Lycée Lakanal

Simulation d'épidémies à l'aide d'automates cellulaires

Arthur Lacoin - Timothé Rios

Table des matières

L'étude de la propagation des maladies est une science qui remonte à la Grèce Antique, lorsque Hippocrate, au quatrième siècle avant Jésus-Christ, créa les termes d'endémie et d'épidémie afin de qualifier respectivement des maladies liées à des endroits et à des périodes donnés. Mais cette science ne se développa vraiment qu'en 1854, quand John Snow, un médecin britannique, étudia la propagation de l'épidémie de choléra dans les quartiers de Londres. Ce sont là les prémices de l'épidémiologie, l'étude du transport des infections. De nos jours, les méthodes ont évolué et la méthode différentielle, que l'on présentera plus loin, est la plus fréquemment utilisée. Nous nous sommes intéressés ici à la simulation par automates cellulaires.

1 Première version

1.1 Présentation

1.1.1 Définition formelle

Un automate cellulaire est une matrice de cellules, chacune ayant un état appartenant à un ensemble prédéfini. L'état de chaque cellule peut varier au cours du temps suivant une fonction de transfert : l'état de la matrice à l'instant $t+1$ dépend ainsi de son état à l'instant t. L'automate doit donc posséder un état initial, c'est-à-dire la matrice des états initiaux des cellules. Même si son principe de base est simple, l'automate cellulaire se complexifie grandement lorsque la taille de la grille augmente, ce qui en fait un modèle couramment utlisé dans l'étude des systèmes complexes.

1.1.2 Historique

Les automates cellulaires sont plutôt récents. Ils ont été mis au point par John Von Neumann dans son étude des systèmes auto-réplicatifs dans les années 1940. Cette notion a été grandement popularisée par le "jeu de la vie" de John Conway, un automate cellulaire en deux dimensions paru dans les années 1970. À ce jour, les automates cellulaires ont de nombreuses applications dans divers domaines :

- diffusion d'un gaz en s'appuyant sur les équations de Navier-Stokes
- simulation des feux de forêts
- simulation du trafic routier

1.1.3 Notre automate cellulaire

Notre automate cellulaire est un automate en deux dimensions qui vise à simuler la propagation d'une épidémie. Les cellules ont donc un état parmi les quatre suivants : Sain, Malade, Guéri, Mort. L'état Sain est l'état initial que chaque cellule sauf une possède à la première étape de la simulation. Lorsqu'une cellule est saine, elle peut tomber malade avec une probabilité p_1 s'il y a des malades dans son entourage. L'état Malade est au départ donné à une seule cellule, le "patient zéro". Une cellule malade peut soit mourir avec une probabilité p_2 soit guérir avec une probabilité p_3. Les cellules mortes ou guéries restent dans cet état indéfiniment, l'hypothèse prise étant qu'une cellule guérie est immunisée et ne peut plus attraper la maladie. Enfin, la simulation s'arrête lorsqu'il n'y a plus de cellule malade, car l'état de l'automate ne peut alors plus varier.

Calcul des probabilités

Les probabilités d'infection, de soin et de mort que nous avons utilisé ont été calculées à partir de paramètres empiriques tirés d'études réelles. On utilisera en particulier τ, le temps moyen qu'un individu reste infecté, et R_0, le nombre moyen d'individus qu'un malade contamine durant son temps de contamination.

On cherche d'abord à déterminer la probabilité d'infection. On se place donc dans (Ω, P) un espace probabilisé fini. On étudie un individu malade entouré de n voisins sains. On considère que l'infection d'un individu sain est indépendante des autres.
On établira d'abord un résultat préliminaire.
Soit X, Y deux variables aléatoires réelles sur Ω et à valeur dans $[\![0, n]\!], n \in \mathbb{N}$, telles que $\forall k \in [\![0, n]\!], X$ sachant que $Y = k$ suit une loi binomiale de paramètres $(n-k, p), p \in]0, 1[$.
Alors $E(X) = (n - E(Y))p$.

On a donc $\forall k \in [\![0,n]\!], E_{\{Y=k\}}(X) = (n-k)p$. Par formule de l'espérance totale,

$$\begin{aligned} E(X) &= \sum_{k=0}^{n} P(Y=k) E_{\{Y=k\}}(X) \\ &= \sum_{k=0}^{n} P(Y=k)(n-k)p \\ &= np \sum_{k=0}^{n} P(Y=k) - p \sum_{k=0}^{n} kP(Y=k) \\ &= np - pE(Y) \\ &= p(n - E(Y)) \end{aligned}$$

Soit $k \in [\![1,\tau]\!], X_k$ est la variable aléatoire donnant le nombre d'individus qui ont été infectés au jour k.
$X_1 \hookrightarrow B(n,p)$ car on infecte les individus indépendamment.
Au jour k il reste

$$n - \sum_{i=0}^{k-1} X_i$$

individus sains.

Montrons par récurrence que $\forall k \in [\![1,\tau]\!], E(X_k) = n(1-p)^{k-1}p$

$\forall k \in [\![1,\tau]\!]$, notons $P(k) =$ "$\forall i \in [\![1,k]\!], E(X_i) = n(1-p)^{i-1}p$".

$P(1)$ est vraie car $X_1 \hookrightarrow B(n,p)$.

Ou bien $\tau = 1$ et c'est fini.

Ou bien $\tau \geq 2$ et on suppose que $P(k)$ est vraie pour $k \in [\![1,\tau-1]\!]$. Montrons que $P(k+1)$ est vraie.

$\forall i \in [\![1,k]\!], E(X_i) = n(1-p)^{i-1}p$ par $P(k)$. Il reste à calculer $E(X_{k+1})$.

$$\begin{aligned} E(X_{k+1}) &= p(n - \sum_{i=1}^{n} E(X_i)) \\ &= p(n - \sum_{i=1}^{n} n(1-p)^{i-1}p \\ &= np(1-p)^k \end{aligned}$$

On conclut par récurrence multiple finie.
Armés de ces résultats, nous pouvons quasiment calculer p, car on a le nombre moyen d'infectés par un malade au cours de sa période d'infection, R_0. On a donc

$$\begin{aligned} R_0 &= \sum_{k=1}^{\tau} E(X_k) \\ &= \sum_{k=1}^{\tau} np(1-p)^{k-1} \\ &= np\frac{1-(1-p)^\tau}{p} \\ &= n(1-(1-p)^\tau) \end{aligned}$$

Ce qui nous amène par de brefs calculs à la conclusion :

$$p = 1 - \sqrt[\tau]{\frac{n-R_0}{n}}$$

.

On cherchera désormais à calculer la probabilité de ne plus être malade, c'est à dire ou bien de guérir ou bien de mourir, c'est à dire $p_{\text{mort}} + p_{\text{guérison}} = q$. Si l'on considère un individu qui est infecté au jour 0, et que

l'on considère la variable aléatoire X donnant le jour de guérison(sachant qu'on ne peut guérir le jour de son infection), $\forall k \in \mathbb{N}^*, P(X = k) = (1-q)^{k-1}q$.
X suit donc une loi géométrique et son espérance, c'est à dire le nombre de jour où l'individu restera malade(τ) vaut par propriété $\frac{1}{q}$.
On a donc

$$q = \frac{1}{\tau}$$

Enfin pour calculer la probabilité de mourir, il suffit de faire mq, où m est la mortalité de la maladie, et la probabilité de guérir vaut $(1-m)q$.
Les résultats réels sont récapitulés dans le tableau suivant :

Maladie	$p_{infection}$	p_{mort}	$p_{guerison}$
Ebola	0.0201	0.0818	0.00909

1.2 Résultats

Voici quelques étapes d'une simulation de notre automate. Le foyer originel -la première cellule infectée- est placée au centre de la grille à l'étape 1. Sur ces images,représentant la grille de l'automate, les cellules saines sont représentées en blanc, celles malades sont rouges, celle guéries sont vertes et celles mortes sont noires.

FIGURE 1 – Étape 36

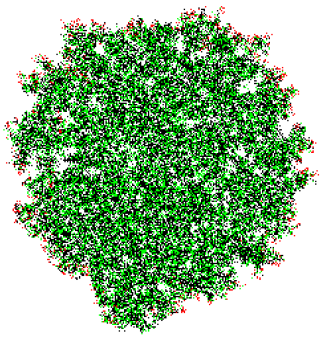


FIGURE 2 – Étape 195

Dans la simulation présentée ci-dessous, les probabilités d'infection, de guérison et de mort ont été choisies arbitrairement afin de procurer un visuel clair du comportement de l'automate. Voici encore d'autres exemples de cas possibles de comportements de maladie, dont l'évolution est cette fois-ci représentée sur un graphique donnant le nombre de personnes saines, malades, guéries et mortes au cours des étapes de la simulation.

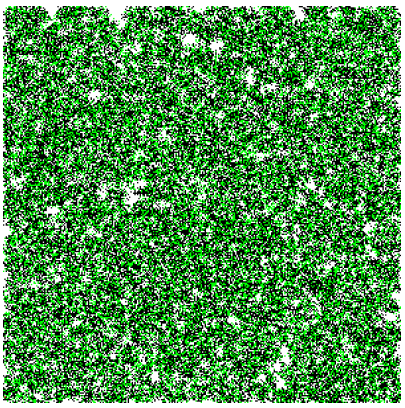
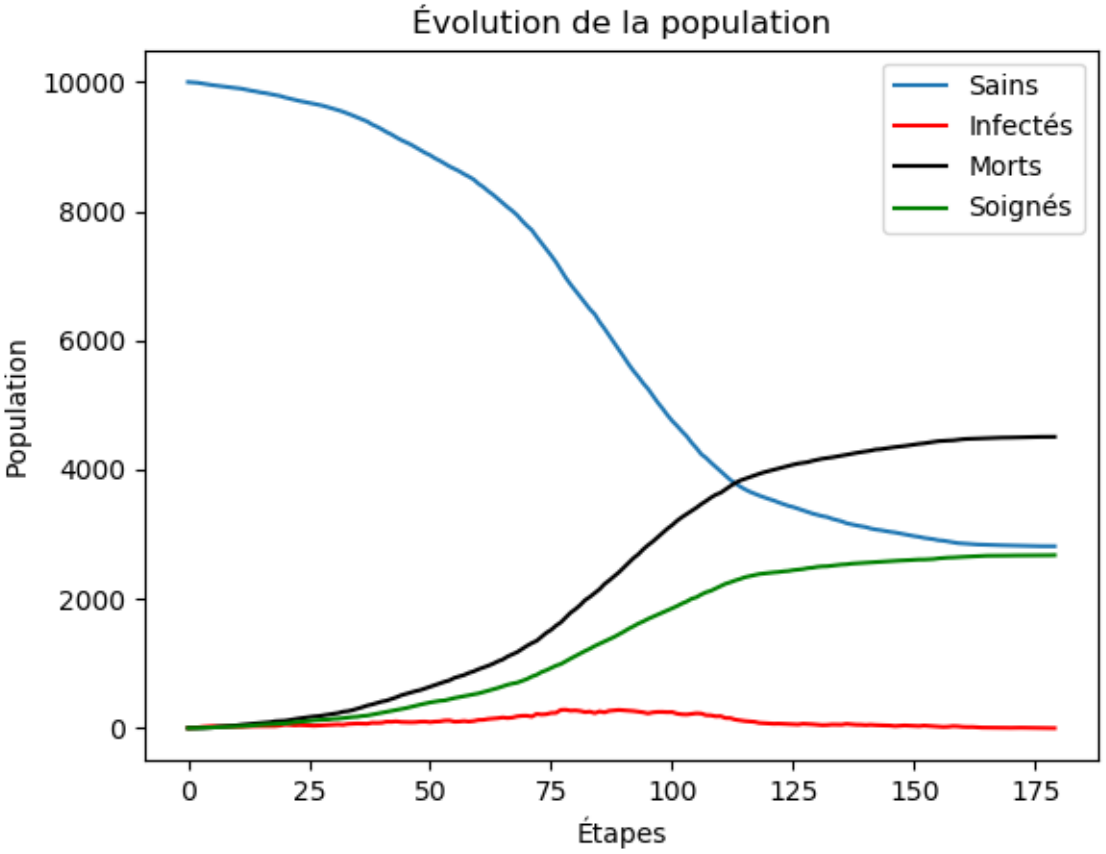
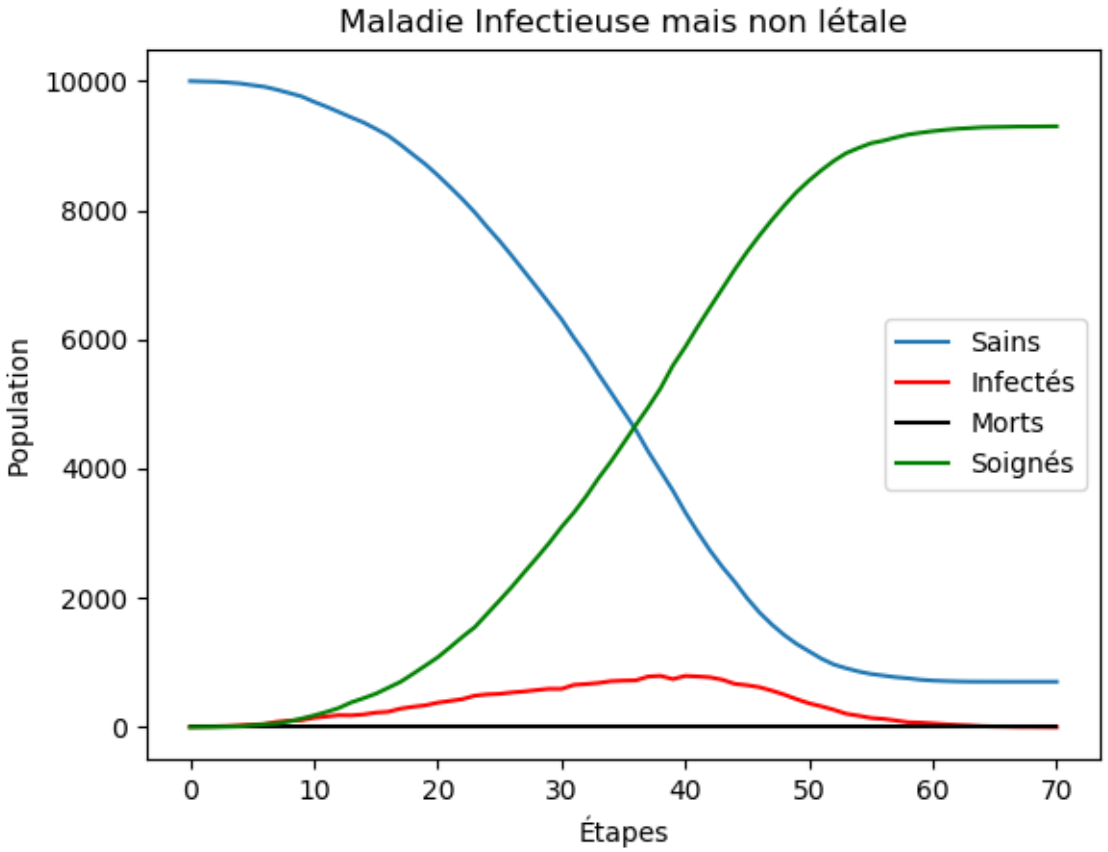


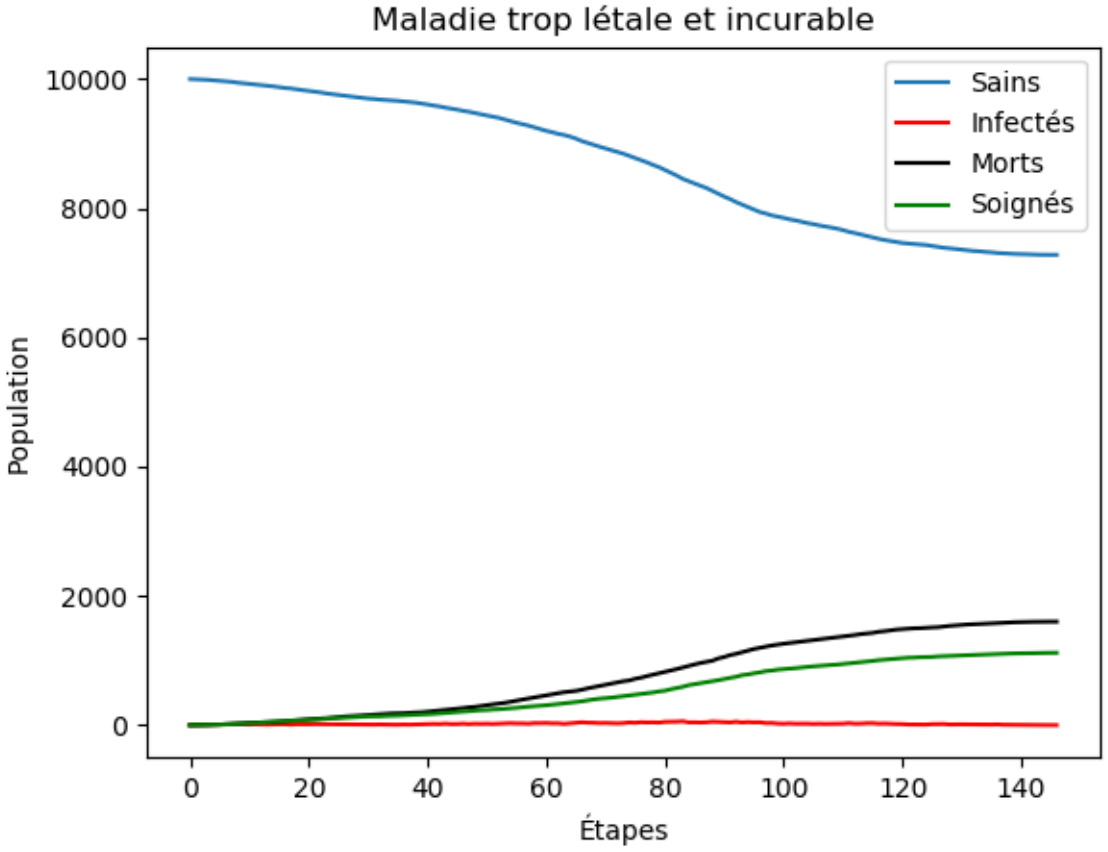
FIGURE 3 – Étape 412



Dans ce cas, p_1 valait 0.2, p_2 valait 0.25, p_3 valait 0.2 et la maladie pouvait se transmettre dans un rayon de deux cellules autour de l'infectée.



Ici, nous pouvons voir l'exemple d'une maladie non létale, se rapprochant d'un rhume tout en étant bien plus infectieuse.



Dans cet exemple, la maladie est trop létale pour pouvoir se propager longtemps : les infectés meurent sans avoir eu le temps de contaminer un de leurs voisins.

1.3 Limites

Cette version de l'automate cellulaire fournit déjà des résultats intéressants. Cependant, le modèle reste trop simple pour pouvoir modéliser la réalité d'une épidémie. En effet, il est ici supposé que les individus de la population étudiée restent sur place tout en étant étroitement collés les uns aux autres. Il apparaît ainsi qu'un grand nombre de paramètres a été négligé : il est ici supposé que la population est uniformément répartie, alors qu'en réalité cette densité dépend d'un nombre important de facteurs, comme le type de terrain, les constructions humaines (villes par exemple) ou les conditions météorologiques. De plus, nous supposons aussi que l'épidémie se déroule en un temps assez court pour que les décès naturels et les naissances soient négligeables, tout comme nous négligeons les mesures que pourraient prendre certains gouvernements en cas de pandémies majeures.

2 Deuxième version

2.1 Présentation

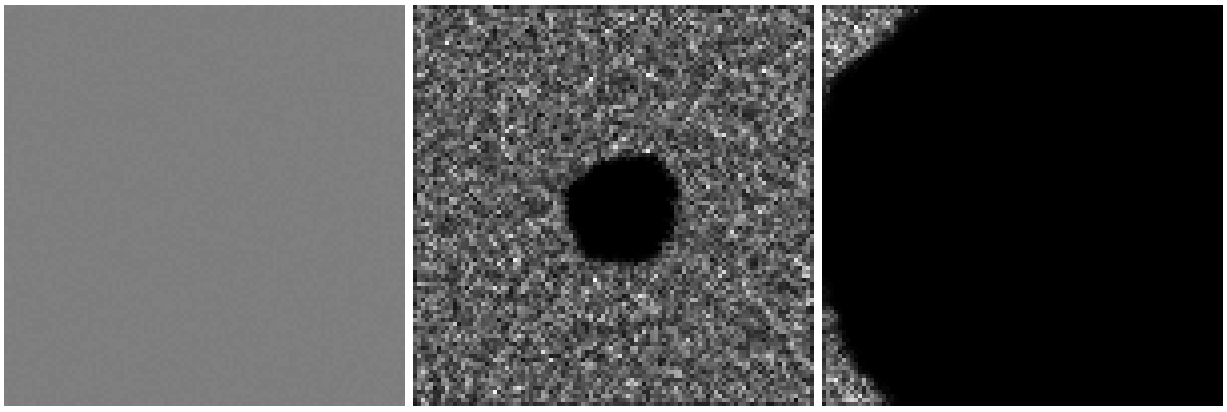
Ce deuxième automate prend en compte plus de paramètre :

- nous avons cette fois considéré les cellules comme des zones géographiques plutôt que des individus, ayant une population ainsi qu'une répartition Sains-Malades-Morts-Guéris propre.
- nous avons pris en compte les densités de population, celles-ci dépendant du 'type' de case, à savoir Ville, Campagne, Montagnes ou Route.
- nous avons pris en compte les mouvements de population, en fonction de deux paramètres : la probabilité qu'une partie de population quitte une case, et la probabilité qu'une case a d'attirer la population des cases adjacentes.

Ces ajouts de paramètres permettent une simulation plus fine et plus proche de la réalité. Cela permet d'utiliser de bien meilleure façon les automates cellulaires, qui ont pour avantage de prendre en compte les paramètres géographiques de la simulation.

2.2 Résultats

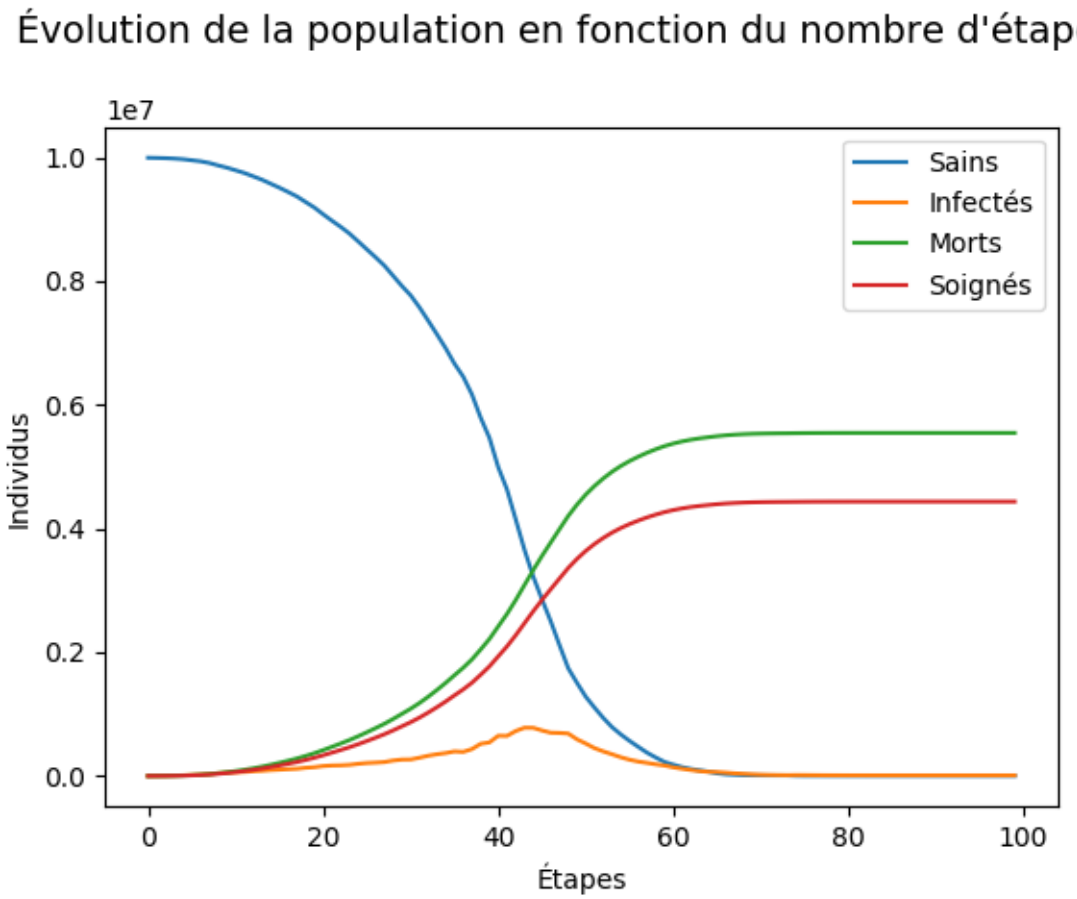
Les résultats graphiques de cet automate permettent d'afficher la densité de population des quatre états au sein de toutes les cellules de la grille. Bien sûr, il n'est maintenant plus possible d'afficher tous les paramètres de la grille sur une seule image. Voici par exemple la répartition des individus sains sur une grille



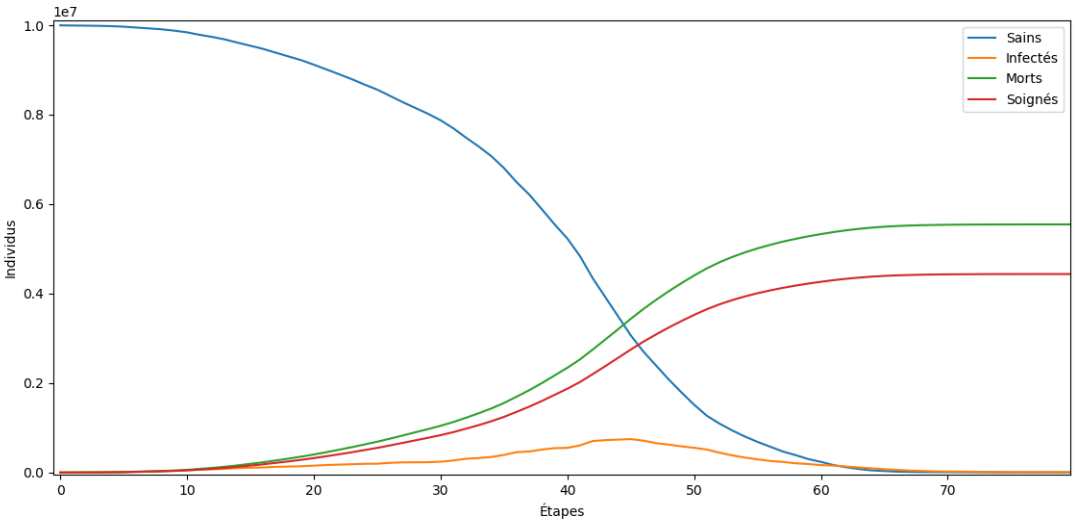
Ici, plus une cellule contient de personnes saines, plus le pixel la représentant sur l'image est blanc. Les cellules sont donc toutes légèrement grisées à la première étape, non pas parce qu'elles contiennent des individus malades, morts ou guéris, mais parce que le programme étalonne automatiquement les couleurs en fonction de la taille de la grille et de sa population afin de pouvoir intensifier la couleur si la population saine d'une cellule augmente. Bien sûr, cette simulation servant d'exemple pour présenter ce nouvel automate, ses cellules possèdent des probabilités de déplacement des populations bien plus élevées que dans la réalité. On remarque dans la dernière image que l'épidémie a complètement envahie le côté droit avant les coins de gauche. Cet état est bien dû à l'incertitude causée par les probabilités.

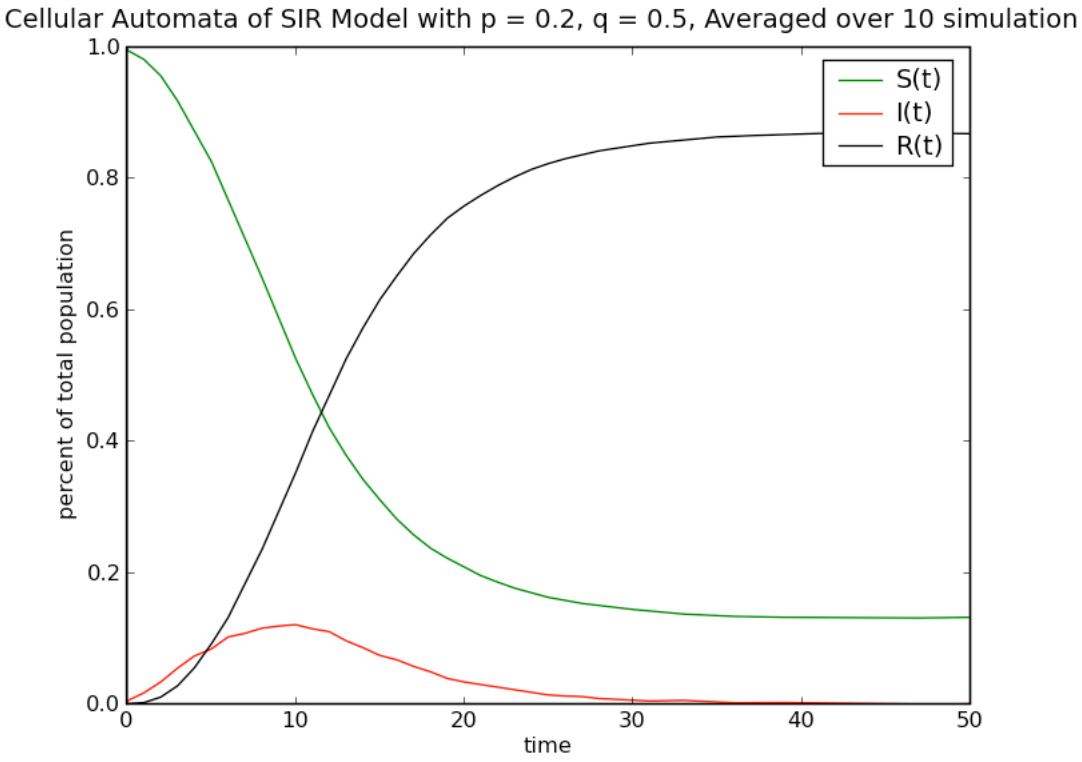
2.3 Comparaison

Ce nouvel automate nous permet donc une modélisation plus précise de l'évolution d'une maladie. Ainsi, nous avons pu comparer nos résultats,comme celui ci-dessous, avec ceux de précédentes études.

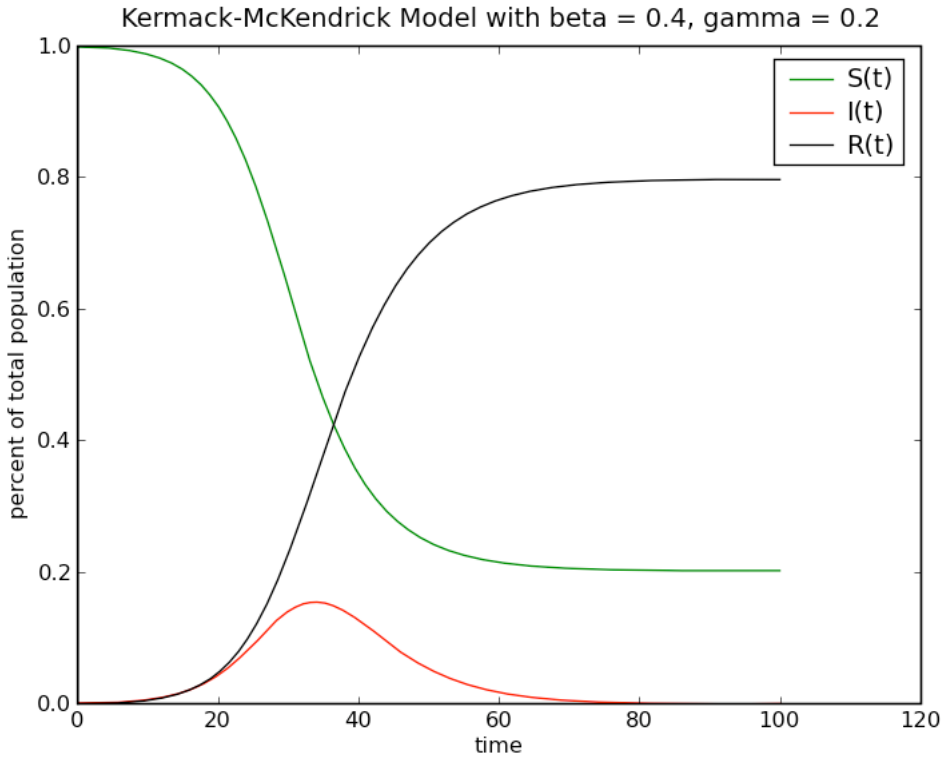


Évolution de la population en fonction du nombre d'étapes





Ci-dessus sont présentés les résultats d'une autre étude avec un automate cellulaire, il apparaît clairement que leurs courbes sont similaires aux nôtres.



De même, les résultats d'une autre étude utilisant la méthode différentielle concordent également à ceux que nous obtenons.

3 Méthode différentielle

On peut aussi simuler les épidémies par la méthode différentielle. Nous nous sommes basés sur le modèle SIR, c'est à dire le modèle sains, infectés, rétablis. Les équations sont les suivantes :

$$\frac{dS(t)}{dt} = -\beta I(t)S(t) \tag{1}$$
$$\frac{dI(t)}{dt} = \beta S(t)I(t) - \gamma I(t) - \delta I(t) \tag{2}$$
$$\frac{dM(t)}{dt} = \delta I(t) \tag{3}$$
$$\frac{dG(t)}{dt} = \gamma I(t) \tag{4}$$

Ces équations différentielles, une fois résolues, donnent des courbes d'évolution des populations que nous présenterons plus tard.
On remarque notamment que en sommant les quatre équations vues plus haut, on obtient 0, ce qui semble logique car la population au sens large(incluant les morts) ne varie pas.

4 Appendice

4.1 Code

```
import random
import numpy as np
import imageio
import sys
import datetime
import matplotlib.pyplot as plt
from PIL import Image
from PIL import ImageDraw
from scipy import ndimage
import math
import os
##


proba_infection = 0.2
proba_mort = 0.25
proba_soin = 0.1
rayon_infection = 2

max_etape = 399


def init_grid(size):
    GEO = [[3, 3, 3], [3, 90, 3], [3, 3, 3]]
    g = Grille(3, GEO, 10000000)
    return g

def compte(grille):
    compteur = [0, 0, 0, 0]
    for i in range(grille.taille):
        for j in range(len(grille.cellules[i])):
            for k in range(len(grille.cellules[i][j].repartition)):
                compteur[k] += grille.cellules[i][j].repartition[k]

    return compteur
```

```
def statistiques_finale(grille, virus, quantite):

    # Ces variables sont des listes de listes, qui contiennent la variation de
        la population
    # au cours du temps, pour chaque itération

    sains = []
    infectes = []
    morts = []
    soignes = []

    for i in range(quantite):
        nouvelle_grille = copy.deepcopy(grille)
        sain, infecte, mort, soigne = statistiques_2(nouvelle_grille, virus)
        sains.append(sain)
        infectes.append(infecte)
        morts.append(mort)
        soignes.append(soigne)

    return sains, infectes, morts, soignes


def statistiques_2(grille, virus):
    k = 0
    stats = []
    while compte(grille)[1] != 0:
        k += 1
        print(grille_pop_sum(grille))
        stats.append(grille.stats())
        grille.next(virus)
        if k > max_etape:
            break

    return zip(*stats)



    """
    fig = plt.figure()
    fig.suptitle('Evolution de la population en fonction du nombre d\'étapes',
        fontsize=14)

    ax = fig.add_subplot(111)
    fig.subplots_adjust(top=0.85)

    ax.set_xlabel('Etapes')
    ax.set_ylabel('Individus')

    etapes = np.array(range(len(sains)))

    sains_courbe, = plt.plot(etapes, sains, label='Sains')
    infectes_courbe, = plt.plot(etapes, infectes, label='Infectés')
    morts_courbe, = plt.plot(etapes, morts, label='Morts')
    soignes_courbe, = plt.plot(etapes, soignes, label='Soignés')

    plt.legend(handles=[sains_courbe, infectes_courbe, morts_courbe,
        soignes_courbe])

    plt.show()
"""
```

```
def init_im(taille):
    im = Image.new("RGB", (taille, taille), "white")
    return im


def transition_image_sains(grille, im, coeff):
    for i in range(grille.taille):
        for j in range(grille.taille):
            intensite = int(grille.cellules[i][j].repartition[0]*coeff)
            if intensite < 0:
                intensite = 0
            if intensite > 255:
                intensite = 255
            im.putpixel((i, j), (intensite, intensite, intensite))
    return im

def transition_image_malades(grille, im, coeff):
    for i in range(grille.taille):
        for j in range(grille.taille):
            intensite = int(grille.cellules[i][j].repartition[1]*coeff)
            if intensite < 0:
                intensite = 0
            if intensite > 255:
                intensite = 255
            im.putpixel((i, j), (intensite, 0, 0))
    return im

def transition_image_morts(grille, im, coeff):
    for i in range(grille.taille):
        for j in range(grille.taille):
            intensite = int(grille.cellules[i][j].repartition[2]*coeff)
            if intensite < 0:
                intensite = 0
            if intensite > 255:
                intensite = 255
            im.putpixel((i, j), (intensite, 0, intensite))
    return im

def transition_image_gueris(grille, im, coeff):
    for i in range(grille.taille):
        for j in range(grille.taille):
            intensite = int(grille.cellules[i][j].repartition[3]*coeff)
            if intensite < 0:
                intensite = 0
            if intensite > 255:
                intensite = 255
            im.putpixel((i, j), (0, intensite, 0))
    return im

def simulation_image_sains(grille, virus):
    im = init_im(grille.taille)
    gif = []
    k = 0
    moyenne = grille.population_initiale//grille.taille**2
    coeff = 127/moyenne
    print(moyenne,coeff)
    while compte(grille)[1] != 0:
        print(population(g)[0])
        k += 1

        im = transition_image_sains(grille, im, coeff)
```

```
        im.save( str(k)+'_sains'+'.png')
        gif.append( str(k)+'_sains'+'.png')
        grille.next(virus)
        if k > max_etape:
            break
        if compte(grille)[0] == 0:
            break
    return gif

def simulation_image_malades(grille , virus):
    im = init_im(grille.taille)
    gif = []
    k = 0
    moyenne = grille.population_initiale//grille.taille**2
    coeff = 127/moyenne
    print(moyenne,coeff)
    while compte(grille)[1] != 0:
        k += 1
        if sum(compte(grille)) > grille.population_initiale:
            print(compte(grille))
            print(k)
        im = transition_image_malades(grille , im, coeff)
        im.save( str(k)+'_malades'+'.png')
        gif.append( str(k)+'_malades'+'.png')
        grille.next(virus)
        if k > max_etape:
            break
    return gif

def simulation_image_morts(grille , virus):
    im = init_im(grille.taille)
    gif = []
    k = 0
    moyenne = grille.population_initiale//grille.taille**2
    coeff = 127/moyenne
    print(moyenne,coeff)
    while compte(grille)[1] != 0:
        k += 1

        im = transition_image_morts(grille , im, coeff)
        im.save( str(k)+'_morts'+'.png')
        gif.append( str(k)+'_morts'+'.png')
        grille.next(virus)
        if k > max_etape:
            break
    return gif

def simulation_image_gueris(grille , virus):
    im = init_im(grille.taille)
    gif = []
    k = 0
    moyenne = grille.population_initiale//grille.taille**2
    coeff = 127/moyenne
    print(moyenne,coeff)
    while compte(grille)[1] != 0:
        k += 1

        im = transition_image_gueris(grille , im, coeff)
        im.save( str(k)+'_gueris'+'.png')
        gif.append( str(k)+'_gueris'+'.png')
        grille.next(virus)
```

```
        if k > max_etape:
            break
    return gif


def create_gif(etat, grille, virus, duration, name):
    images = []
    if etat == "sain":
        filenames = simulation_image_sains(grille, virus)
    elif etat == "malade":
        filenames = simulation_image_malades(grille, virus)
    elif etat == "mort":
        filenames = simulation_image_morts(grille, virus)
    elif etat == "gueri":
        filenames = simulation_image_gueris(grille, virus)
    else:
        return("etat incorrect")
    for filename in filenames:
        images.append(imageio.imread(filename))
    output_file =   name + '-' + etat + '-%s.gif' % datetime.datetime.now().
       strftime('%Y-%M-%d-%H-%M-%S')
    imageio.mimsave(output_file, images, duration=duration)

def population (grille):
    S=0
    Ma=0
    Mo=0
    G=0
    for i in range(grille.taille):
        for j in range(grille.taille):
            S+= grille.cellules[i][j].repartition[0]
            Ma+= grille.cellules[i][j].repartition[1]
            Mo+= grille.cellules[i][j].repartition[2]
            G+= grille.cellules[i][j].repartition[3]
    return (S,Ma,Mo,G,S+Ma+Mo+G)

def verif_geo(geo):
    im = Image.new("RGB", (len(geo), len(geo)), "white")
    for i in range(len(geo)):
        for j in range(len(geo)):
            if geo[i][j] == 90:
                im.putpixel((j, i), (200, 0, 0))
            elif geo[i][j] == 0:
                im.putpixel((j, i), (0, 0, 0))
            elif geo[i][j] == 2:
                im.putpixel((j, i), (100, 90, 100))
    plt.imshow(im)
    plt.show()

import random
import copy

# Repartition est [Sain, Malade, Mort, Gueri]
# Population est malade + sain + gueri


campagne = 3
ville = 90
route = 0
montagne = 2
```

```
class Cellule:
    def __init__(self, population, prob_mvt, coeff_attractivite):
        self.repartition = [population, 0, 0, 0]
        self.prob_mvt = prob_mvt
        self.coeff_attractivite = coeff_attractivite

    def population(self):
        return self.repartition[0] + self.repartition[1]+ self.repartition[3]

    def repartition_pr(self):
        repartition_pourcentage = []
        for i in self.repartition:
            repartition_pourcentage.append(i / sum(self.repartition))

        return repartition_pourcentage

    def repartition_pr_vivant(self):
        repartition_pourcentage = []
        for i in self.repartition:
            if self.population == 0:
                repartition_pourcentage.append(0)
            else :
                repartition_pourcentage.append(i / self.population())

        return repartition_pourcentage

    # virus = [proba_infection, proba_mort, proba_soin]
    def changement_interne(self, virus):
        ancien_etat = copy.copy(self.repartition)
        # On calcule les infectés
        if self.repartition[1] != 0:
            self.repartition[1] += int(ancien_etat[0] * virus[0])
            self.repartition[0] = ancien_etat[0] +  ancien_etat[1] - self.
                repartition[1]

        # On calcule les morts avant les gueris

        self.repartition[2] += int(self.repartition[1] * virus[1])
        self.repartition[1] -= int(self.repartition[1] * virus[1])

        self.repartition[3] += int(self.repartition[1] * virus[2])
        self.repartition[1] -= int(self.repartition[1] * virus[2])


# Fonctions auxilliaires servant à la classe grille.

def cases_touchees(x, y, rayon, minx, maxx, miny, maxy):
    coords = []

    for i in range(max(minx, x - rayon), min(maxx, x + rayon + 1)):
        for j in range(max(miny, y - rayon), min(maxy, y + rayon + 1)):
            if j == y and i == x:
                continue
            coords.append((i, j))

    return coords


def choix_proba(liste_proba):
    r = random.random()
    somme_cumulée = 0
```

```
    for i in range(len(liste_proba)):
        somme_cumulée += liste_proba[i]
        if r <= somme_cumulée:
            return i

    return random.choice(list(range(len(liste_proba))))


class Grille:
    # Geographie c'est une grille de valeur discretes décrivant l'environnement
    # 3 -> campagne
    # 90 -> ville
    # 0 -> routes
    # 2 -> montagnes

    def __init__(self, taille, geographie, population_totale):
        self.taille = taille
        self.population_initiale = population_totale
        total_geo = sum([item for sublist in geographie for item in sublist])
        self.cellules = []
        for i in range(taille):
            sous_liste = []
            for j in range(taille):
                if(geographie[i][j] == campagne):
                    attractivite = 0.1
                    proba_mvt = 0.1
                elif(geographie[i][j] == ville):
                    attractivite = 0.3
                    proba_mvt = 0.5
                elif(geographie[i][j] == route):
                    attractivite = 0.7
                    proba_mvt = 0.7
                elif(geographie[i][j] == montagne):
                    attractivite = 0.1
                    proba_mvt = 0.1

                # On ajoute une part d'aléatoire dans la répartition de la
                    population

                sous_liste.append(Cellule(int(population_totale * geographie[i][
                    j] / total_geo * random.uniform(0.98, 1.02)), proba_mvt,
                    attractivite))

            self.cellules.append(sous_liste)

    def next(self, virus):
        # On applique d'abord les changements relatifs à la maladie.
        for i in range(len(self.cellules)):
            for j in range(len(self.cellules)):
                self.cellules[i][j].changement_interne(virus)

        # On calcule ensuite les mouvements de population
        nouvelle_cellules = copy.deepcopy(self.cellules)

        for i in range(len(self.cellules)):
            for j in range(len(self.cellules[i])):
                repartitions = self.cellules[i][j].repartition_pr_vivant()
                # Calcul de la population partie de la case (i, j)

                population_partie = int(self.cellules[i][j].population() * self.
                    cellules[i][j].prob_mvt)
```

```
                # On répartit les pertes de population

                sains_partis = int(population_partie * repartitions[0])
                malades_partis = int(population_partie * repartitions[1])
                soignes_partis = int(population_partie * repartitions[3])

                nouvelle_cellules[i][j].repartition[0] -= sains_partis
                nouvelle_cellules[i][j].repartition[1] -= malades_partis
                nouvelle_cellules[i][j].repartition[3] -= soignes_partis


                # Choix de la case où la population part
                # Ici le rayon vaut 1 car on considére que les populations ne
                    peuvent, par exemple
                #Â prendre l'avion.
                voisins = cases_touchees(i, j, 1, 0, self.taille, 0, self.taille
                    )

                liste_probas = [self.cellules[x][y].coeff_attractivite for x, y
                    in voisins]
                nouvelle_liste_probas = []

                # On recalcule les probas afin que la somme des probas fasse 1.

                for k in range(len(liste_probas)):
                    nouvelle_liste_probas.append(liste_probas[k]/sum(
                        liste_probas))

                x, y = voisins[choix_proba(nouvelle_liste_probas)]

                nouvelle_cellules[x][y].repartition[0] += sains_partis
                nouvelle_cellules[x][y].repartition[1] += malades_partis
                nouvelle_cellules[x][y].repartition[3] += soignes_partis

        self.cellules = nouvelle_cellules


    def stats(self):
        valeur_de_retour = [0, 0, 0, 0]
        for ligne in self.cellules:
            for cellule in ligne:
                for i in range(4):
                    valeur_de_retour[i] += cellule.repartition[i]
        return valeur_de_retour


# Fonctions utilisées pour débugger.

def grille_pop(g):
    L = []
    for i in range(len(g.cellules)):
        D = []
        for j in range(len(g.cellules[0])):
            D.append(int(g.cellules[i][j].population()))
        L.append(D)
    return L


def grille_pop_sum(g):
```

```
somme = 0
for i in range(len(g.cellules)):
    for j in range(len(g.cellules)):
        somme += g.cellules[i][j].population()

return somme
```

Commentaire bibliographique

[?] : Cet article nous a permis de bien comprendre la théorie des automates cellulaires, et de mettre l'accent sur les paramètres intéressants qu'une simulation par automate cellulaires permettait de prendre en compte. Il présente aussi les principaux avantages des automates cellulaires par rapport à la méthode différentielle.

[?] : Cet article, en plus de présenter un autre modèle d'automates cellulaires, fourni une liste relativement exhaustive de paramètres à prendre en compte dans la simulation(même si les auteurs en ont négligé une partie qui ne présentait qu'un intérêt minime.

[?] : Cet article présente plus en profondeur les différences des autres méthodes, et fournit une approche intéressante en matière de résultats et de leur présentation.

[?] : Cet article présente la méthode différentielle, et est très fourni en graphiques bien expliqués.

[?] : Ce livre nous a fournit des éléments sur l'histoire de l'épidémiologie.