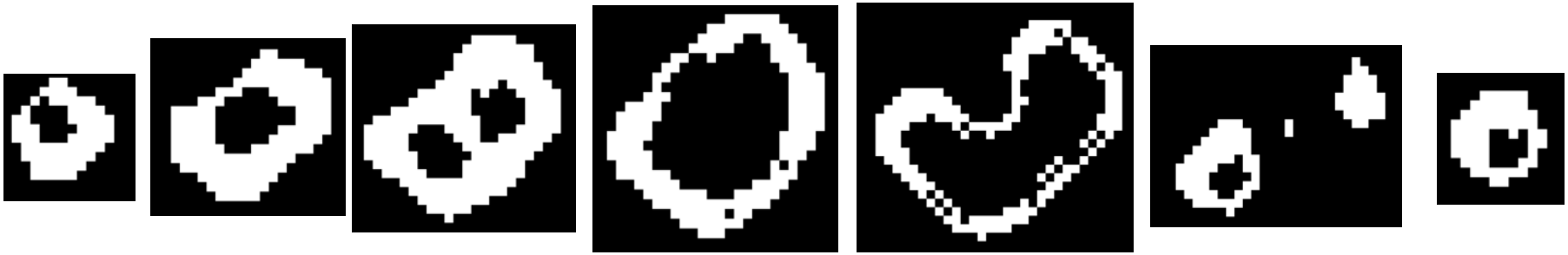
$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$





Observations de différents motifs

Motif « bosco », règle homonyme (LtL)



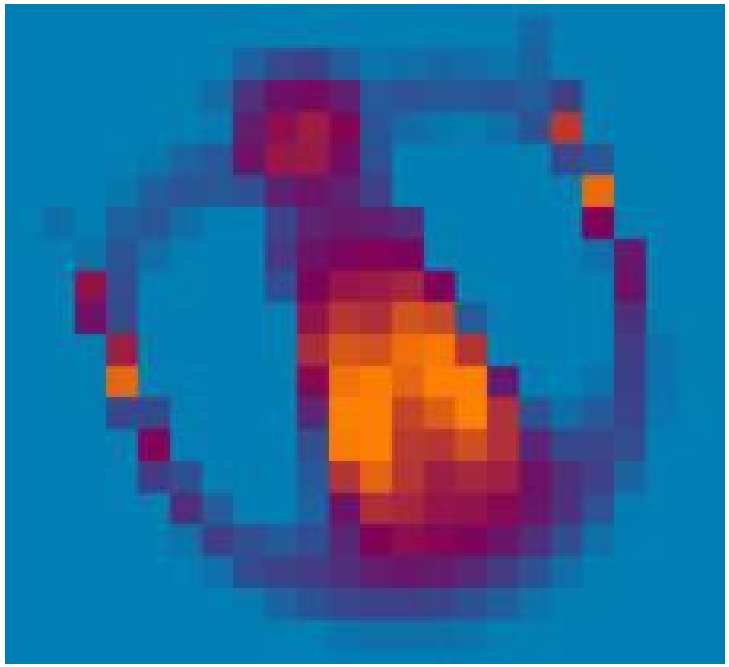
Primordia, 2/3+0, 0, 0

Primordia, 2/5+1, 0, 1

Primordia 3/6+0, 1, 1

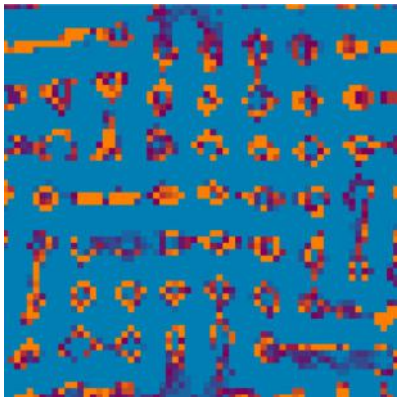
Primordia 5/8+1, 1, 2

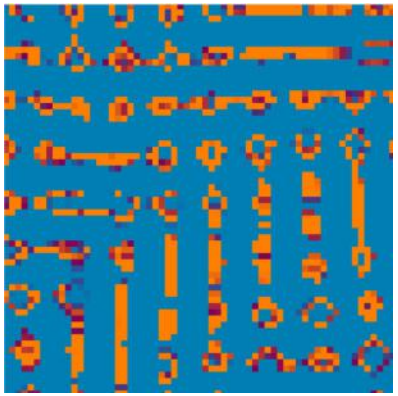
Orbium (Lenia)

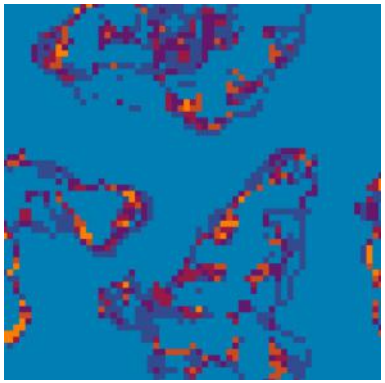


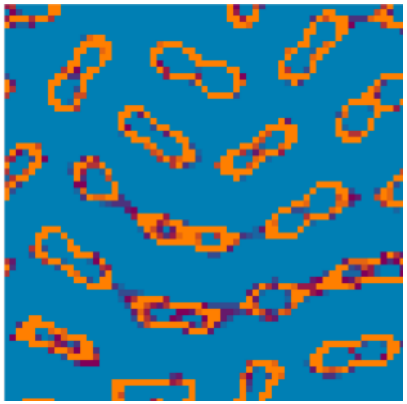
Analyses qualitatives supplémentaires des résultats

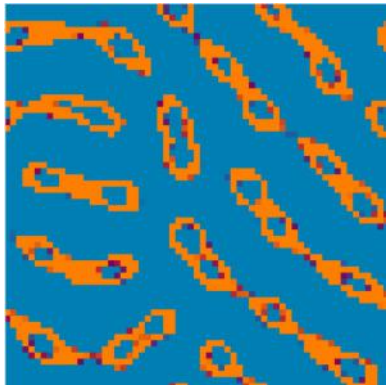
Configuration initiale aléatoire : apparition de formes stables
Lenia : très dépendant des paramètres initiaux



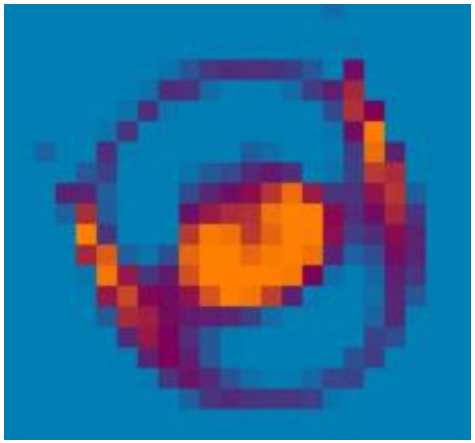






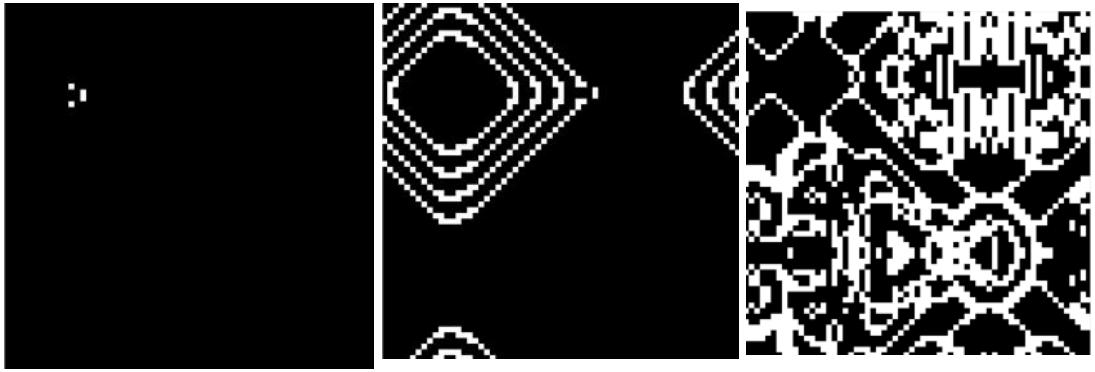


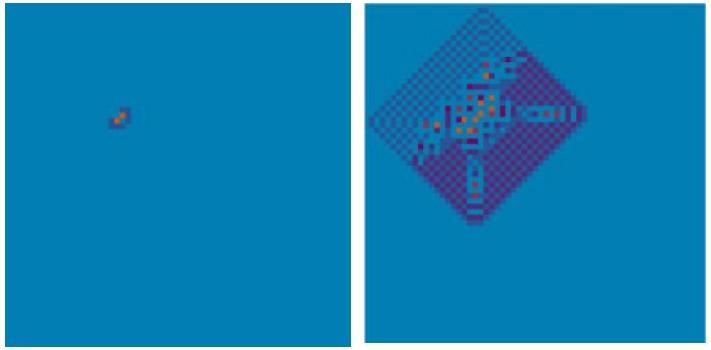
L’orbium : motif fragile



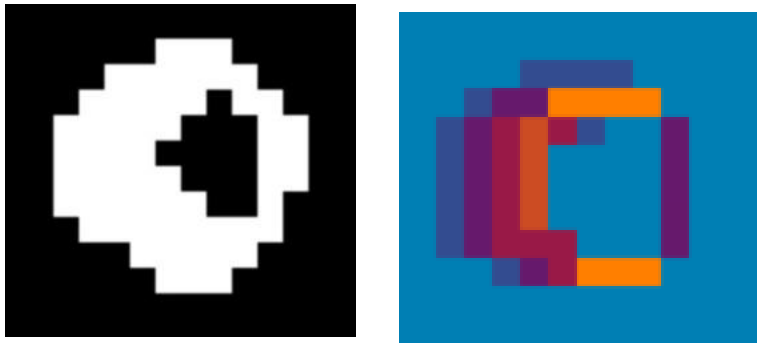
Analyses qualitatives supplémentaires des résultats

« motifs de départ » remplissant la grille
Exemples avec Primordia 1/2+2, 0, 0 et 5/2+2, 1, 1





« faux motifs » disparaissant



Conclusion

Lenia : un automate cellulaire continu mais qui reste sensible à certains paramètres

Interface graphique : buts donnés remplis (affichage et enregistrements)

motifs : plusieurs motifs simples et connus retrouvés

Annexe

Définition formelle d'un automate cellulaire :

On définit un automate cellulaire par un 4-uplet (R, Q, V, δ)

où R est le réseau de l'automate (en général $\mathrm{R} = \mathbb{Z}$ pour une ligne ou $\mathrm{R} = \mathbb{Z}^2$ pour une grille)

Q est son alphabet, l'ensemble des états possibles pris par une case (une cellule)

$V \subseteq R$ est son voisinage (un sous-ensemble fini du réseau)

$\delta : Q^{|V|} \to Q$ est sa règle locale de transition

On pose $V_x = \{x + v, v \in V\}$ le voisinage de la cellule $x \in R$

On pose $A : R \to Q$ la configuration, telle que $A_t(x)$ donne l'état de la cellule x à l'instant t

Pour connaître l'état suivant de la cellule, on applique la règle locale: $A_{t+1}(x) = \delta(V_x)$

On pose enfin G, une fonction de croissance vérifiant:

$A_{t+1}(x) = max(a, min(b, A_t(x) + G_t(x)))$ avec $a = min(Q)$ et $b = max(Q)$

Et W la fonction renvoyant la somme des états des voisins de x:

$$W_t(x) = \sum_{v \in V} A_t(x + v) = \sum_{w \in V_x} A_t(w)$$

Annexe

Fonctions pour le Jeu de la vie

$$A_{t+1}(x) = max(0, min(1, A_t(x) + G_t(x))) \text{ avec}$$

$$G_t(x) = \begin{cases} 1 & \text{si } W_t(x) - x = 3 \\ 0 & \text{si } W_t(x) - x = 2 \text{ (une cellule ayant exactement 2 voisins garde le même état)} \\ -1 & \text{sinon} \end{cases}$$

```
def growth1(n):
    """n: nb de cellules voisines"""
    if n==3:
        return 1
    if n<2 or n>3:
        return -1
    return 0


def update1(A):
    """jeu de la vie -> états discrets, espace continu, temps discret"""
    taille = variables["taille"]
    noyau = np.asarray([[1,1,1],
                        [1,0,1],
                        [1,1,1]]) #noyau permettant de calculer la somme des voisins, donc le nombre de voisins
    voisinage = scipy.signal.convolve2d(A, noyau, mode='same', boundary='wrap') #calcule la somme des voisins pour chaque cellule
    #l'option "same" indique que la matrice résultante fait la même taille que A.
    #L'option "wrap" signifie que l'on prolonge les cotés de A par les cellules de l'autre coté pour les calculs
    resultat = np.zeros_like(A)
    for i in range(taille):
        for j in range(taille):
            # Récupération du nombre de voisins vivants pour la cellule (i, j)
            voisins = voisinage[i][j]

            #règles:
            #Une cellule morte possédant exactement trois cellules voisines vivantes devient vivante (elle naît)
            #Une cellule vivante ne possédant pas exactement deux ou trois cellules voisines vivantes meurt
            delta = growth1(voisins)
            resultat[i][j] = clip(A[i][j]+delta, 0, 1)

    return resultat
```

Annexe

Fonctions pour Primordia

```
def growth2(n):
    """n: somme des voisins"""
    K, L, M, N  = variables["primordia_params"]
    if K <= n <= K+L:
        return 1
    elif n < K-M or n > K+L+N:
        return -1
    else:
        return 0

def update2(A):
    """Primordia"""
    taille = variables["taille"]
    primordia_states = variables["primordia_states"]
    noyau = np.asarray([[1,1,1],
                        [1,0,1],
                        [1,1,1]])
    resultat = np.zeros_like(A)
    voisinage = scipy.signal.convolve2d(A, noyau, mode='same', boundary='wrap')
    for i in range(taille):
        for j in range(taille):
            voisins = voisinage[i][j]
            delta = growth2(voisins)
            resultat[i][j] = clip(A[i][j]+delta, 0, primordia_states-1)
    return resultat
```

Annexe

Fonctions pour Larger-than-Life

```
def growth3(n):
    """ fonction pour LtL"""
    ltl_birth = variables["ltl_birth"]
    ltl_survival = variables["ltl_survival"]
    b1, b2 = ltl_birth
    s1, s2 = ltl_survival
    score = 0
    if b1<=n<=b2:
        score+=1
    if n<s1 or n>s2:
        score-=1
    return score

def update3(A):
    """LtL"""
    taille = variables["taille"]
    R_ltl = variables["R"]
    ltl_states = variables["ltl_states"]
    noyau = np.ones((2*R_ltl+1, 2*R_ltl+1))
    noyau[R_ltl][R_ltl] = 0 #IMPORTANT, ne pas compter le centre
    resultat = np.zeros_like(A)
    voisinage = scipy.signal.convolve2d(A, noyau, mode='same', boundary='wrap')
    for i in range(taille):
        for j in range(taille):
            voisins = voisinage[i][j]
            delta = growth3(voisins)
            resultat[i][j] = clip(A[i][j]+delta, 0, ltl_states-1)
    return resultat
```

Annexe

Fonctions pour Lenia

```
def smooth_kernel(R, mu, sigma):
    noyau = np.zeros((2*R+1, 2*R+1))
    distance=lambda x,y: np.sqrt((x-R)**2+(y-R)**2)/R
    for i in range(len(noyau)):
        for j in range(len(noyau)):
            d=distance(i, j)
            if 0<d<=1:
                noyau[i][j] = gauss(d, mu, sigma)
    return noyau

def growth4(x):
    """ fonction pour Lenia"""
    return variables["growth"](x) #fonction choisie dans le menu

def update4(A):
    """Lenia"""
    taille = variables["taille"]
    T = variables["T"]

    noyau0 = variables["noyau"]
    noyau = noyau0/np.sum(noyau0)
    resultat = np.zeros_like(A)
    voisinage = scipy.signal.convolve2d(A, noyau, mode='same', boundary='wrap')
    for i in range(taille):
        for j in range(taille):
            #print(i)
            voisins = voisinage[i][j]
            delta = growth4(voisins)
            resultat[i][j] = clip(A[i][j]+(1/T)*delta, 0, 1)
    return resultat
```

Création d'un noyau K en forme « d'anneau lisse » :

$$g_{\mu,\sigma} : x \mapsto e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$$d : (x, y) \mapsto \frac{\sqrt{(x-R)^2 + (y-R)^2}}{R}$$

$$\begin{cases} \text{si } d \leq 1, & K_{x,y} = g_{\mu,\sigma}(d(x,y)) \\ \text{si } d > 1, & K_{x,y} = 0 \end{cases}$$

main.py

```python
from interface import *
import pprint

np.random.seed(9)

variables = {"matrice":None,
             "taille":None,
             "R":None,
             "primordia_states":None,
             "ltl_states":None,
             "ltl_birth":None,
             "ltl_survival":None,
             "T":None,
             "play":False,
             "colour":False,
             "growth":None,
             "noyau":None,
             "widgets":[],
             "id":None}


def print_mat(A):
    '''affichage de matrice'''
    pprint.pp(list(A))
    print()

def gauss(x, m, s):
        return np.exp(-((x-m)/s)**2 /2)

def clip(x, x_min, x_max):
    if x>x_max:
        return x_max
    else:
        return max(x_min, x)


def growth1(n):
    """n: nb de cellules voisines"""
    if n==3:
        return 1
    if n<2 or n>3:
        return -1
    return 0


def update1(A):
    """jeu de la vie -> états discrets, espac continu, temps discret"""
    taille = variables["taille"]
    noyau = np.asarray([[1,1,1],
                        [1,0,1],
                        [1,1,1]]) #noyau permettant de calculer la somme des voisins, donc le nombre de voisins
    voisinage = scipy.signal.convolve2d(A, noyau, mode='same', boundary='wrap') #calcule la somme des voisins pour chaque cellule
    resultat = np.zeros_like(A)
    for i in range(taille):
        for j in range(taille):
            # Récupération du nombre de voisins vivants pour la cellule (i, j)
            voisins = voisinage[i][j]

            #règles:
            #Une cellule morte possédant exactement trois cellules voisines vivantes devient vivante (elle naît)
            #Une cellule vivante ne possédant pas exactement deux ou trois cellules voisines vivantes meurt
            delta = growth1(voisins)
            resultat[i][j] = clip(A[i][j]+delta, 0, 1)

    return resultat


def growth2(n):
    """n: somme des voisins"""
    K, L, M, N  = variables["primordia_params"]
    if K <= n <= K+L:
        return 1
    elif n < K-M or n > K+L+N:
        return -1
    else:
        return 0

def update2(A):
    """Primordia"""
    taille = variables["taille"]
    primordia_states = variables["primordia_states"]
    noyau = np.asarray([[1,1,1],
                        [1,0,1],
                        [1,1,1]])
    resultat = np.zeros_like(A)
    voisinage = scipy.signal.convolve2d(A, noyau, mode='same', boundary='wrap')
    for i in range(taille):
        for j in range(taille):
            voisins = voisinage[i][j]
            delta = growth2(voisins)
            resultat[i][j] = clip(A[i][j]+delta, 0, primordia_states-1)
    return resultat


def growth3(n):
    """ fonction pour LtL"""
    ltl_birth = variables["ltl_birth"]
    ltl_survival = variables["ltl_survival"]
    b1, b2 = ltl_birth
    s1, s2 = ltl_survival
    score = 0
    if b1<=n<=b2:
        score+=1
    if n<s1 or n>s2:
        score-=1
    return score

def update3(A):
    """LtL"""
    taille = variables["taille"]
    R_ltl = variables["R"]
    ltl_states = variables["ltl_states"]
    noyau = np.ones((2*R_ltl+1, 2*R_ltl+1))
    noyau[R_ltl][R_ltl] = 0 #IMPORTANT, ne pas compter le centre
    resultat = np.zeros_like(A)
    voisinage = scipy.signal.convolve2d(A, noyau, mode='same', boundary='wrap')
    for i in range(taille):
        for j in range(taille):
            voisins = voisinage[i][j]
            delta = growth3(voisins)
            resultat[i][j] = clip(A[i][j]+delta, 0, ltl_states-1)
    return resultat



def smooth_kernel(R, mu, sigma):
    noyau = np.zeros((2*R+1, 2*R+1))
    distance=lambda x,y: np.sqrt((x-R)**2+(y-R)**2)/R
    for i in range(len(noyau)):
        for j in range(len(noyau)):
            d=distance(i, j)
            if 0<d<=1:
                noyau[i][j] = gauss(d, mu, sigma)
    return noyau

def growth4(x):
    """ fonction pour Lenia"""
    return variables["growth"](x) #fonction choisie dans le menu

def update4(A):
    """Lenia"""
    taille = variables["taille"]
    T = variables["T"]

    noyau0 = variables["noyau"]
    noyau = noyau0/np.sum(noyau0)
    resultat = np.zeros_like(A)
    voisinage = scipy.signal.convolve2d(A, noyau, mode='same', boundary='wrap')
    for i in range(taille):
        for j in range(taille):
            #print(i)
            voisins = voisinage[i][j]
            delta = growth4(voisins)
            resultat[i][j] = clip(A[i][j]+(1/T)*delta, 0, 1)
    return resultat



def lancement(taille, variante_id, states=None, R=None, T=None, colour=False, ltl_birth=None, ltl_survival=None, primord=[], growth="1 if 0.12<=x<=0.15 else -1", noyau=None, matrice=None, smooth_ = []):
    '''initialisation des paramètres depuis l'interface graphique'''
    print(smooth_)
    variables["colour"] = colour
    variables["id"] = variante_id
    #variables["growth"] = growth
    variables["R"] = R
    variables["T"]=T
    variables["ltl_survival"] = ltl_survival
    variables["ltl_birth"] = ltl_birth
    s='variables["growth"] = lambda x: '+growth
    exec(s)
    if variante_id == 0:
        #jeu de la vie
        A = np.random.randint(2, size=(taille, taille))
        update = update1
        show_mat = show_mat1
    elif variante_id == 1:
        #primordia
        variables["primordia_states"] = states
        variables["primordia_params"] = primord
        A = np.random.randint(variables["primordia_states"], size=(taille, taille))
        update = update2
        if not variables["colour"]:
            show_mat = lambda x: show_mat2(x, variables["primordia_states"])
        else:
            show_mat = lambda x: show_mat3(x/(variables["primordia_states"]))
    elif variante_id == 2:
        #ltl
        variables["ltl_states"] = states
        A = np.random.randint(variables["ltl_states"], size=(taille, taille))
        update = update3
        if not variables["colour"]:
            show_mat = lambda x: show_mat2(x, (variables["ltl_states"]))
        else:
            show_mat = lambda x: show_mat3(x/(variables["ltl_states"]))
    else:
        #lenia
        variables["noyau"] = noyau
        if len(noyau) == 0:
            print(smooth_)
            R, m, s = smooth_
```

```
            R, m, s = smooth_
            variables["noyau"] = smooth_kernel(R, m, s)
        A=np.random.randint(100, size=(taille, taille))
        A=A/100
        update = update4
        if variables["colour"]:
            show_mat = show_mat3
        else:
            show_mat = lambda x: show_mat2((100*x).astype(int), 100)#on convertit les états entre 0 et 1 à des entiers entre 0 et 100
    if matrice is None:
        variables["matrice"] = A
    else:
        variables["matrice"] = matrice
    variables["taille"] = taille
    play=Button(tk, text="Play / Pause", command=lambda: (variables.update({"play": not variables["play"]}), loop(update, show_mat)))
#bouton pause
    play.pack()
    enregistreX = Scale(tk, orient="horizontal", from_=0, to=(taille-1), resolution=1, length=250, label="x0:")
    enregistreX.pack()
    enregistreY = Scale(tk, orient="horizontal", from_=0, to=(taille-1), resolution=1, length=250, label="y0:")
    enregistreY.pack()
    enregistreX2 = Scale(tk, orient="horizontal", from_=0, to=(taille-1), resolution=1, length=250, label="x1:")
    enregistreX2.pack()
    enregistreY2 = Scale(tk, orient="horizontal", from_=0, to=(taille-1), resolution=1, length=250, label="y1:")
    enregistreY2.pack()
    enregistreX2.set(taille-1)
    enregistreY2.set(taille-1)
    btnenregistre = Button(tk, text="enregistrer motif entre (x0, y0) et (x1, y1)", command=lambda:enregistrer())
    btnenregistre.pack()
    variables["widgets"] = [enregistreX, enregistreY, enregistreX2, enregistreY2]
    loop(update, show_mat)


def enregistrer():
    enregistreX, enregistreY, enregistreX2, enregistreY2 = variables["widgets"]
    taille = variables["taille"]
    x0 = enregistreX.get()
    y0 = enregistreY.get()
    x1 = enregistreX2.get()
    y1 = enregistreY2.get()
    x0, x1 = min(x0, x1), max(x0, x1)
    y0, y1 = min(y0, y1), max(y0, y1)
    mat = variables["matrice"]
    var_id = variables["id"]
    matrix = [[0 for i in range(x1-x0+1)] for j in range(y1-y0+1)]
    for y in range(y1-y0+1):
        for x in range(x1-x0+1):
            if var_id == 3:#lenia
                matrix[y][x] = float(mat[y+y0][x+x0])
            else:
                matrix[y][x] = int(mat[y+y0][x+x0])
    file = fichiers[variables["id"]]
    enregistrements = {}
    if file in os.listdir("."):
        with open(file, "r") as f:
            contenu = f.read()
            contenu = contenu.replace("\n", "")
            enregistrements = json.loads(contenu)
            f.close()
    print(len(matrix))
    enregistrements["nouveau motif ({}, {}), ({}, {})".format(x0, y0, x1, y1)] = matrix
    print(enregistrements)
    with open(file, "w") as f:
        json.dump(enregistrements, f)
        f.close()
    enregistreX.set(0)
    enregistreY.set(0)
    enregistreX2.set(taille-1)
    enregistreY2.set(taille-1)


def loop(update, show_mat):
    global energistreX, enregistreY, energistreX2, enregistreY2, btnenregistre
    #print(variables["can_record"])
    if variables["play"]:
        tk.after(2, lambda: iteration(update, show_mat))
    else:#pause
        A = variables["matrice"]
        show_mat(A)
        #print(variables["primordia_params"])


def iteration(update, show_mat):#sinon RecursionError
    A = variables["matrice"]
    show_mat(A)
    variables["matrice"] = update(A)
    loop(update, show_mat)

start_ui(callback0=lancement)

```

interface.py

```python
from tkinter import *
import matplotlib.pyplot as plt
import numpy as np
import math
import scipy
import os
import json

tk = Tk()
tk.title("Lenia")
tk.configure(cursor = "hand2")
taille_canevas = 500
w = Canvas(tk,width=taille_canevas,height=taille_canevas,bg="black")
w.pack()

var_ = {"taille":-1}
fichiers = ["Jeudelavie.json", "Primordia.json", "Ltl.json", "Lenia.json"]

def convert_color(n):
    a = str(hex(int(n)))[2:]
    if len(a)==1:
        a="0"+a
    return a


def f_color(x):
    return 255*x, 255*(abs(x-0.5)), 180*(1-x)


def show_mat1(A):
    w.itemconfig("tile", state='hidden')
    unite = int(1+taille_canevas/len(A))
    for i in range(len(A)):
        for j in range(len(A[0])):
            if A[i][j] == 1:
                w.create_rectangle(j*unite, i*unite, (j+1)*unite, (i+1)*unite, fill="white", tag="tile")
    tk.update()


def show_mat2(A, states):
    color_unit = 255//(states-1)
    w.itemconfig("tile", state='hidden')
    unite = int(1+taille_canevas/len(A))
    for i in range(len(A)):
        for j in range(len(A[0])):
            color = str(hex(int(A[i][j])*color_unit))[2:]
            if len(color) == 1:
                color = "0"+color
            w.create_rectangle(j*unite, i*unite, (j+1)*unite, (i+1)*unite, outline="#"+3*color, fill="#"+3*color, tag="tile")
    tk.update()


def show_mat3(A):
    w.itemconfig("tile", state='hidden')
    unite = int(1+taille_canevas/len(A))
    for i in range(len(A)):
        for j in range(len(A[0])):
            r, g, b = list(map(convert_color, f_color(A[i][j])))
            w.create_rectangle(j*unite, i*unite, (j+1)*unite, (i+1)*unite, outline="#"+r+g+b, fill="#"+r+g+b, tag="tile")
    tk.update()


variantes = ["Jeu de la vie", "Primordia", "LtL", "Lenia"]
variante_id = 0


def select_taille():
    global variable, taille, btn2, opt
    taille = echelleTaille.get()
    var_["taille"] = taille
    echelleTaille.destroy()
    btn1.destroy()
    variable = StringVar()
    variable.set(variantes[0])
    opt = OptionMenu(tk, variable, *variantes)
    opt.place(x=200, y=250)
    btn2 = Button(tk, text="choisir la variante", command=select_variante)
    btn2.pack()


def select_variante():
    global KPrimordia, LPrimordia, MPrimordia, NPrimordia, btnChoix0, btnChoix1, variante_id, btnPrimordia, optPrimordia, optLtl, optLtl0, optLtl1, optLtl2, optLtl3, optLtl4, btnLtl, optLenia0, variableLenia, optLenia1, variableLenia2, optLenia, btnLenia, col, couleur
    variante = variable.get()
    for i in range(len(variantes)):
        if variantes[i] == variante:
            variante_id = i
    btn2.destroy()
    opt.destroy()
    couleur = StringVar()
    if variante_id == 1:
        optPrimordia = Scale(tk, orient="horizontal", from_=2, to=50, resolution=1, length=500, label="états:")
        optPrimordia.set(6)
        optPrimordia.place(rely=0.0)
        KPrimordia = Scale(tk, orient="horizontal", from_=0, to=20, resolution=1, length=500, label="K:")
        KPrimordia.set(2)
        KPrimordia.place(rely=0.1)
        LPrimordia = Scale(tk, orient="horizontal", from_=0, to=20, resolution=1, length=500, label="L:")
        LPrimordia.set(0)
        LPrimordia.place(rely=0.2)
        MPrimordia = Scale(tk, orient="horizontal", from_=0, to=20, resolution=1, length=500, label="M:")
        MPrimordia.set(0)
        MPrimordia.place(rely=0.3)
        NPrimordia = Scale(tk, orient="horizontal", from_=0, to=20, resolution=1, length=500, label="N:")
        NPrimordia.set(0)
        NPrimordia.place(rely=0.4)
        col = Checkbutton(tk, text="en couleur", variable=couleur, onvalue="selected", offvalue="not selected")
        couleur.set("not selected")
        col.pack()
        btnPrimordia = Button(tk, text="Sélectionner le nombre d'états", command=lambda:start_Primordia(taille, variante_id))
        btnPrimordia.pack()
    elif variante_id == 2:
        col = Checkbutton(tk, text="en couleur", variable=couleur, onvalue="selected", offvalue="not selected")
        couleur.set("not selected")
        col.pack()
        optLtl = Scale(tk, orient="horizontal", from_=2, to=50, resolution=1, length=500, label="nombre d'états:")
        optLtl.set(2)
        optLtl.place(rely=0.0)
        optLtl0 = Scale(tk, orient="horizontal", from_=1, to=20, resolution=1, length=500, label="rayon:")
        optLtl0.set(5)
        optLtl0.place(rely=0.1)
        optLtl1 = Scale(tk, orient="horizontal", from_=0, to=250, resolution=1, length=500, label="valeur minimale de naissance")
        optLtl1.set(33)
        optLtl1.place(rely=0.2)
        optLtl2 = Scale(tk, orient="horizontal", from_=0, to=250, resolution=1, length=500, label="valeur maximale de naissance")
        optLtl2.set(57)
        optLtl2.place(rely=0.3)
        optLtl3 = Scale(tk, orient="horizontal", from_=0, to=250, resolution=1, length=500, label="valeur minimale de survie")
        optLtl3.set(34)
        optLtl3.place(rely=0.4)
        optLtl4 = Scale(tk, orient="horizontal", from_=0, to=250, resolution=1, length=500, label="valeur maximale de survie")
        optLtl4.set(45)
        optLtl4.place(rely=0.5)
        btnLtl = Button(tk, text="confirmer la sélection de paramètres", command=lambda:start_Ltl(taille, variante_id))
        btnLtl.pack()
        tk.update()
    elif variante_id == 3:
        fonctions =["1 if 0.12<=x<=0.15 else -1"]
        m = [[1 for i in range(11)] for j in range(11)]
        m[5][5] = 0
        noyaux = {"R = 5 (11x11), centre non inclus": m}
        if "fonctions.txt" in os.listdir("."):
            with open("fonctions.txt", "r") as file:
                contenu = file.read()
                fonctions = contenu.split("\n")
                file.close()
        if "kernel.json" in os.listdir("."):
            with open("kernel.json", "r") as file:
                contenu = file.read()
                contenu2 = contenu.replace("\n", "")
                noyaux = json.loads(contenu2)
                file.close()
        noyaux["smooth kernel"] = []
        noy = list(noyaux.keys())

        variableLenia = StringVar()
        variableLenia.set(fonctions[0])
        optLenia0 = OptionMenu(tk, variableLenia, *fonctions)
        optLenia0.place(x=200, y=250)
        variableLenia2 = StringVar()
        variableLenia2.set(noy[0])
        optLenia1 = OptionMenu(tk, variableLenia2, *noy)
        optLenia1.place(x=200, y=350)
        col = Checkbutton(tk, text="en couleur", variable=couleur, onvalue="selected", offvalue="not selected")
        couleur.set("selected")
        col.pack()
        optLenia = Scale(tk, orient="horizontal", from_=1, to=30, resolution=1, length=250, label="fraction de temps:")
        optLenia.set(10)
        optLenia.pack()
        btnLenia = Button(tk, text="confirmer la fonction de croissance", command=lambda:start_Lenia(taille, variante_id, noyaux))
        btnLenia.pack()
    else:
        if "Jeudelavie.json" in os.listdir("."):
            btnChoix0 = Button(tk, text="position de départ aléatoire", command=lambda:(deleteChoixPos(),callback(taille, variante_id)))
            btnChoix0.pack()
            btnChoix1 = Button(tk, text="position de départ indiquée manuellement", command=lambda:start_ajoutMotifs(taille, variante_id, aleatoire=False,
                               lancer_=lambda A:callback(taille, variante_id, matrice=A)))
            btnChoix1.pack()
        else:
            callback(taille, variante_id)#permet de revenir à la fonction lancement() dans main.py


def deleteChoixPos():
    btnChoix0.destroy()
    btnChoix1.destroy()


def start_ajoutMotifs(taille, variante_id, aleatoire, lancer_, etats=2):
    global variablemotif, optmotif, btnAjoutmotif, btnFinmotif, Xmotif, Ymotif, lstmotifs, motifs, matriceinit
    deleteChoixPos()
    if aleatoire:
        callback(taille, variante_id)
    else:
        with open(fichiers[variante_id], "r") as file:
            contenu = file.read()
            contenu = contenu.replace("\n", "")
            motifs = json.loads(contenu)
            file.close()

        lstmotifs = list(motifs.keys())
        variablemotif = StringVar()
        optmotif = OptionMenu(tk, variablemotif, *lstmotifs)
        variablemotif.set(lstmotifs[0])
        optmotif.pack()
        Xmotif = Scale(tk, orient="horizontal", from_=0, to=(taille-1), resolution=1, length=250, label="x (coin supérieur gauche du motif):")
        Xmotif.set(0)
        Xmotif.pack()
        Ymotif = Scale(tk, orient="horizontal", from_=0, to=(taille-1), resolution=1, length=250, label="y (coin supérieur gauche du motif):")
        Ymotif.set(0)
```

```
        optmotif = OptionMenu(tk, variablemotif, *lstmotifs)
        variablemotif.set(lstmotifs[0])
        optmotif.pack()
        Xmotif = Scale(tk, orient="horizontal", from_=0, to=(taille-1), resolution=1, length=250, label="x (coin supérieur gauche du motif):")
        Xmotif.set(0)
        Xmotif.pack()
        Ymotif = Scale(tk, orient="horizontal", from_=0, to=(taille-1), resolution=1, length=250, label="y (coin supérieur gauche du motif):")
        Ymotif.set(0)
        Ymotif.pack()
        matriceinit = np.zeros((taille, taille))
        btnAjoutmotif = Button(tk, text="ajouter", command=lambda:AjoutMotif(ajout=True, etats=etats))
        btnAjoutmotif.pack()
        btnFinmotif = Button(tk, text="configuration terminée", command=lambda:AjoutMotif(ajout=False, lancer=lancer_, etats=etats))
        btnFinmotif.pack()


def AjoutMotif(ajout, etats=2, lancer=None, lenia=False):
    global matriceinit
    if ajout:
        #on ajoute le motif à la matriceinit
        x = Xmotif.get()
        y = Ymotif.get()
        motif = motifs[variablemotif.get()]
        for i in range(len(motif)):
            for j in range(len(motif[0])):
                matriceinit[(y+i)%taille][(x+j)%taille] = motif[i][j]
        if etats == -1:#Lenia
            show_mat3(matriceinit)
        else:
            show_mat2(matriceinit, etats)
        Xmotif.set(0)
        Ymotif.set(0)
        variablemotif.set(lstmotifs[0])
    else:
        #suppression des derniers boutons,on lance la fonction lancement de main.py avec tout les paramètres
        optmotif.destroy()
        Xmotif.destroy()
        Ymotif.destroy()
        btnAjoutmotif.destroy()
        btnFinmotif.destroy()
        lancer(matriceinit)


def start_Primordia(taille, variante_id):
    global btnChoix0, btnChoix1
    etats = optPrimordia.get()
    K = KPrimordia.get()
    L = LPrimordia.get()
    M = MPrimordia.get()
    N = NPrimordia.get()
    params = [K, L, M, N]
    optPrimordia.destroy()
    btnPrimordia.destroy()
    KPrimordia.destroy()
    LPrimordia.destroy()
    MPrimordia.destroy()
    NPrimordia.destroy()
    c = (couleur.get() == "selected")
    col.destroy()
    if "Primordia.json" in os.listdir("."):
        btnChoix0 = Button(tk, text="position de départ aléatoire", command=lambda:(deleteChoixPos(),callback(taille, variante_id, states=etats, colour=c, primord=params)))
        btnChoix0.pack()
        btnChoix1 = Button(tk, text="position de départ indiquée manuellement", command=lambda:start_ajoutMotifs(taille, variante_id, aleatoire=False, etats=etats, lancer_=lambda A:callback(taille, variante_id, states=etats, colour=c, primord=params, matrice=A)))
        btnChoix1.pack()

    else:
        callback(taille, variante_id, states=etats, colour=c)


def start_Ltl(taille, variante_id):
    global btnChoix0, btnChoix1
    etats = optLtl.get()
    r = optLtl0.get()
    b1 = optLtl1.get()
    b2 = optLtl2.get()
    s1 = optLtl3.get()
    s2 = optLtl4.get()
    optLtl.destroy()
    optLtl0.destroy()
    optLtl1.destroy()
    optLtl2.destroy()
    optLtl3.destroy()
    optLtl4.destroy()
    btnLtl.destroy()
    c = couleur.get() == "selected"
    col.destroy()
    if "Primordia.json" in os.listdir("."):
        btnChoix0 = Button(tk, text="position de départ aléatoire", command=lambda:(deleteChoixPos(),callback(taille, variante_id, R=r, states=etats, ltl_survival=[min(s1, s2), max(s1, s2)], ltl_birth=[min(b1, b2), max(b1, b2)], colour=c)))
        btnChoix0.pack()
        btnChoix1 = Button(tk, text="position de départ indiquée manuellement", command=lambda:start_ajoutMotifs(taille, variante_id, aleatoire=False, etats=etats, lancer_=lambda A:callback(taille, variante_id, R=r, states=etats, ltl_survival=[min(s1, s2), max(s1, s2)], ltl_birth=[min(b1, b2), max(b1, b2)], colour=c, matrice=A)))
        btnChoix1.pack()

    else:
        callback(taille, variante_id, R=r, states=etats, ltl_survival=[min(s1, s2), max(s1, s2)], ltl_birth=[min(b1, b2), max(b1, b2)], colour=c)


def start_Lenia(taille, variante_id, noyaux):
    global btnChoix0, btnChoix1, kernelR, kernelM, kernelS, kernelBtn
    fonction = variableLenia.get()
    noy = variableLenia2.get()
    kernel = np.array(noyaux[noy])
    #print(kernel, type(kernel[0]))
    optLenia0.destroy()
    t = optLenia.get()
    optLenia.destroy()
    optLenia1.destroy()
    btnLenia.destroy()
    c = couleur.get() == "selected"
    col.destroy()
    x = np.linspace(0, 1, 400)
    f = lambda x: eval(fonction)
    f_vect = np.vectorize(f)#on rend la fonction compatible avec les np.arrays
    y = f_vect(x)

    plt.plot(x, y)
    plt.title("f(x) = "+fonction)
    plt.xlabel("x")
    plt.ylabel("f(x)")
    plt.show(block=False)
    smooth_k = []
    if noy == "smooth kernel":
        kernelR = Scale(tk, orient="horizontal", from_=1, to=20, resolution=1, length=250, label="R:")
        kernelM = Scale(tk, orient="horizontal", from_=0, to=1, resolution=0.01, length=250, label="m:")
        kernelS = Scale(tk, orient="horizontal", from_=0.01, to=1, resolution=0.01, length=250, label="s:")
        kernelR.set(13)
        kernelM.set(0.5)
        kernelS.set(0.15)
        kernelR.pack()
        kernelM.pack()
        kernelS.pack()
        kernelBtn = Button(tk, text="confirmer paramètres gaussienne f(x)=0.5*exp(-((x-m)/s)^2) pour noyau de rayon R", command=lambda:select_start_pos(taille, variante_id, t, c, fonction, kernel))
        kernelBtn.pack()
    else:
        select_start_pos(taille, variante_id, t, c, fonction, kernel)


def select_start_pos(taille, variante_id, t, c, fonction, kernel):
    global btnChoix0, btnChoix1
    smooth_ker = get_smooth_k()
    print(smooth_ker)
    if "Lenia.json" in os.listdir("."):
        btnChoix0 = Button(tk, text="position de départ aléatoire", command=lambda:(deleteChoixPos(),callback(taille, variante_id, T=t, colour=c, growth=fonction, noyau=kernel, smooth_=smooth_ker)))
        btnChoix0.pack()
        btnChoix1 = Button(tk, text="position de départ indiquée manuellement", command=lambda:start_ajoutMotifs(taille, variante_id, aleatoire=False, etats=-1, lancer_=lambda A:callback(taille, variante_id, T=t, colour=c, growth=fonction, noyau=kernel, matrice=A, smooth_=smooth_ker)))
        btnChoix1.pack()

    else:

        callback(taille, variante_id, T=t, colour=c, growth=fonction, noyau=kernel, smooth_=smooth_ker)


def get_smooth_k():
    print("appel")
    try:
        print("début")
        R = kernelR.get()
        m = kernelM.get()
        s = kernelS.get()
        print("fin-1")
        kernelR.destroy()
        kernelM.destroy()
        kernelS.destroy()
        kernelBtn.destroy()
        print("fin0")
        smooth_k = [R, m, s]
        print("done")
    except Exception as e:
        print(str(e))
        smooth_k = []
    print("sm", smooth_k)
    return smooth_k

def start_ui(callback0):
    global echelleTaille, btn1, callback
    callback = callback0
    echelleTaille = Scale(tk, orient="horizontal", from_=1, to=150, resolution=1, length=250, label="taille de la grille")
    echelleTaille.set(64)
    echelleTaille.pack()
    btn1 = Button(tk, text="confirmer la taille", command=select_taille)
    btn1.pack()
    mainloop()


def click(event):
    taille = var_["taille"]
    if taille != -1:
        unit = 250//taille
        print("x:",event.x*taille//500, "y:", event.y*taille//500)

tk.bind("<Button-3>", click)

def f(x):
    M=3
    N=2
    K=10
    L=6
    if K<=x<=K+L:
        return 1
    elif K-M<=x<=K+L+N:
        return 0
    else:
        return -1
if __name__ == "__main__":
    n=50
    A=[[i for i in range(n+1)] for j in range(n+1)]
    show_mat2(A, states=50)

```

Annexe

Jeudelavie.json

```
{"une cellule vivante": [[1]],
 "glider": [[0, 1, 0], [0, 0, 1], [1, 1, 1]],
 "nouveau motif (0, 0), (0, 1)": [[0.0], [0.0]]}
```

Annexe

Primordia.json


```
{"une cellule de valeur 1": [[1]],
 "une cellule de valeur 5": [[5]],
 "nouveau motif (2, 0), (4, 3)": [[1, 0, 0], [0, 0,
1], [0, 0, 1], [1, 0, 0]],
 "nouveau motif (0, 0), (3, 3)": [[0, 1, 0, 0], [1, 0,
2, 1], [0, 2, 0, 0], [0, 1, 0, 0]],
 "une cellule de valeur 2": [[2]],
 "une cellule de valeur 3": [[3]],
 "une cellule de valeur 4": [[4]],
 "nouveau motif (0, 0), (4, 4)": [[0, 2, 0, 0, 0], [0,
0, 2, 0, 0], [1, 2, 2, 0, 0], [0, 1, 0, 0, 0], [0, 0,
0, 0, 0]],
"nouveau motif (0, 0), (7, 7)": [[0, 0, 0, 0, 0, 0, 0,
0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0,
0], [0, 0, 1, 0, 1, 0, 0, 0], [0, 0, 1, 2, 4, 1, 0,
0], [0, 0, 0, 0, 4, 1, 0, 0], [0, 0, 0, 0, 0, 0, 0,
0], [0, 0, 0, 0, 0, 0, 0, 0]]}
```

Annexe

Ltl.json


```
{"une cellule de valeur 1": [[1]],
 "nouveau motif (0, 0), (2, 0)": [[1, 0, 0]],
 "nouveau motif (32, 8), (55, 28)": [[0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0,
0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0], [0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 1, 1, 1, 1, 1,
1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0], [1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [1, 1, 1, 1,
0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0], [1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0], [1, 1, 1, 1, 1, 0, 0, 0,
1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0,
1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0], [0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 1, 1,
1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0], [0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]],
"Bosco":[[0,0,0,0,1,0,0,0,0,0], [0,0,1,1,1,1,1,0,0,0],
[0,1,1,0,0,1,1,1,1,0], [1,1,0,0,0,1,1,1,1,1],
[1,0,0,0,0,0,1,1,1,1], [1,1,0,0,0,0,0,1,1,1,1],
[1,1,1,0,1,1,1,1,1,1], [0,1,1,1,1,1,1,1,1,0],
[0,1,1,1,1,1,1,1,0,0], [0,0,1,1,1,1,1,0,0,0],
[0,0,0,1,1,1,0,0,0,0]]}
```

Annexe

Lenia.json


```
{"une cellule vivante": [[1]], "orbium": [[0, 0, 0, 0,
0, 0, 0.1, 0.14, 0.1, 0, 0, 0.03, 0.03, 0, 0, 0.3, 0,
0, 0, 0], [0, 0, 0, 0, 0, 0.08, 0.24, 0.3, 0.3, 0.18,
0.14, 0.15, 0.16, 0.15, 0.09, 0.2, 0, 0, 0, 0], [0, 0,
0, 0, 0, 0.15, 0.34, 0.44, 0.46, 0.38, 0.18, 0.14,
0.11, 0.13, 0.19, 0.18, 0.45, 0, 0, 0], [0, 0, 0, 0,
0.06, 0.13, 0.39, 0.5, 0.5, 0.37, 0.06, 0, 0, 0, 0.02,
0.16, 0.68, 0, 0, 0], [0, 0, 0, 0.11, 0.17, 0.17,
0.33, 0.4, 0.38, 0.28, 0.14, 0, 0, 0, 0, 0, 0.18,
0.42, 0, 0], [0, 0, 0.09, 0.18, 0.13, 0.06, 0.08,
0.26, 0.32, 0.32, 0.27, 0, 0, 0, 0, 0, 0, 0.82, 0, 0],
[0.27, 0, 0.16, 0.12, 0, 0, 0, 0.25, 0.38, 0.44, 0.45,
0.34, 0, 0, 0, 0, 0, 0.22, 0.17, 0], [0, 0.07, 0.2,
0.02, 0, 0, 0, 0.31, 0.48, 0.57, 0.6, 0.57, 0, 0, 0,
0, 0, 0, 0.49, 0], [0, 0.59, 0.19, 0, 0, 0, 0, 0, 0.2,
0.57, 0.69, 0.76, 0.76, 0.49, 0, 0, 0, 0, 0, 0.36, 0],
[0, 0.58, 0.19, 0, 0, 0, 0, 0, 0.67, 0.83, 0.9, 0.92,
0.87, 0.12, 0, 0, 0, 0, 0.22, 0.07], [0, 0, 0.46, 0,
0, 0, 0, 0, 0.7, 0.93, 1, 1, 1, 0.61, 0, 0, 0, 0,
0.18, 0.11], [0, 0, 0.82, 0, 0, 0, 0, 0, 0.47, 1, 1,
0.98, 1, 0.96, 0.27, 0, 0, 0, 0.19, 0.1], [0, 0, 0.46,
0, 0, 0, 0, 0, 0.25, 1, 1, 0.84, 0.92, 0.97, 0.54,
0.14, 0.04, 0.1, 0.21, 0.05], [0, 0, 0, 0.4, 0, 0, 0,
0, 0.09, 0.8, 1, 0.82, 0.8, 0.85, 0.63, 0.31, 0.18,
0.19, 0.2, 0.01], [0, 0, 0, 0.36, 0.1, 0, 0, 0, 0.05,
0.54, 0.86, 0.79, 0.74, 0.72, 0.6, 0.39, 0.28, 0.24,
0.13, 0], [0, 0, 0, 0.01, 0.3, 0.07, 0, 0, 0.08, 0.36,
0.64, 0.7, 0.64, 0.6, 0.51, 0.39, 0.29, 0.19, 0.04,
0], [0, 0, 0, 0, 0.1, 0.24, 0.14, 0.1, 0.15, 0.29,
0.45, 0.53, 0.52, 0.46, 0.4, 0.31, 0.21, 0.08, 0, 0],
[0, 0, 0, 0, 0, 0.08, 0.21, 0.21, 0.22, 0.29, 0.36,
0.39, 0.37, 0.33, 0.26, 0.18, 0.09, 0, 0, 0], [0, 0,
0, 0, 0, 0, 0.03, 0.13, 0.19, 0.22, 0.24, 0.24, 0.23,
0.18, 0.13, 0.05, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0,
0, 0.02, 0.06, 0.08, 0.09, 0.07, 0.05, 0.01, 0, 0, 0,
0, 0]], "nouveau motif (27, 57), (35, 63)": [[0.2,
0.7, 0.9, 0.8999999999999999, 0.14000000000000007,
0.0, 0.3400000000000001, 0.7, 0.0], [0.2, 0.7, 0.9,
0.10000000000000003, 0.0, 0.0, 0.0, 0.5, 0.0], [0.2,
0.75, 0.8999999999999999, 0.10000000000000003, 0.0,
0.0, 0.0, 0.5, 0.0], [0.3500000000000001, 0.9, 0.76,
0.10000000000000003, 0.0, 0.0, 0.0, 0.5, 0.0], [0.2,
0.2, 0.8999999999999999, 0.7100000000000001,
0.10000000000000003, 0.10000000000000003,
0.10000000000000003, 0.2, 0.0], [0.0, 0.2, 0.2,
0.30000000000000004, 0.78, 0.7, 0.4, 0.0, 0.0], [0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]], "nouveau
motif (10, 8), (22, 21)": [[0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0,
0.0, 0.0, 0.10000000000000014, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.4, 0.9, 1.0,
1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0,
0.10000000000000003, 0.2, 0.0, 0.0, 0.4, 1.0, 1.0,
0.4, 0.0, 0.0, 0.0], [0.0, 0.10000000000000003, 0.2,
0.0, 0.0, 0.0, 0.0, 0.30000000000000004, 0.5, 0.7,
0.30000000000000004, 0.0, 0.0], [0.0, 0.5, 0.1, 0.0,
0.0, 0.0, 0.0, 0.30000000000000004, 0.5, 0.7, 0.4,
0.0, 0.0], [0.10000000000000014, 0.5000000000000001,
0.1, 0.0, 0.0, 0.0, 0.0, 0.20000000000000004, 0.4,
0.7, 0.4, 0.0, 0.0], [0.0, 0.20000000000000015,
0.9999999999999999, 0.0, 0.0, 0.0,
2.7755575615628914e-17, 0.20000000000000004, 0.0, 0.6,
0.30000000000000004, 0.0, 0.0], [0.0, 0.0,
0.6000000000000001, 0.7, 0.30000000000000004,
0.20000000000000004, 0.10000000000000003, 0.5, 0.5,
0.30000000000000004, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0,
0.8, 1.0, 0.9999999999999999, 0.6, 0.4, 0.0, 0.0, 0.0,
0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.4, 0.6,
0.30000000000000004, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]],
"nouveau motif (6, 46), (17, 56)": [[0.0, 0.0, 0.0,
0.0, 0.2, 0.2, 0.2, 0.2, 0.0, 0.0, 0.0, 0.0], [0.0,
0.0, 0.2, 0.4, 0.4, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0,
0.0], [0.0, 0.2, 0.4, 0.6000000000000001, 0.8,
0.6000000000000001, 0.20000000000000007,
5.551115123125783e-17, 0.0, 0.4, 0.0, 0.0], [0.0, 0.2,
0.4, 0.6000000000000001, 0.8, 0.0, 0.0, 0.0, 0.0, 0.4,
0.0, 0.0], [0.0, 0.2, 0.4, 0.6000000000000001, 0.8,
0.0, 0.0, 0.0, 0.0, 0.4, 0.0, 0.0], [0.0, 0.2, 0.4,
0.6000000000000001, 0.8, 0.0, 0.0, 0.0, 0.0,
0.4000000000000001, 0.0, 0.0], [0.0, 0.2, 0.4,
0.6000000000000001, 0.6000000000000001,
0.6000000000000001, 0.0, 0.0, 0.0, 0.4, 0.0, 0.0],
[0.0, 0.0, 0.0, 0.2, 0.4, 0.6000000000000001, 1.0,
1.0, 1.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0]]}
```

Annexe

kernel.json


```
{"R = 5 (11x11), centre non inclus":
[[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
[1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1],
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]
,
"R = 1, noyau utilisé par d'autres automates":
[[1, 1, 1],
[1, 0, 1],
[1, 1, 1]],
"R = 5, cercle":
[[0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0],
[0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0],
[0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
[0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
[1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1],
[1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1],
[1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1],
[0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
[0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
[0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0],
[0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0]],
"R = 13":
[[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
[1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1],
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]
}
```

Annexe

fonctions.txt

```
1 if 0.12<=x<=0.15 else -1
math.sin(2*math.pi*x)
2*math.sin(math.pi*x) - 1
2*math.exp(-400*(x-1/2)**2) - 1
x - 1/2
2*math.exp(-((x-0.15)/0.015)**2 / 2) - 1
2*math.exp(-((x-0.16)/0.015)**2 / 2) - 1
2*math.exp(-((x-0.26)/0.015)**2 / 2) - 1
```