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 attardé ici à la simulation par automate cellulaire.

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 2 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 4 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 rattraper 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.

1.2 Résultats

Voici quelques étapes d'une simulation de notre automate. Le foyer original -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.

Dans la simulation présentée ci-dessus, 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 comportement 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 étape de la simulation.

FIGURE 1 – Étape 36

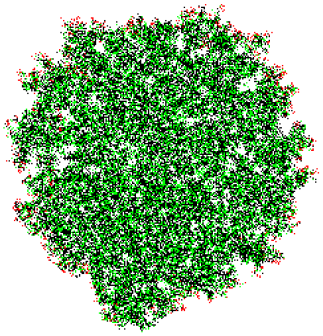


FIGURE 2 – Étape 195

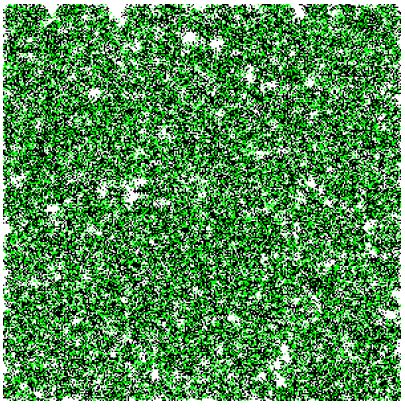
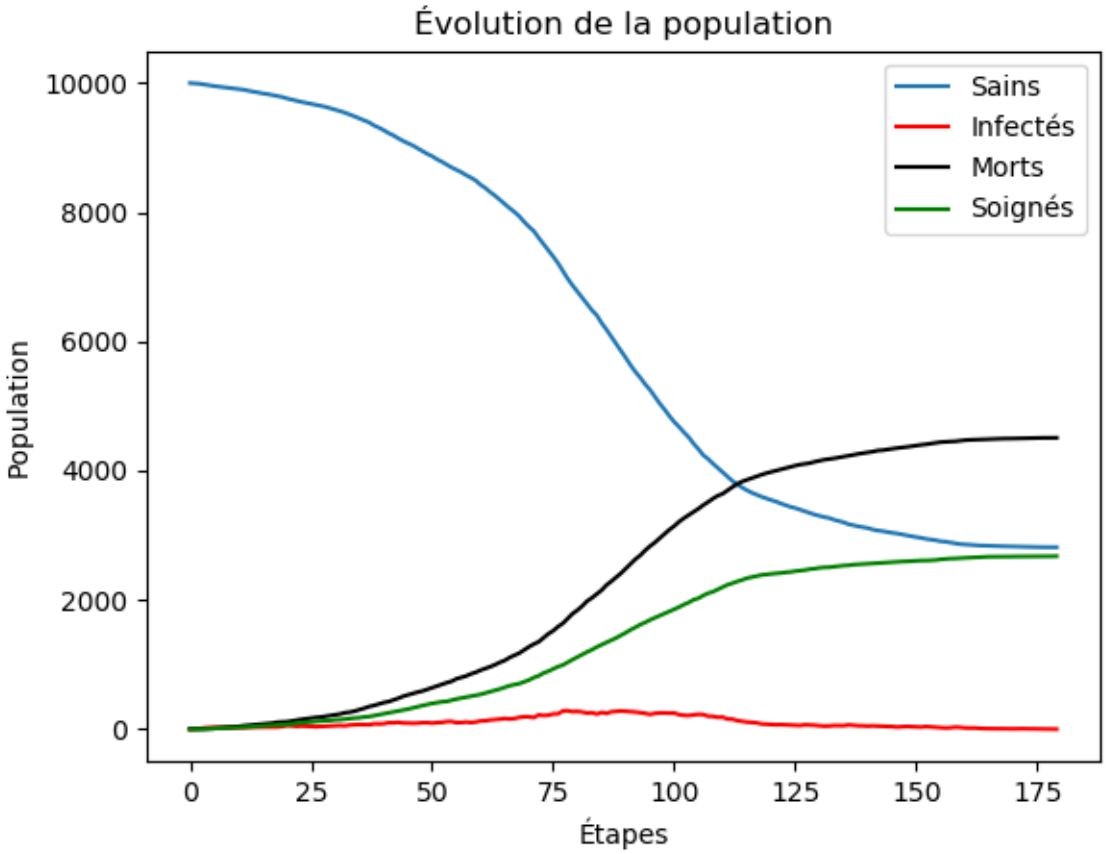
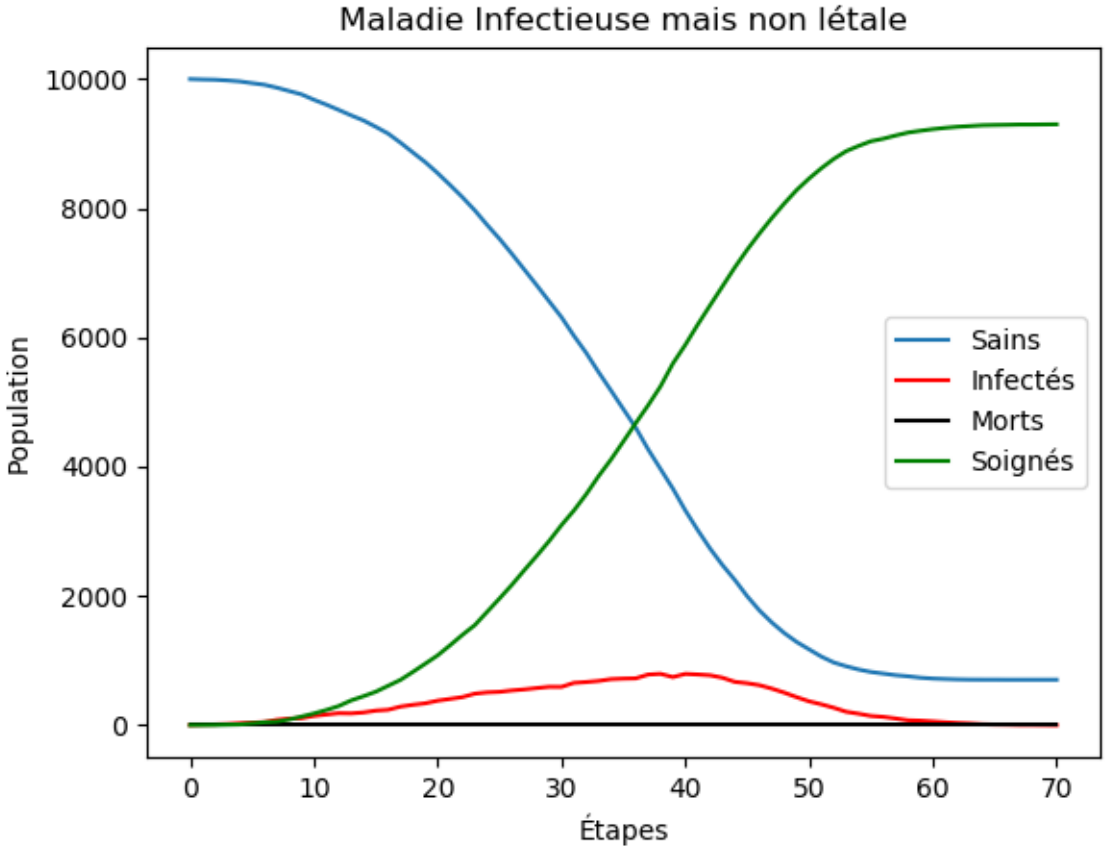


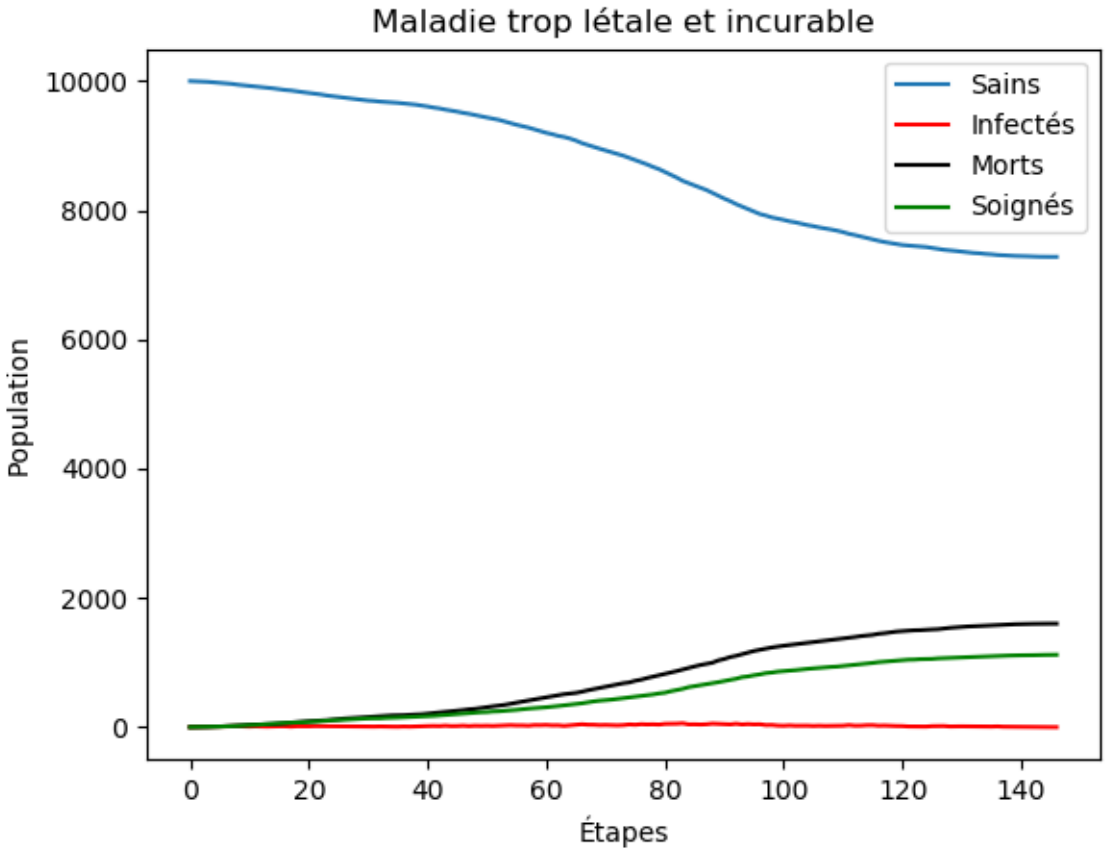
FIGURE 3 – Étape 412



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 collé 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

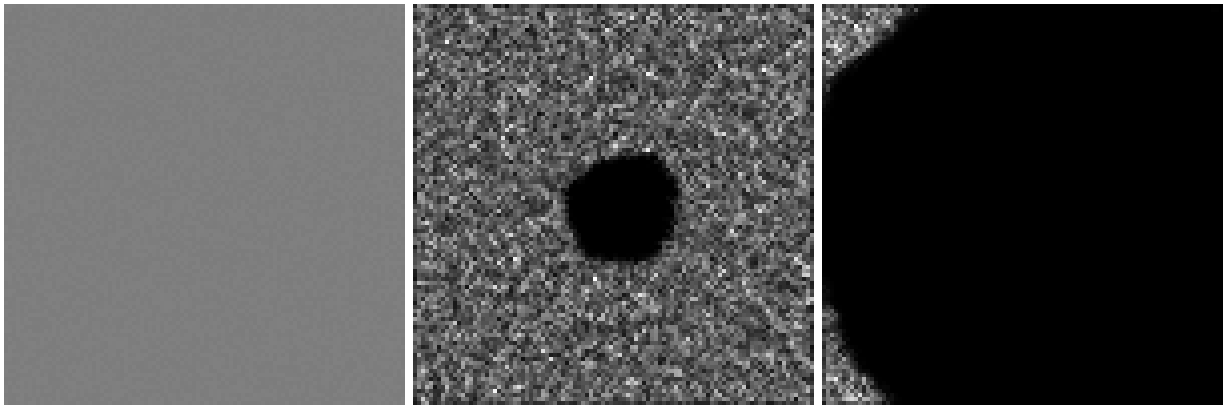
Ce deuxième automate prend en compte plus de paramètres :

- 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

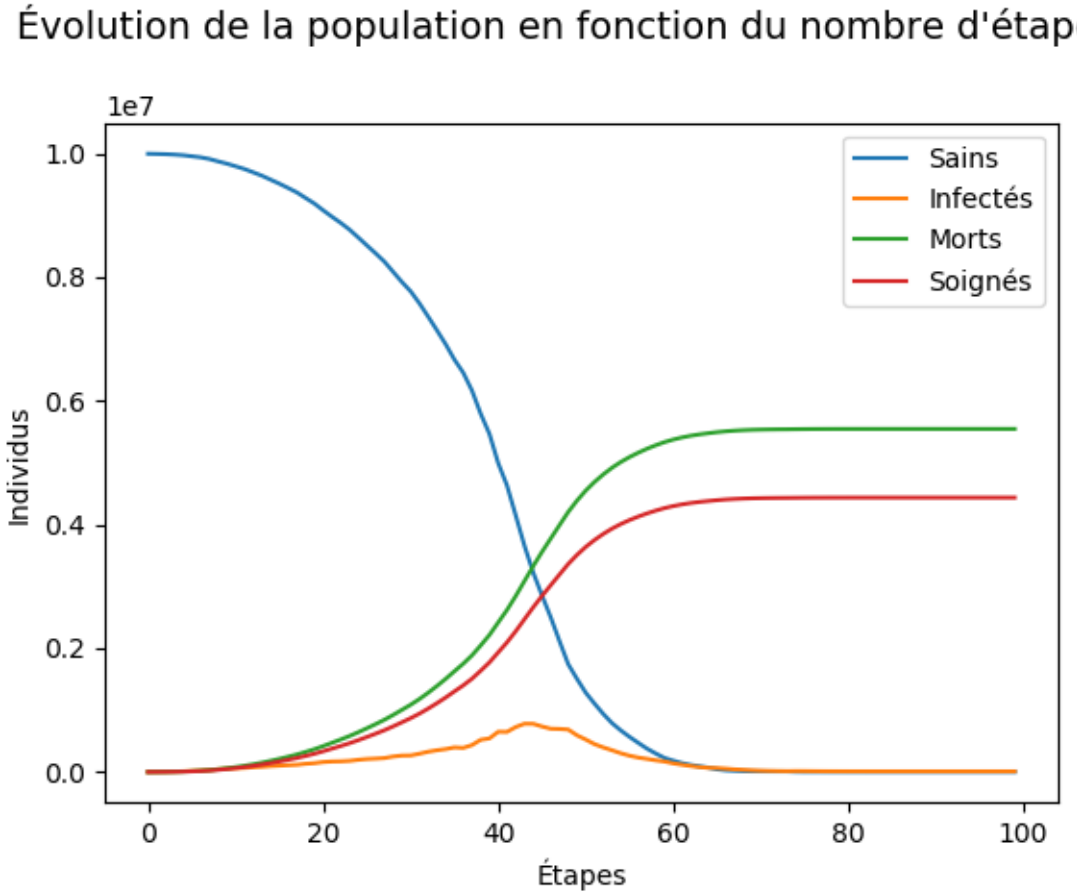
Les résultats graphiques de cet automate permettent d'afficher la densité de population des quatre états au sein de toutes les cellules de la grille. Bien sûr, il n'est maintenant plus possible d'afficher tous les paramètres de la grille sur une seule image. Voici par exemple la répartition des individus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



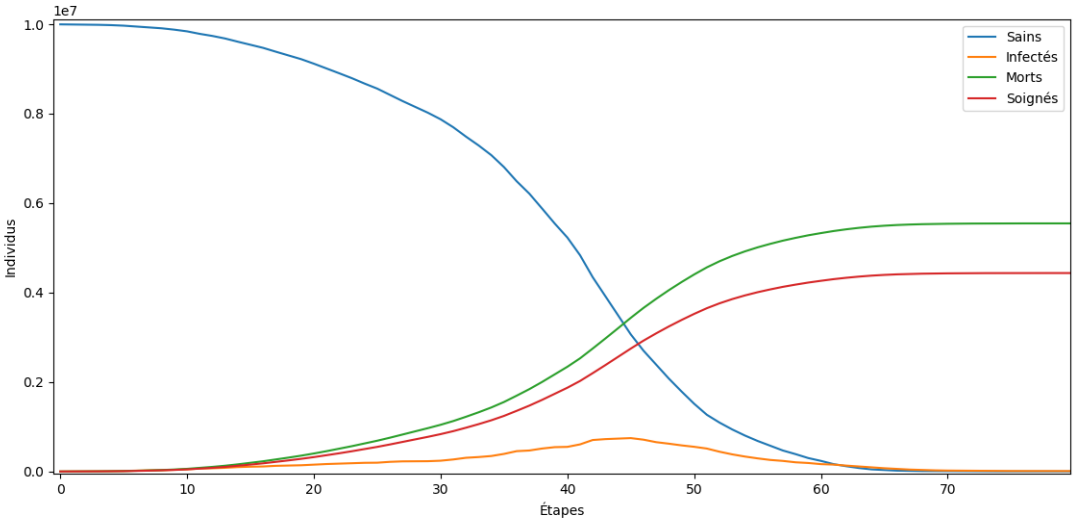
Ici, plus une cellule contient de personnes saines, plus le pixel la représentant sur l'image est blanc. Les cellules sont donc toutes légèrement grisées à la première étape, non pas parce qu'ellles contiennent des individus malades, morts ou guéris, mais parce que le programme étalonne automatiquement 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 presenter 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.

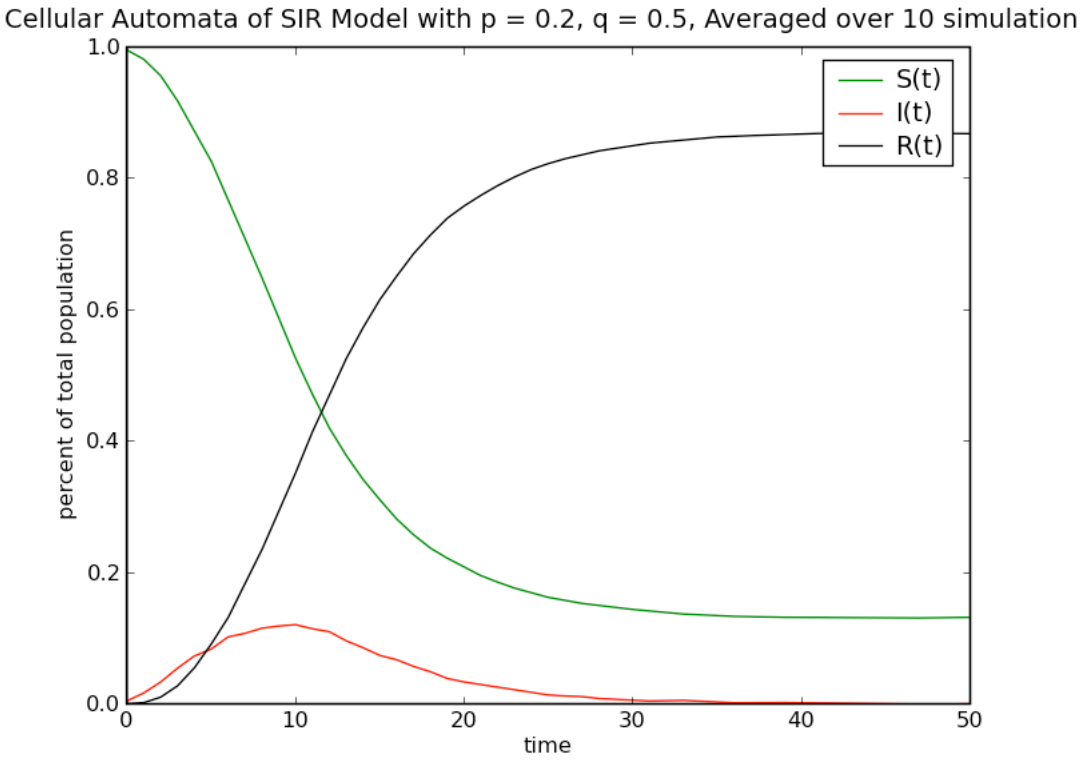
2.3 Comparaison

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.

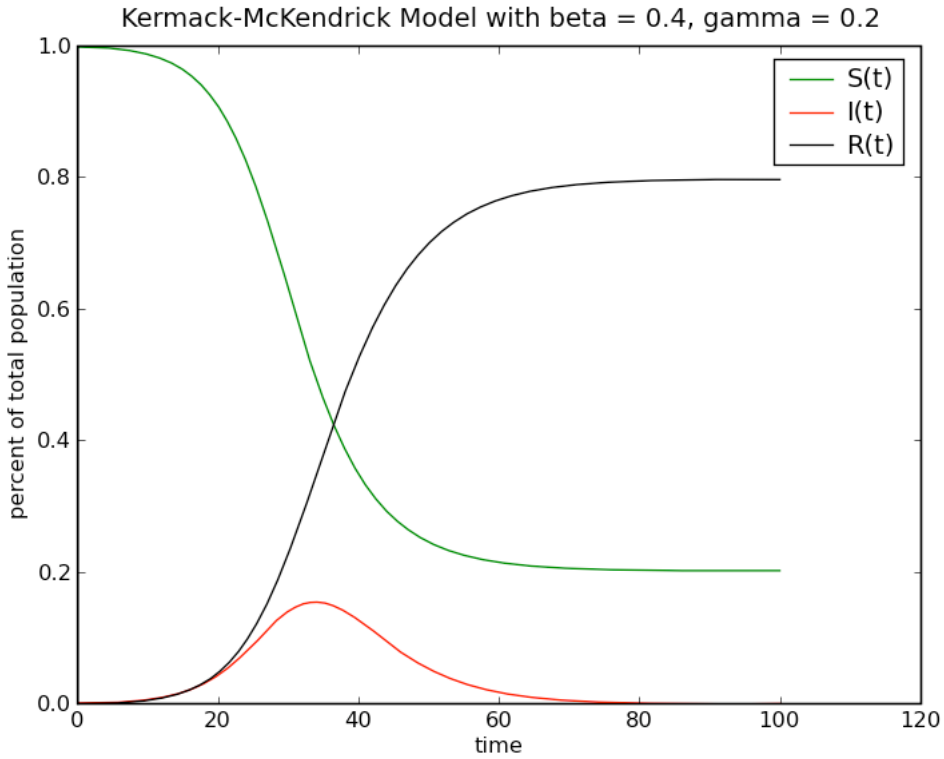


Évolution de la population en fonction du nombre d'étapes





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 Appendice

3.1 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


def init_grid(size):
    # G = []
    # for i in range(size):
    #     sG = []
    #     for j in range(size):
    #         sG.append(0)
    #     G.append(sG)
    #G[size // 2][size // 2] = 1
    # return G
    GEO = [[3, 3, 3], [3, 90, 3], [3, 3, 3]]
    g = Grille(3, GEO, 10000000)
    return g


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 contact(grille, i, j):
    compteur = 0
    for i in cases_touchees(i, j, rayon_infection, 0, len(grille), 0, len(grille
        [0])):
        if grille[i[0]][i[1]] == 1:
            compteur += 1

    return compteur
```

```
# Cette fonction retourne la probabilité d'être infecté, dépendant du nombre de
# contacts avec des infectés et une probabilité dépendante de la maladie

def calcul_proba_infection(contacts, rayon, base_proba):
    max_contact = (1 + 2 * rayon)**2 - 1
    return base_proba * math.log(1 + contacts) / math.log(1 + max_contact)


def binomial(p):
    if random.random() <= p:
        return 1
    return 0


def mort_ou_soigne(p1, p2):
    if random.random() <= p1:
        return 2
    if random.random() <= p2:
        return 3
    return 1


def etat_suivant(grille):
    G = init_grid(len(grille))
    for i in range(len(grille)):
        for j in range(len(grille[i])):
            if grille[i][j] == 0:
                contacts = contact(grille, i, j)
                G[i][j] = binomial(calcul_proba_infection(
                    contacts, rayon_infection, proba_infection))
            elif grille[i][j] == 1:
                G[i][j] = mort_ou_soigne(proba_mort, proba_infection)
            else:
                G[i][j] = grille[i][j]

    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 print_grille(grille):
    for i in grille:
        print(i)


def simulation(taille):
    grille = init_grid(taille)
    etape = 0
    while compte(grille)[1] != 0:
        etape += 1
        grille = etat_suivant(grille)
```

```
    return etape, compte(grille)


def statistiques_2(grille, virus):
    k = 0
    stats = []
    while compte(grille)[1] != 0:
        k += 1
        print(k)
        stats.append(grille.stats())
        grille.next(virus)
        if k > 99:
            break
    sains, infectes, morts, soignes = 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 statistiques(taille, echantillon):
    etapes = []
    comptes = []

    for i in range(echantillon):
        etape, compte = simulation(taille)
        etapes.append(etape)
        comptes.append(compte)
        print(i)

    sains, infectes, morts, soignes = zip(*comptes)
    print('Moyenne = {}'.format(sum(etapes) / len(etapes)))
    print('Max = {}'.format(max(etapes)))
    print('Sains:\n\tMoyenne = {}'.format(sum(sains) / len(sains)))
    print('\tMax = {}'.format(max(sains)))
    print('\tMin = {}'.format(min(sains)))
    print('Infectés:\n\tMoyenne = {}'.format(sum(infectes) / len(infectes)))
    print('\tMax = {}'.format(max(infectes)))
    print('\tMin = {}'.format(min(infectes)))
    print('Morts:\n\tMoyenne = {}'.format(sum(morts) / len(morts)))
    print('\tMax = {}'.format(max(morts)))
    print('\tMin = {}'.format(min(morts)))
    print('Soignés:\n\tMoyenne = {}'.format(sum(soignes) / len(soignes)))
    print('\tMax = {}'.format(max(soignes)))
```

```
    print('\tMin = {}'.format(min(soignes)))


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("D:\\autres\\Me\\Github\\pics\\"+str(k)+'_sains'+'.png')
        gif.append("D:\\autres\\Me\\Github\\pics\\"+str(k)+'_sains'+'.png')
        grille.next(virus)
        if k > 9999:
            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)) > 10000000:
            print(compte(grille))
            print(k)
        im = transition_image_malades(grille, im, coeff)
        im.save("D:\\autres\\Me\\Github\\pics\\"+str(k)+'_malades'+'.png')
        gif.append("D:\\autres\\Me\\Github\\pics\\"+str(k)+'_malades'+'.png')
        grille.next(virus)
        print(population(g)[1])
        if k > 9999:
            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("D:\\autres\\Me\\Github\\pics\\"+str(k)+'_morts'+'.png')
        gif.append("D:\\autres\\Me\\Github\\pics\\"+str(k)+'_morts'+'.png')
        grille.next(virus)
        if k > 9999:
            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("D:\\autres\\Me\\Github\\pics\\"+str(k)+'_gueris'+'.png')
        gif.append("D:\\autres\\Me\\Github\\pics\\"+str(k)+'_gueris'+'.png')
        grille.next(virus)
        if k > 9999:
            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 = "D:\\autres\\Me\\Github\\pics\\"+ name + '-' + etat + '-%s.gif
        ' % datetime.datetime.now().strftime('%Y-%M-%d-%H-%M-%S')
    imageio.mimsave(output_file, images, duration=duration)

##
if __name__ == '__main__':
    #GEO2 = [[3 for i in range(100)] for j in range(100)]
    g = Grille(100, GEO2, 1000000)
    g.cellules[50][50].repartition[1] += 300
    g.cellules[50][50].population += 300

    virus = [1, 0.1, 0.1]
##
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


# Fonction 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
```

Références

[1] Sharon Chang. Cellular automata model for epidemics. page 12. Cet article présente plus en profondeur les différences des autres méthodes, et fournit une approche intéréssante en matière de résultats et de leur présentation.

[2] Shih Ching Fu and George Milne. Epidemic modelling using cellular automata. page 15. Cet article, en plus de présenter un autre modèle d'automates cellulaires, founit une liste relativement exhaustive de paramètres à prendre en compte dans la simulation(même si les auteurs en ont négligé une partie qui ne pésentait qu'un intérêt minime.

[3] Alfredo MORABIA. Historical developments in epidemiology, pages 92-93. Ce livre nous a fournit des éléments sur l'histoire de l'épidémiologie.

[4] G. Rodriguez Sánchez S. Hoya White, A. Martin del Rey. Using cellular automata to simulate epidemic diseases. *Applied Mathematical Sciences, Vol. 3*, pages 959 – 968, 2007. Cet article nous a permis de bien comprendre la théorie des automates cellulaires, et de mettre l'accent sur les paramètres intéréssants 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.

[5] Victor CAMEO PONZ Laura LANCE Alban MERCIER Jérémy RISSO BOURGÈS Cécile VASSEUR Aurore VIMONT. Modélisation de la propagation d'un virus. Cet article présente la méthode différentielle, et est très fourni en graphiques bien expliqués.