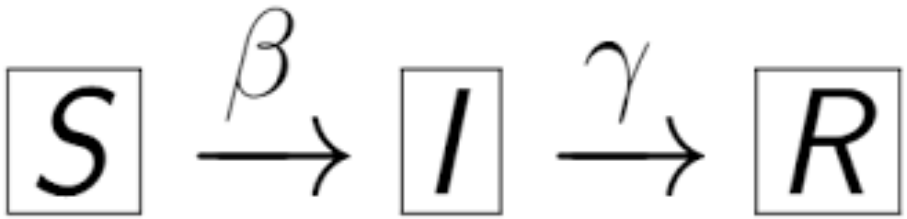
# Simulation de la propagation d'un virus à l'aide de systèmes multi-agents

Mathurin de Crécy

8 juin 2022

# Sommaire

# Le modèle SIR



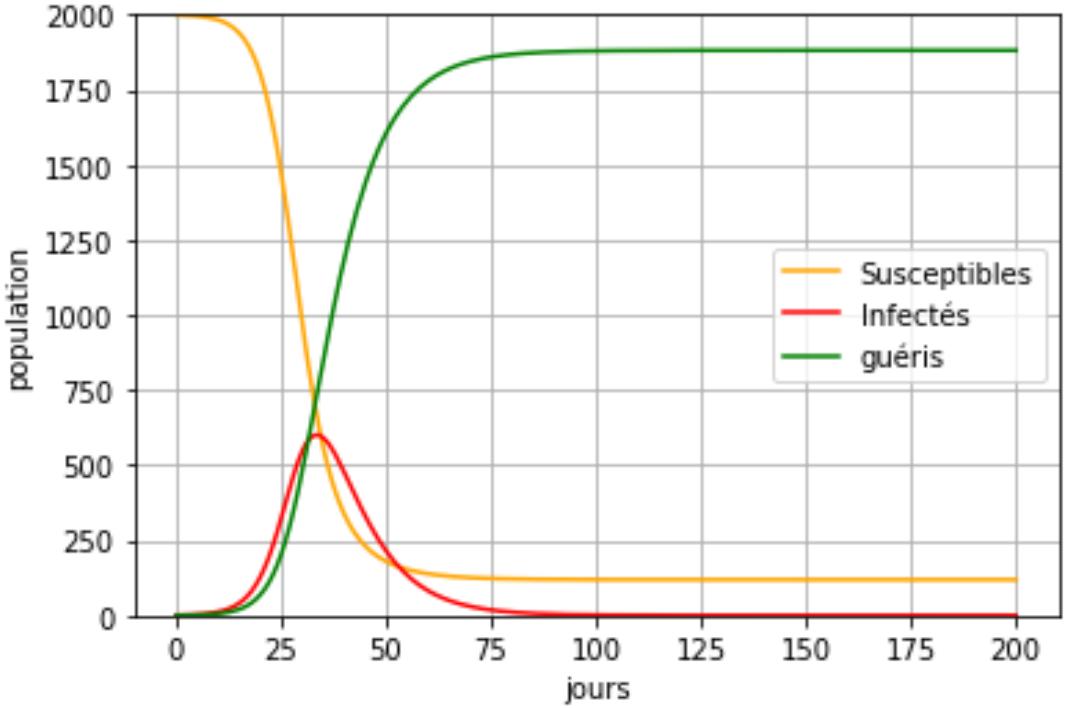

Le modèle SIR est le plus célèbre modèle épidemiologique pouvant prédire la propagation d'un virus. Il s'agit d'un modèle à équations différentielles dans lequel la population est divisée en catégories Sains, Infféctés et Rémis. Les échanges entre ces catégories sont régis par les équations suivantes:

$$\begin{cases} \frac{dS}{dt} = -\frac{S\beta}{N} I \\ \frac{dI}{dt} = \frac{S\beta}{N} I - \gamma I \\ \frac{dR}{dt} = \gamma I \end{cases}$$

## Définitions

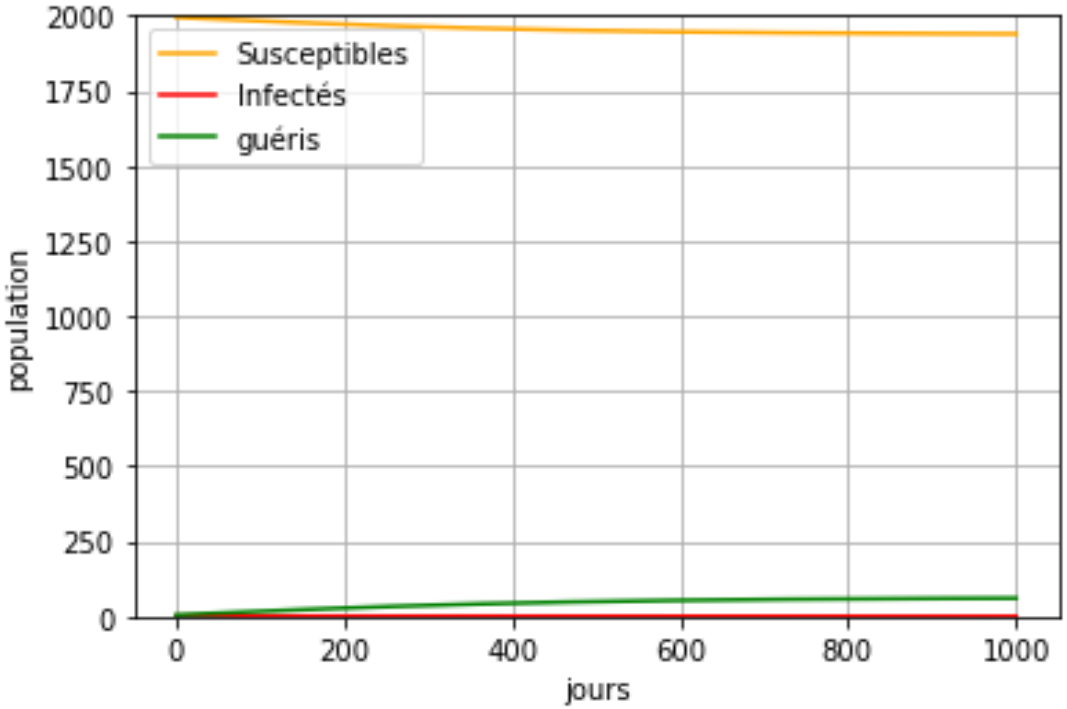
- $\gamma = 1/\text{Tr}$
- $\beta = \text{KP}$
- $\text{R0} = \beta/\gamma$

Les paramètres $\gamma$ et $\beta$ désignent respectivements le taux de guérison où Tr est le temps de guérison et le taux de contagion où P est la probabilité de transmission lors d'un contact et K est le nombre de contact d'une personne lors d'une journée. Enfin, R0 est ce que l'on appel le taux de reproduction de la maladie, il représente le nombre de personnes saines qu'un infecté vas contaminer en une journée.




Etude d'une population de 1200 habitants avec un patient initial sur 200 jours. R0=3 (covid-19 pré-confinement), temps de rémission de 8 jours. Sauf précision on reprendra ces valeurs dans les autres simulations. L'imunité collective intevient lorsque les paramètre N et S conduisent à $\frac{dI}{dt} \leq 0$ c'est à dire que $\frac{S\beta}{N} \leq \gamma$ donc, $S \leq \frac{N\gamma}{\beta} = \frac{N}{R0}$. Dans cet exemple, cela correspond à S=$\frac{N}{3}$, il faut donc que les 2/3 de la population (800 personnes) soient contaminées avant d'avoir une immunitée collective efficace.

# Théorème du seuil et imunité collective



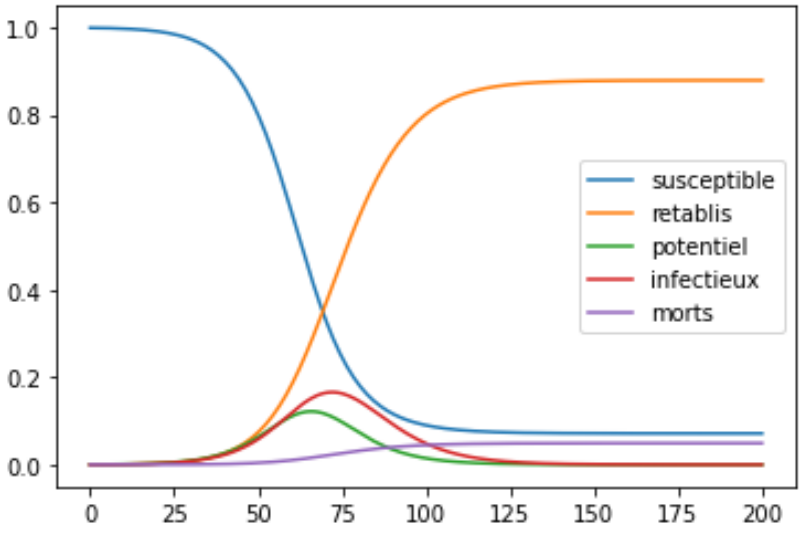

Etude d'une population de 1200 habitants avec 5 patients initiaux sur 1000 jours. R0=1

Le théorème du seuil stipule que si R0$\leq$1 alors l'épidemie ne se propage pas. Ce théorème peut être facilement prouvé à l'aide d'une étude de la fonction régissant I sachant que $\beta = \gamma$.

# La variante SEIR

$$\boxed{S} \xrightarrow{\beta} \boxed{E} \xrightarrow{\alpha} \boxed{I} \xrightarrow{\gamma} \boxed{R}$$

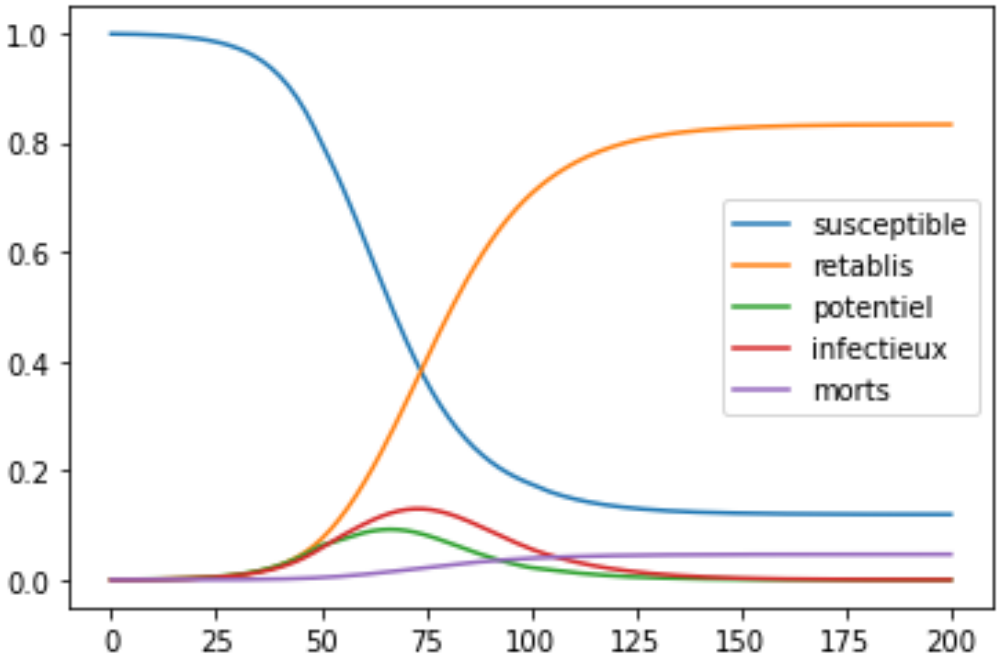
$$\begin{cases} \frac{dS}{dt} = -\frac{S\beta}{N} I \\ \frac{dE}{dt} = \frac{S\beta}{N} I - \alpha E \\ \frac{dI}{dt} = \alpha E - \gamma I - \mu I \\ \frac{dR}{dt} = \gamma I \\ \frac{dF}{dt} = \mu I \end{cases}$$




Ici on rajoute une catégorie E pour le temps d'incubation de la maladie, le coefficient $\alpha$ correspond à l'inverse du temps d'incubation. F est la catégorie des décès, $\mu$ correspond à la létalité de la maladie.

# Les gestes barrières

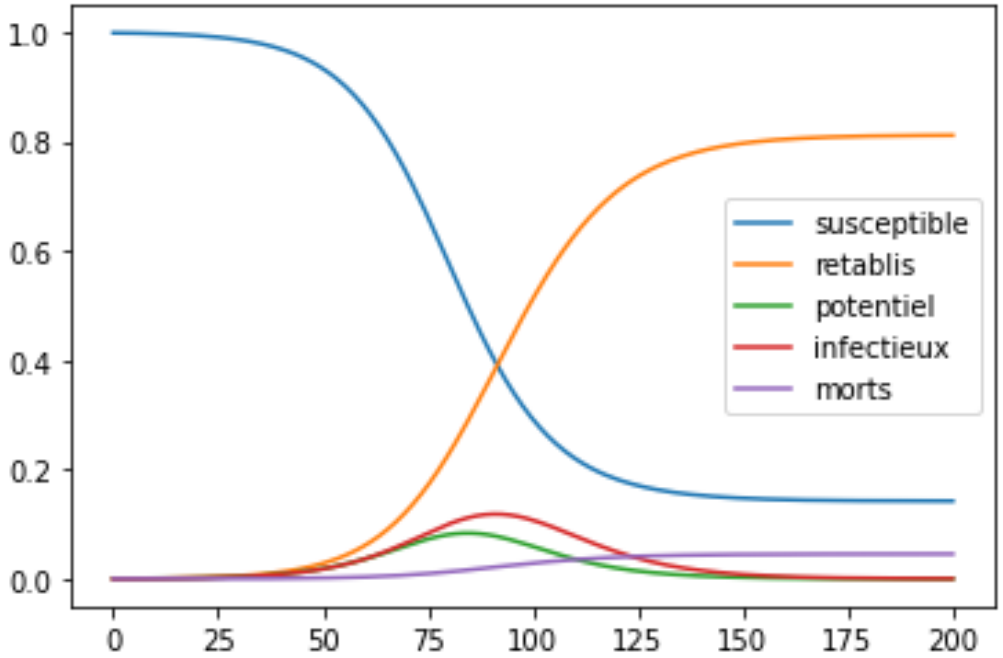
Port du masque du 50ème au 100ème jour




$$\begin{cases} \frac{dS}{dt} = -\frac{S\beta}{N} I \\ \frac{dE}{dt} = \frac{(1-u)S\beta}{N} I - \alpha E \\ \frac{dI}{dt} = \alpha E - \gamma I - \mu I \\ \frac{dR}{dt} = \gamma I \\ \frac{dF}{dt} = \mu I \end{cases}$$

Pour rendre compte de l'efficacité des gestes barrières, on multipli $\beta$ par un coefficient 1-u où u est un paramètre (déterminé expérimentalement) représentant l'efficacité de cette mesure, ici u=0.2.

Port du masque tout au long de l'épidémie

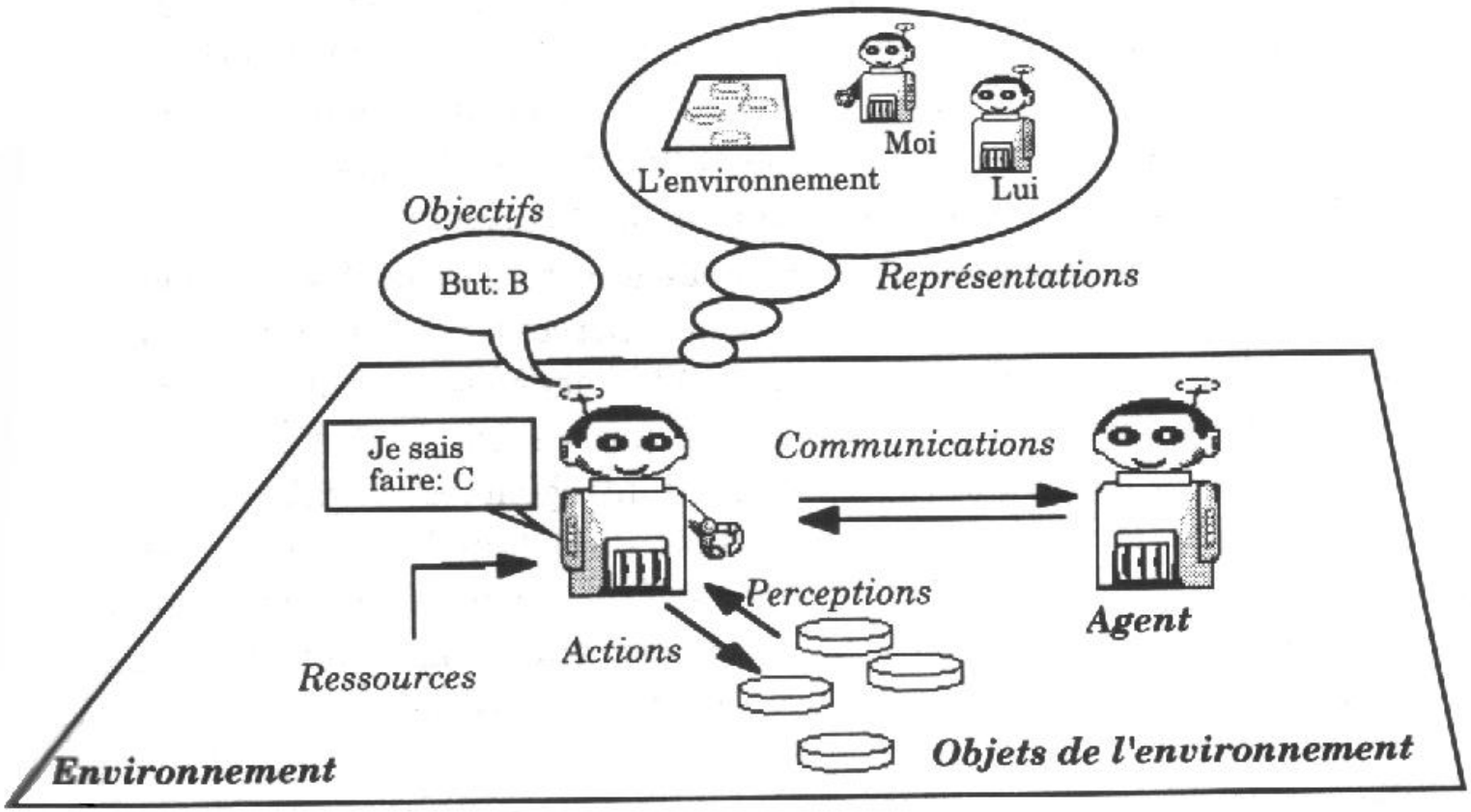



On constate que le port du masque tout au long de l'épidémie permet deretarder le pic de contamination, mais ne le diminue pas beaucoup.

# Les faiblesses du modèle

- ne rends pas conte des interactions spatiales
- ne considère pas les comportements sociaux
- impossibilité de mener une étude à l'échelle de l'individu

# Les Systèmes Multi-Agents (SMA)




Les SMA coordonnent plusieurs entités informatiques appelées agents qui ont un but, une perception de leur environnement et des autres agents et agissent de concerts.

# A quoi ça sert?

- coordination d'entités informatiques
- gestion d'un environnement
- simulation de phénomènes complexes

# L'agent

$$\langle P, Percept, F, I, S \rangle$$

Un agent est défini par cinq éléments:

- P: la fonction de perception
- Percept: les conséquences de l'environnement sur l'agent
- F: les possibles modifications de l'état de l'agent
- I: les fonctions qui modifient l'état de l'agent
- S: les états possibles de l'agent

# L'environnement

$$\langle E, \Gamma, \Sigma, R \rangle$$

Un environnemnt est définie par quatre éléments:

- E: l'espace dans lequel les agents évolueront
- $\Gamma$: comment les agents peuvent modifier les états de l'environnement
- $\Sigma$: les états de l'environnement
- R: la façon dont l'environnement se modifie lui-même

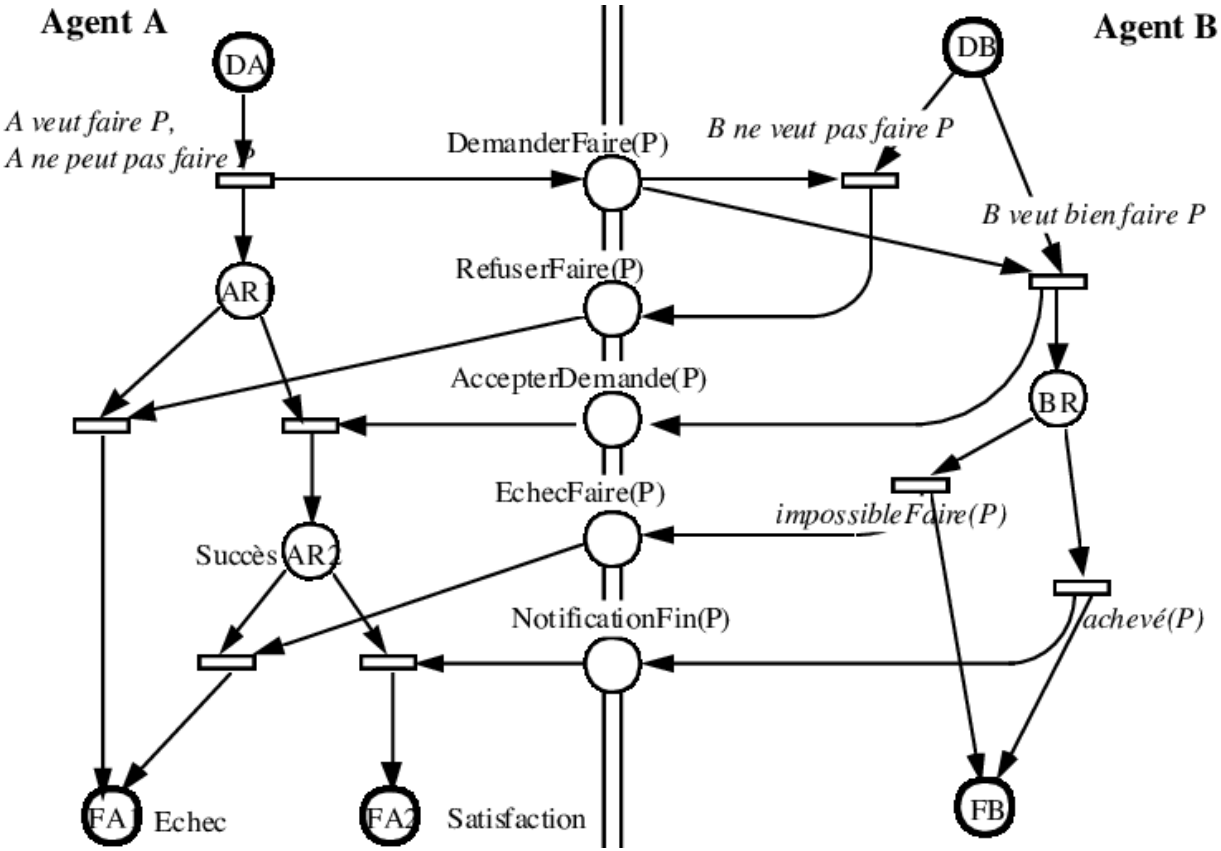
# Les relations agents-environnement

Les fonctions des agents et de l'environnement prennent en arguments les états des agents et de l'environnnement et définissent ainsi leurs états futures. P:$\Sigma \rightarrow$ Percept F:S$\times$Percept$\rightarrow$S I:S$\rightarrow \Gamma$ R:$\Sigma\times\Gamma\rightarrow\Sigma$

Ce qui se réécrit en $\begin{cases} s(t+1) = F(s(t) \cdot P(\sigma(t))) \\ \sigma(t+1) = R(\sigma(t) \cdot I(s(t))) \end{cases}$

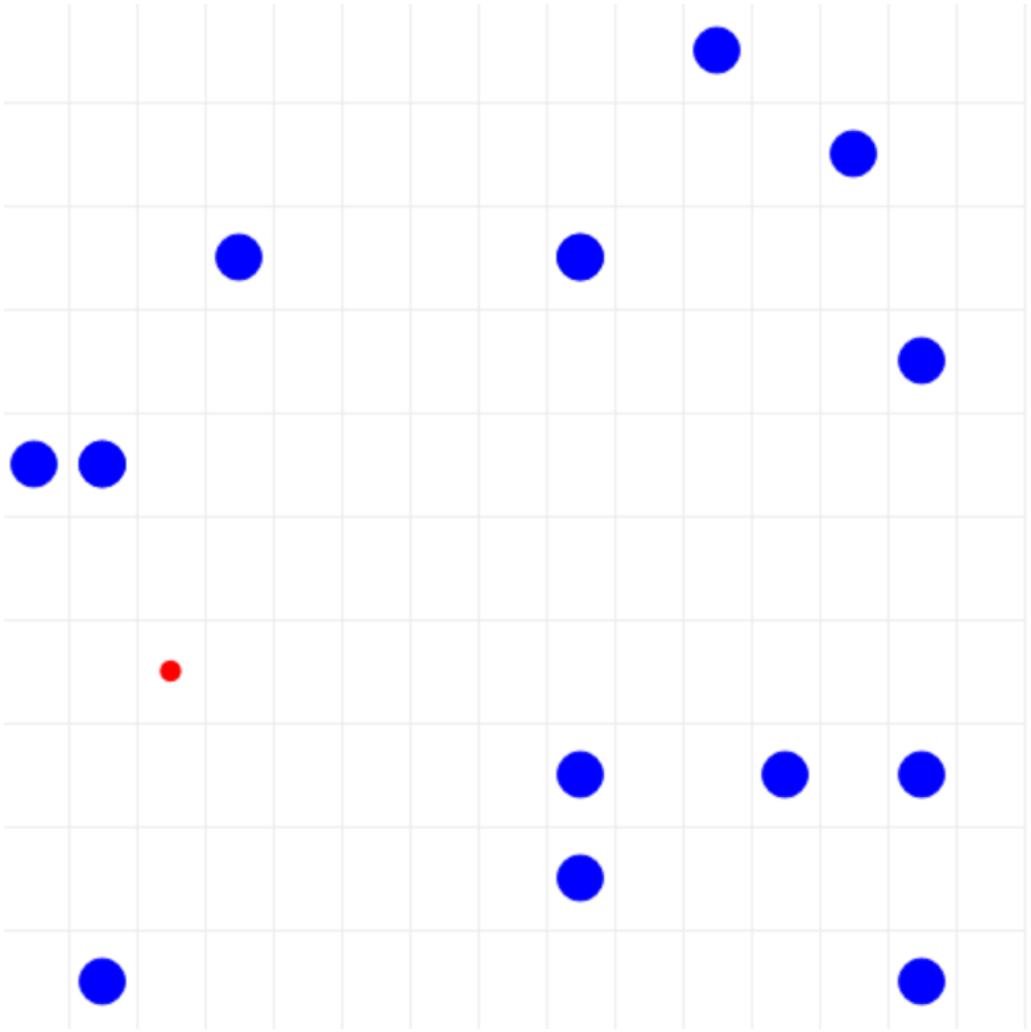
$s \in S$

$\sigma \in \Sigma$

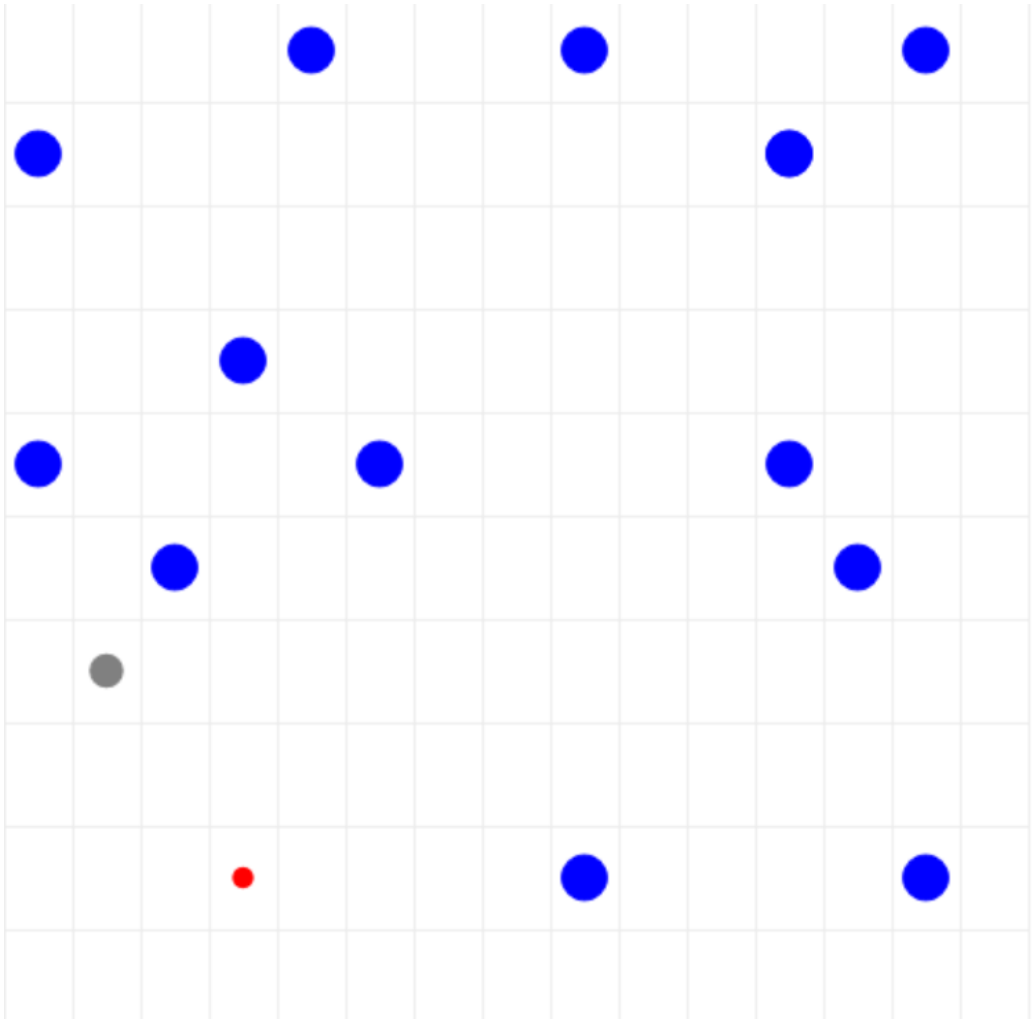



Exemple de communication entre deux agents schématiser par un réseau de Petri.

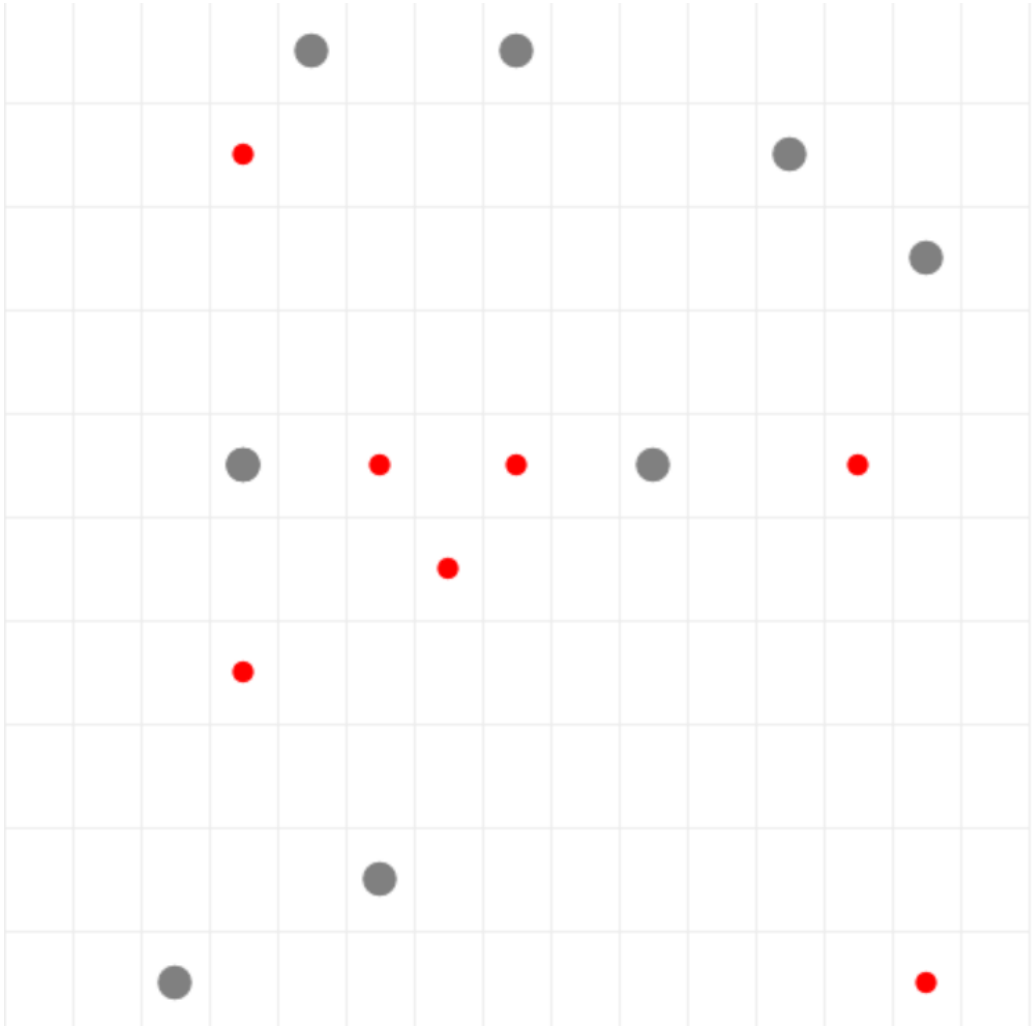
# La simulation



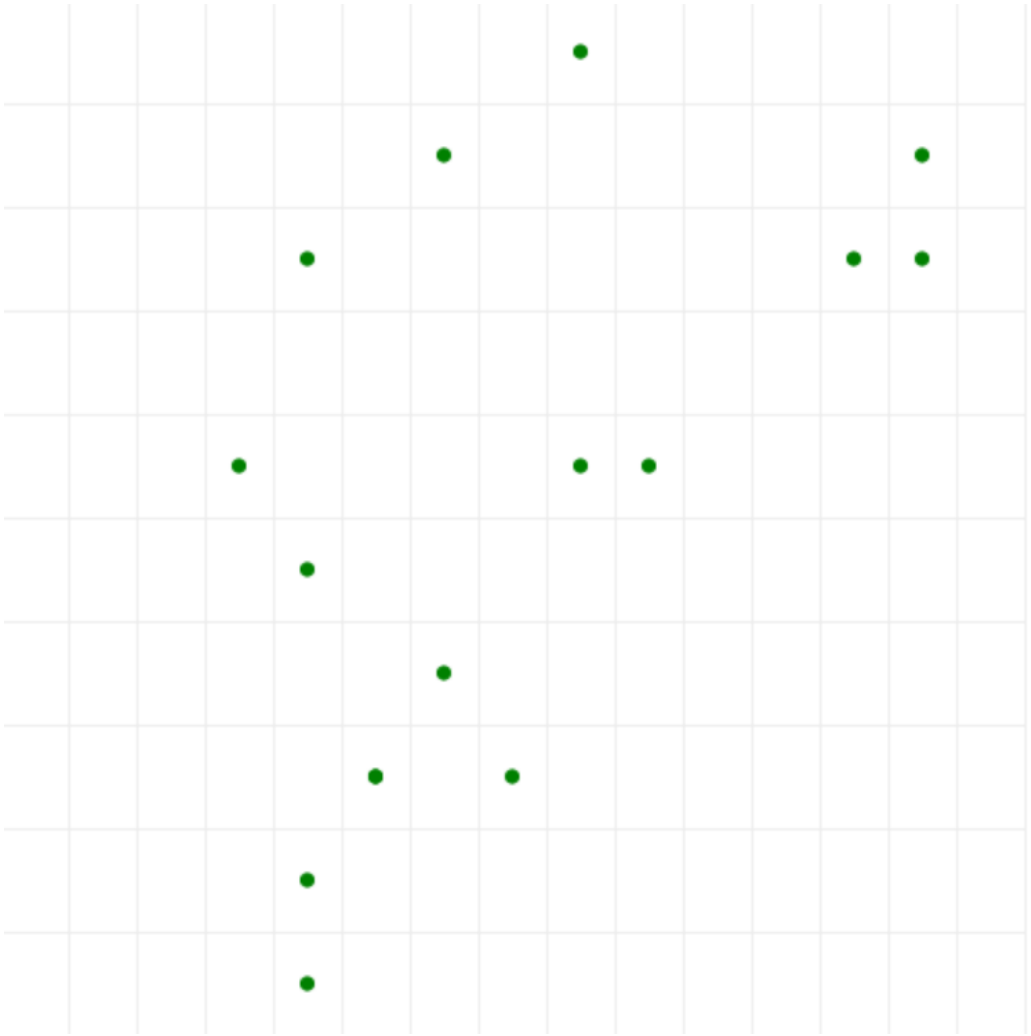
représentation d'agents sains (bleu) et d'agents contagieux (rouge) dans un espace fermé.



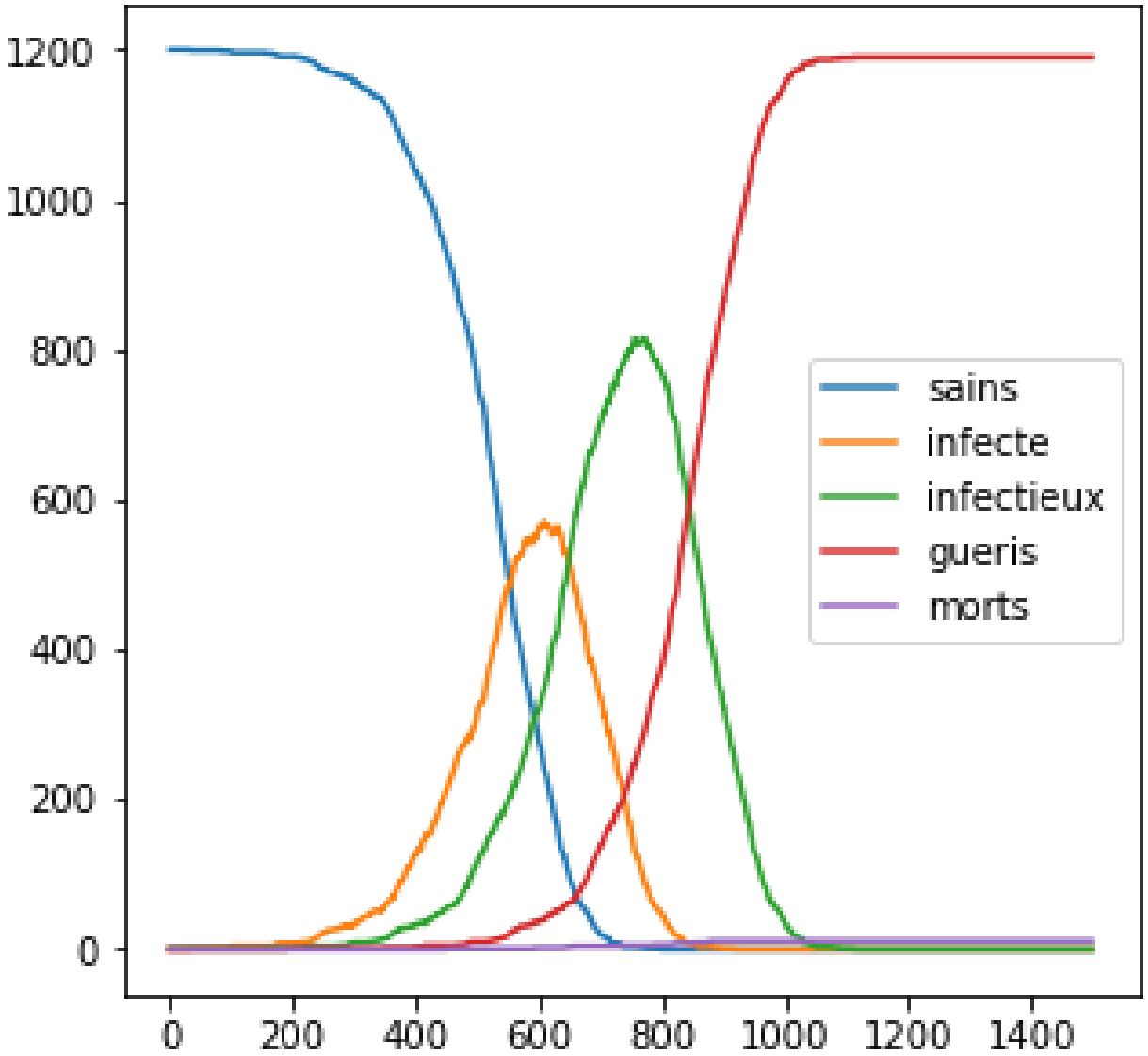
Les agents se déplacent dans l'espace et interagissent, lorsqu'un agent infectieux rencontre un agent sains, il peut le contaminé, il est alors en phase d'incubation (gris).



Au bout d'un certain temps, les agents contaminés deviennent eux-mêmes contagieux.



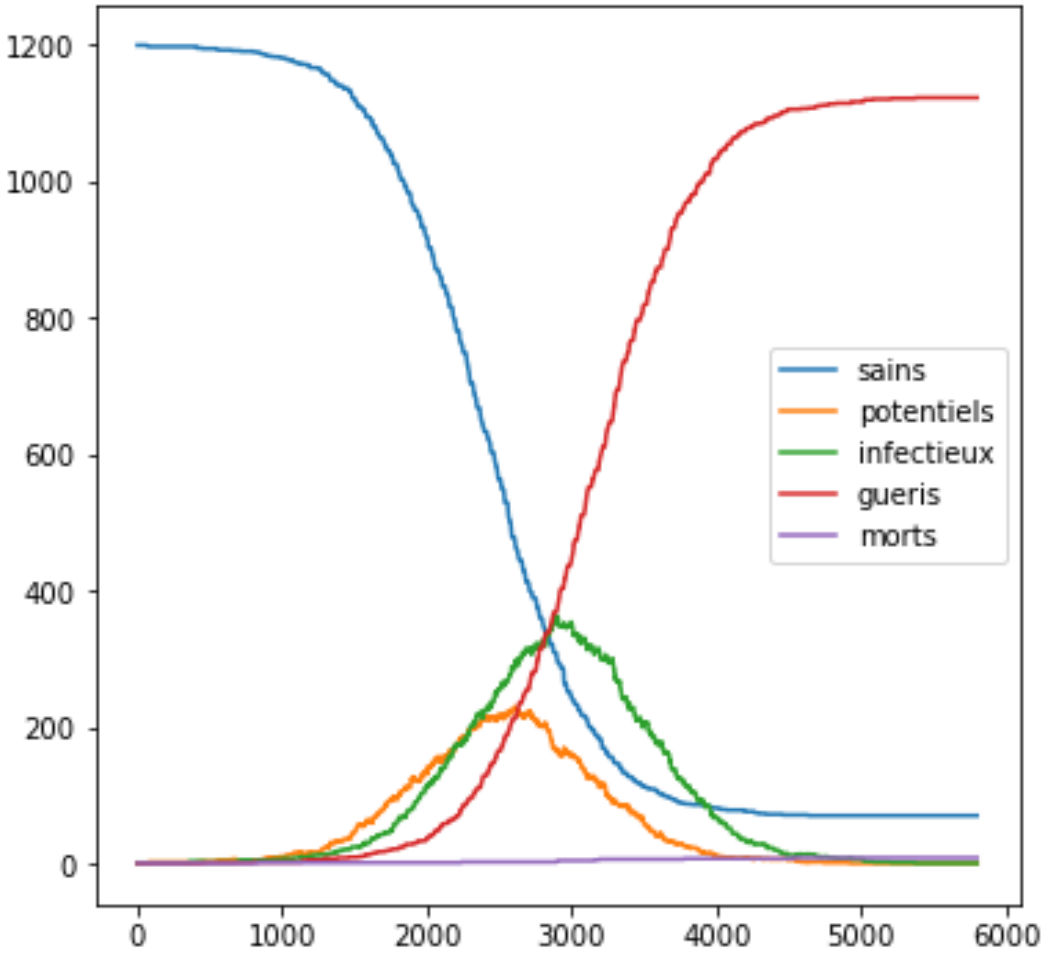
Au bout du temps de guérison, les agents sont guéris (verts).




La courbe d'évolution de la simulation (abscisse en demi-heures).

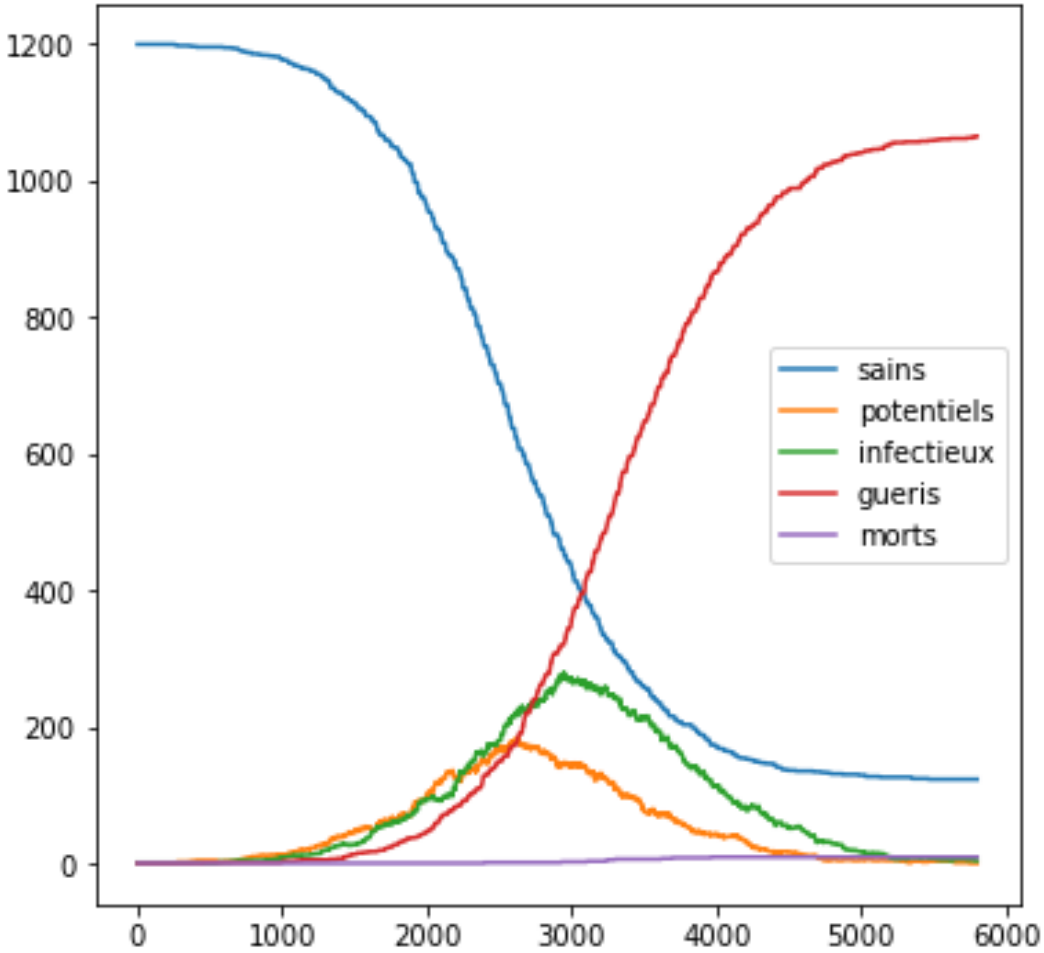
# Simulation des agents

Communauté de 1200 habitants sur un durée de 29 jours



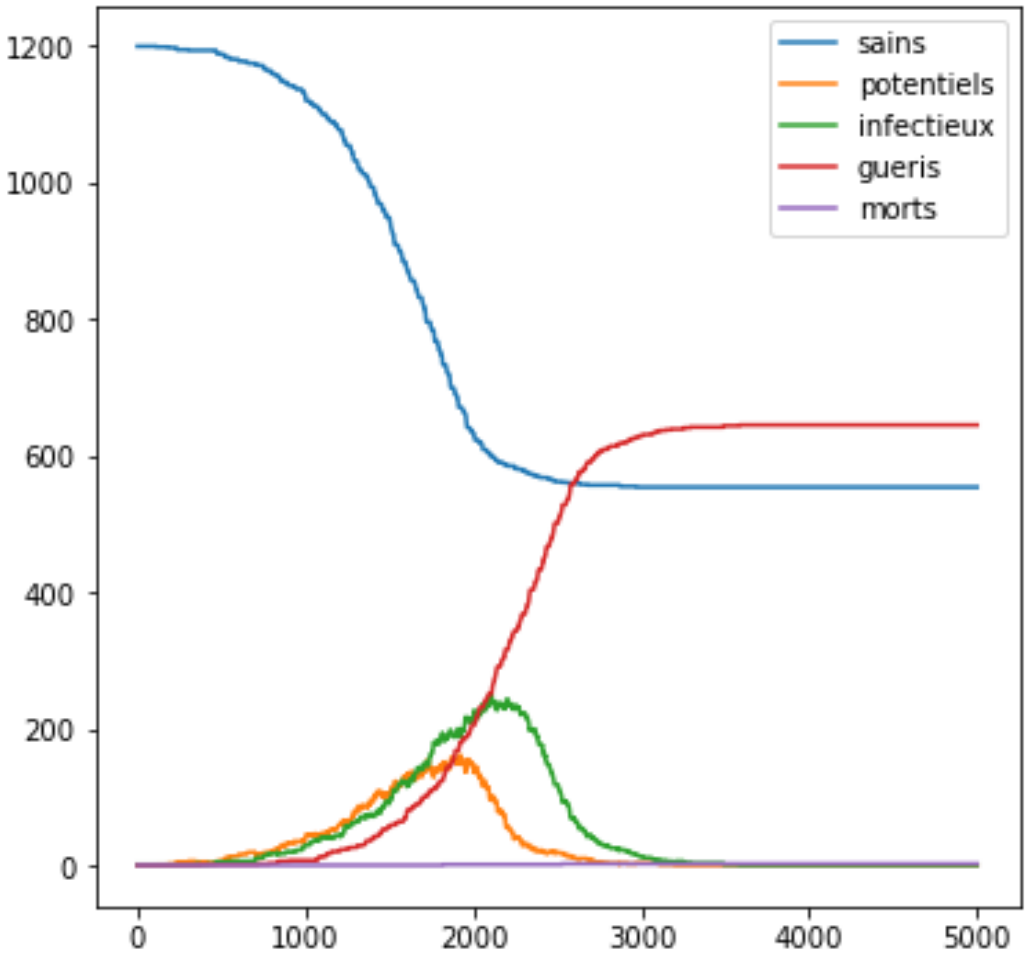

1131 personnes contaminées au total, 778 dans un bus, 145 à l'école, 168 sur le lieu de travail, 39 à domicile et 8 morts.

# Avec gestes barrières




1078 personnes contaminées au total, 657 dans le bus, 166 à l'école, 212 au travail, 42 à domicile et 9 morts.

# Avec tests




Les tests commencent au 42eme jour
647 personnes contaminées au total, 382 dans le bus, 139 à l'école, 73 au travail, 52 à domicile et 2 morts.

# Bilan

## Avantages

- précision des informations
- adaptabilité du modèle
- étude des aspects sociaux et spatiaux du problème
- permet d'évaluer l'efficacité de protocoles sanitaires précis et variés

## Inconvénients

- la complexité du modèle croît très vite en fonction de sa précision, de la population et de la période étudiée
- temps de calcul

# Modèle SIR

```
import matplotlib.pyplot as plt
from scipy import integrate
N = 2000. #nombre d'individus
t_remission=8
I0, R0 = 1, 0 #nombres initiaux d'infectés et de soignés
S0 = N - I0 - R0 #nombre de personnes susceptible
beta, gamma = 1/8, 1/t_remission #rythmes de contact et de rémission
tmax = 1000
Nt = 1000
t = np.linspace(0, tmax, Nt+1)
def derivative(X, t):
    S, I, R = X
    dotS = -beta * S * I /N
    dotI = beta * S * I/N  - gamma * I   #système d'equation des blocs
    dotR = gamma * I
    return np.array([dotS, dotI, dotR])
X0 = S0, I0, R0 #Initial conditions vector
res = integrate.odeint(derivative, X0, t) #résolution du système
S, I, R = res.T
plt.figure()
plt.grid()
plt.plot(t, S, 'orange', label='Susceptibles')
plt.plot(t, I, 'r', label='Infectés')
plt.plot(t, R, 'g', label='guéris')
plt.xlabel('jours')
plt.ylabel('population')
plt.ylim([0,N])
plt.legend()
```

# Modèle SEIR

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
#u=0 distanciation sociale (0-1)
        #0 = pas de distanciation
        #0.1 = masques
        #0.2 = classes en distenciel
        #0.3= teletravail
t_gueris=8
t_incub=5
R0=3
N= 1200 #population
#conditions initiales
e0=1/N #prop de pop infecté
i0=0 #infectieux
r0=0 #retablis
f0=0 #morts
s0= 1-e0-i0-r0 #prop de gens susceptible
x0=[s0,e0,i0,r0,f0]
a=1/t_incub   #infecté->infectieux
g=1/t_gueris  #infecteux->retablis
b=R0*g        #susceptible->infectés(non infectieux)
m=0.007     #infecté->mort
```

```
def maladie(x,t):
    s,e,i,r,f=x
    if 0<t<200: #pors du masque entre le 0 et le 200ème jour
        u=0.2
    else:
        u=0
    dx=np.zeros(5)
    dx[0]=-(1-u)*b*s*i
    dx[1]= (1-u)*b*s*i-a*e
    dx[2]= a*e-g*i-m*i
    dx[3]=g*i
    dx[4]=m*i
    return dx
t=np.linspace(0,200,200)
p=odeint(maladie, x0,t)
s=p[:,0]; e=p[:,1]; i=p[:,2]; r=p[:,3]; f=p[:,4]
plt.plot(t,s,label='susceptible')
plt.plot(t,r,label='retablis')
plt.plot(t,e,label='potentiel')
plt.plot(t,i,label='infectieux')
plt.plot(t,f,label='morts')
plt.legend()
plt.show()
#print(p)
```

# SMA basique

```python
import pylab
from mesa import Agent, Model
from mesa.space import MultiGrid
from mesa.time import StagedActivation
from random import random
from mesa.datacollection import DataCollector


class individus(Agent): #définition des agents
    def __init__(self,_id_,model,healtH):
        super().__init__(_id_,model)
        self.t_incub=0#temps d'incubation et de guérison initialisés à 0
        self.t_infect=0
        self.health=healtH
        self.death=10
    def move(self): #fonction de mouvement dans l'espace
        possible_steps=self.model.grid.get_neighborhood(self.pos,moore=True,include_center
        new_position=self.random.choice(possible_steps)
        self.model.grid.move_agent(self,new_position)
    def etat(self):#fonction régissant l'état de santé de l'individus
        if self.health=="infecte":
            if self.t_incub<5:
                self.t_incub+=1/48 #1 pas toutes les demi-heures
            else:
                self.health="infectieux"
                self.model.infectieux+=1
                self.model.infecte-=1
                if random()<0.007:
                    self.death=8*random()
        elif self.health=="infectieux":
            if self.death<=self.t_infect:
                self.health="morts"
```

```
                self.health="morts"
                self.model.grid.remove_agent(self)
                self.model.schedul.remove(self)
                self.model.infectieux-=1
                self.model.morts+=1
            elif self.t_infect<8:
                self.t_infect+=1/24
            else:
                    self.health="gueris"
                    self.model.gueris+=1
                    self.model.infectieux-=1
    def contamination(self):#fonction permettant à un agent d'en contaminer un autre
        if self.health=="infectieux":
            cellmates=[(self.model.grid.get_cell_list_contents(i))
            for i in (self.model.grid.get_neighborhood
                      (self.pos,moore=True,include_center=True))]
            for j in cellmates:
                for a in j:
                    if a.health=="sains":
                        if random()<=3/384:
                            a.health="infecte"
                            self.model.infecte+=1
                            self.model.sains-=1



class environnement(Model): #definition de l'environnement
    def __init__(self,N,m,width,height):
        self.infectieux=m
        self.sains=N
        self.infecte=0
        self.agent=[]       #paramètres de l'environnement
```

```
class environnement(Model): #definition de l'environnement
    def __init__(self,N,m,width,height):
        self.infectieux=m
        self.sains=N
        self.infecte=0
        self.agent=[]       #paramètres de l'environnement
        self.gueris=0
        self.heure=0
        self.morts=0
        self.date=0
        self.grid = MultiGrid(width, height, False)#espace de l'environnement
        stages=["move","contamination","etat"]
        self.schedul= StagedActivation(self,stages,True)#stockage des agents
        for i in range((self.sains)): #création des agents sains
            a=individus(i,self,"sains")
            self.schedul.add(a)
            q=self.random.randrange(self.grid.width)
            p=self.random.randrange(self.grid.height)
            self.grid.place_agent(a, (q,p)) #placer les agents dans l'espace
            self.agent.append(a) #stocker les agents dans le model

        for i in range((self.sains),(self.sains)+(self.infectieux)):
            a=individus(i,self,"infectieux")
            self.schedul.add(a)
            q=self.random.randrange(self.grid.width)
            p=self.random.randrange(self.grid.height)
            self.grid.place_agent(a, (q,p))
            self.agent.append(a)

        self.dc=DataCollector(  #collecte de donnés pour la representation
            model_reporters={"sains":lambda m:m.sains,"infectieux":lambda m:m.infectieux,
                             "infecte":lambda m:m.infecte,"gueris":lambda m:m.gueris,"morts
            agent_reporters={"health":"health"})
```

```
        self.dc=DataCollector(  #collecte de donnés pour la representation
            model_reporters={"sains":lambda m:m.sains,"infectieux":lambda m:m.infectieux,
                             "infecte":lambda m:m.infecte,"gueris":lambda m:m.gueris,"morts
            agent_reporters={"health":"health"})

    def step(self): #ce que fera le modèle à chaque pas
        if self.heure==0:
            self.date+=1
        self.heure=(self.heure+1/2)%24
        self.dc.collect(self)
        self.schedul.step()
```

```
m=environnement(1200,1,30,40)

import matplotlib.pyplot as plt
x=[]
y=[]
y2=[]
y3=[]
y4=[]
y5=[]
for i in range(1500):
    m.step()
    x.append(i)
    y.append(m.sains)
    y2.append(m.infecte)
    y3.append(m.infectieux)
    y4.append(m.gueris)
    y5.append(m.morts)
pylab.figure(figsize=(5,5))
plt.plot(x,y, label="sains")
plt.plot(x,y2, label="infecte")
plt.plot(x,y3, label="infectieux")
plt.plot(x,y4, label="gueris")
plt.plot(x,y5, label="morts")
plt.legend()
plt.show()
```

# Modèle complexe

```python
import pylab
from mesa import Agent, Model
from mesa.space import MultiGrid
from mesa.time import StagedActivation
from mesa.time import RandomActivation
from random import random
import random as rd
from mesa.datacollection import DataCollector


class individus(Agent):
    def __init__(self,model,_id_,function,health):
        super().__init__(_id_,model)
        self.t_incub=0
        self.t_infect=0
        self.bus_number=-1#bus dans-lequel l'agent se trouve
        self.function=function #rôle sociale (étudiant ou travailleur)
        self.health=health
        self.positif=False#tester positif
        self.tc=0#temps en confinement
        self.barriere=0#efficacité des gestes barrières
        self.death=100
        self._id_=_id_
        self.position=0#lieu où se trouve l'agent
        if self.function!="mineur":
            self.foyer=(self._id_-self.model.mineurs)//2#numero de la maison
        else:
```

```
        if self.function!="mineur":
            self.foyer=(self._id_-self.model.mineurs)//2#numero de la maison
        else:
            self.foyer=(self._id_//2)
    def etat(self):
        if self.positif and self.model.heure==1:
            if self.tc<7:#temps restant à passer en confinement
                self.tc+=1
            else:
                self.positif=False
        if self.health=="infecte":
            if self.t_incub<5:
                self.t_incub+=1/48 #1 pas toutes les demi heures
            else:
                self.health="infectieux"
                self.model.infectieux+=1
                self.model.infecte-=1
                if random()<=0.007:
                    self.death=8*random()
        elif self.health=="infectieux":
            if self.t_infect<8:
                self.t_infect+=1/48
            elif self.death<=self.t_infect:
                self.health="mort"
                self.model.infectieux-=1
                self.model.morts+=1
                self.model.popu.remove(self._id_)
                self.model.dead_agents.append((self._id_,self.model.date,self.model.heure))
            else:
                self.health="gueris"
                self.model.gueris+=1
```

```
                self.health="gueris"
                self.model.gueris+=1
                self.model.infectieux-=1
    def contamination(self,grille):
        if self.health=="infectieux":
            cellmates=[(grille.get_cell_list_contents(i)) for i in (grille.get_neighborhood(
            for j in cellmates:
                for a in j:
                    if a.health=="sains":
                        if random()<=(1-self.barriere)*3/384:
                            a.health="infecte"
                            self.model.infecte+=1
                            self.model.sains-=1
                            self.model.contaminé_tot+=1
    def bus_move(self):#mouvement des agents dans les bus
            possible_steps=self.model.bus[self.bus_number][0].get_neighborhood(self.pos,moor
            new_position=self.random.choice(possible_steps)
            self.model.bus[self.bus_number][0].move_agent(self,new_position)
    def bus_contamination(self):#contamination des agents dans le bus
        x=self.model.contaminé_tot
        self.contamination(self.model.bus[self.bus_number][0])
        self.model.bus_contagion+=self.model.contaminé_tot-x
```

```
    def bus(self):#comportement des agents dans le bus
        destination=self.model.bus_pos[self.bus_number %4]#là où le bus vas
        if self.health!="mort":
            if destination==1 and self.function=="mineur":#les étudiants vont à l'école
                self.model.classes[(self._id_)//40][1].add(self)#répartition dans les classe
                x = rd.randrange(7)
                y = rd.randrange(10)
                self.model.bus[self.bus_number][1].remove(self)
                self.model.bus[self.bus_number][0].remove_agent(self)
                self.model.classes[(self._id_)//40][0].place_agent(self,(x,y))
                self.model.classes[(self._id_)//40][0].move_to_empty(self)
                self.position=1
            elif destination==2 and self.function=="adulte":
                self.model.usine[1].add(self)
                x = rd.randrange(126)
                y = rd.randrange(161)
                self.model.bus[self.bus_number][1].remove(self)
                self.model.bus[self.bus_number][0].remove_agent(self)
                self.model.usine[0].place_agent(self,(x,y))
                self.position=2
            elif destination==0:
                self.model.bus[self.bus_number][1].remove(self)
                self.model.bus[self.bus_number][0].remove_agent(self)
                self.model.menages[self.foyer][1].add(self)
                x = rd.randrange(5)
                y = rd.randrange(10)
                self.model.menages[self.foyer][0].place_agent(self,(x,y))
                self.position=0
    def famille_move(self):
```

```
    def famille_move(self):
            possible_steps=self.model.menages[self.foyer][0].get_neighborhood(self.pos,moore
            new_position=self.random.choice(possible_steps)
            self.model.menages[self.foyer][0].move_agent(self,new_position)
    def famille_contamination(self):
        if self.health!="mort":
            x=self.model.contaminé_tot
            self.contamination(self.model.menages[self.foyer][0])
            self.model.famille_contagion+=self.model.contaminé_tot-x
    def school(self):
        if self.health!="mort":
            x=self.model.contaminé_tot
            self.contamination(self.model.classes[(self._id_)//40][0])
            self.model.ecole_contagion+=self.model.contaminé_tot-x
    def usine_move(self):
            possible_steps=self.model.usine[0].get_neighborhood(self.pos,moore=True,include_
            new_position=self.random.choice(possible_steps)
            self.model.usine[0].move_agent(self,new_position)
    def usine_contamination(self):
        if self.health!="mort":
            q=self.model.contaminé_tot
            self.contamination(self.model.usine[0])
            self.model.usine_contagion+=(self.model.contaminé_tot-q)
    def arret_bus(self):
```

```
    def arret_bus(self):
        if self.health!="mort":
            for i in range(4):
                if self.model.bus_pos[i]==self.position:
                    u=i
            if self.model.bus[u][1].get_agent_count()<60:
                m=u
            elif self.model.bus[u+4][1].get_agent_count()<60:
                m=u+4
            else:
                m=u+8
            self.bus_number=m
            self.model.arret_bus[self.position].remove(self)
            self.model.bus[m][1].add(self)
            x =rd.randrange(4)
            y =rd.randrange(14)
            self.model.bus[m][0].place_agent(self,(x,y))


class ville(Model):
    def __init__(self,population,nb_malades):
        self.running=True
```

```
class ville(Model):
    def __init__(self,population,nb_malades):
        self.running=True
        self.schedule=RandomActivation(self)
        self.pas=0
        self.bus_contagion=0
        self.ecole_contagion=0  #paramètres de mesure du modèle
        self.famille_contagion=0
        self.usine_contagion=0
        self.sains=population-nb_malades
        self.infectieux=nb_malades
        self.infecte=0
        self.gueris=0
        self.morts=0
        self.contaminé_tot=nb_malades
        self.liste_matin=[]
        self.liste_aprem=[]
        self.popu=[i for i in range(population)]
        malades=[rd.randrange(population+1) for i in range(nb_malades)]
        self.agents=[]
        self.dead_agents=[]
        self.mineurs=2*int(population*0.09)
        self.adultes=[i for i in range(self.mineurs,population)]
        self.famille=[(i,i+1,i+self.mineurs,i+self.mineurs+1)
        for i in range(0,self.mineurs,2)]+[(i,i+1) for i in range(2*self.mineurs,population,
        self.nb_familles=len(self.famille)
        self.nb_classes=((self.mineurs)//40)+1
        self.date=0
        self.heure=0
```

```
        self.nb_classes=((self.mineurs)//40)+1
        self.date=0
        self.heure=0
        self.sains=population
        arrets_stage=["etat","arret_bus"]
        self.arret_bus=[ StagedActivation(self,arrets_stage) for i in range(4)]
        self.destinations=["quartier_res","ecole","usine","hospitale"]
        self.bus_pos=[0,1,2,3]
        school_stages=["etat","school"]
        usine_stage=["etat","usine_move","usine_contamination"]
        bus_stages=["etat","bus_move","bus_contamination","bus"]
        famille_stage=["etat","famille_move","famille_contamination"]
        self.bus=[(MultiGrid(4,14,False),StagedActivation(self,bus_stages,True)) for i in ra
        self.menages=[(MultiGrid(6,11,False),StagedActivation(self,famille_stage,True))
                      for i in range(self.nb_familles)]
        self.classes=[(MultiGrid(8,11,False),StagedActivation(self,school_stages,True))
                      for i in range((self.mineurs//40)+1)]
        self.usine=(MultiGrid(126,161,False),StagedActivation(self,usine_stage,True))#(126,1
        for i in range(self.mineurs):
            if i in malades:
                santé="infectieux"
            else:
                santé="sains"
            q=individus(self,i,"mineur",santé)
            self.menages[q.foyer][1].add(q)
            x = self.random.randrange(5)
            y = self.random.randrange(10)
            self.menages[q.foyer][0].place_agent(q,(x,y))
            self.agents.append(q)
```

```
            q=individus(self,i,"mineur",santé)
            self.menages[q.foyer][1].add(q)
            x = self.random.randrange(5)
            y = self.random.randrange(10)
            self.menages[q.foyer][0].place_agent(q,(x,y))
            self.agents.append(q)
            self.schedule.add(q)
        for i in range(self.mineurs,population):
            if i in malades:
                santé= "infectieux"
            else:
                santé="sains"
            q=individus(self,i,"adulte",santé)
            self.menages[q.foyer][1].add(q)
            x = self.random.randrange(5)
            y = self.random.randrange(10)
            self.menages[q.foyer][0].place_agent(q,(x,y))
            self.agents.append(q)
            self.schedule.add(q)
        self.dc=DataCollector(
            model_reporters={"sains":lambda m:m.sains,"infectieux":lambda m:m.infectieux,
            "infecte":lambda m:m.infecte,"gueris":lambda m:m.gueris,"morts":lambda m:m.morts
            "date": lambda m:m.date, "heure": lambda m:m.heure, "contaminés tot": lambda m:m
            "bus contamination": lambda m:m.bus_contagion, "ecole contamination": lambda m:m
            "usine contamination": lambda m:m.usine_contagion,"famille contamination": lambd
            agent_reporters={"health":"health"})

    def step(self):
        self.dc.collect(self)
```

```
def step(self):#déroulement de la journée
    self.dc.collect(self)
    self.pas+=1
    if self.heure==0:
        self.date+=1
        self.liste_matin=[i for i in self.popu]#ceux qui prennent le bus le matin
        self.liste_aprem=(self.liste_matin).copy()
        rd.shuffle(self.liste_matin)#ceux qui le prennent l'aprem
        rd.shuffle(self.liste_aprem)
    self.heure=(self.heure+1/2)%24
    if 0<=self.heure<7:
        for i in self.agents:
            i.etat()
    if 7<=self.heure<=10:
        if len(self.liste_matin)>=180:
            a=180
        else:
            a=len(self.liste_matin)
        for i in range(a):
            x=self.liste_matin.pop()
            agent=self.agents[x]#agent d'id x
            if self.date>=42 and agent.health=="infectieux" and random()<0.6:
                agent.positif=True
            elif not agent.positif:
                if agent.function=="mineur":
                    maison=x//2
                else:
                        maison=(x-self.mineurs)//2
                self.menages[maison][0].remove_agent(agent)
```

```
            elif not agent.positif:
                if agent.function=="mineur":
                    maison=x//2
                else:
                        maison=(x-self.mineurs)//2
                self.menages[maison][0].remove_agent(agent)
                self.menages[maison][1].remove(agent)
                self.arret_bus[0].add(agent)

        for i in range(self.nb_familles):
            self.menages[i][1].step()

        for i in range(3):
           self.arret_bus[i].step()
        for i in range(self.nb_classes):
            self.classes[i][1].step()
        self.usine[1].step()

    if 10<self.heure<18:
        for i in range(self.nb_familles):
            self.menages[i][1].step()
        for i in range(self.nb_classes):
            self.classes[i][1].step()
        self.usine[1].step()
        for i in range(3):
           self.arret_bus[i].step()
```

```
        if 10<self.heure<18:
            for i in range(self.nb_familles):
                self.menages[i][1].step()
            for i in range(self.nb_classes):
                self.classes[i][1].step()
            self.usine[1].step()
            for i in range(3):
               self.arret_bus[i].step()

        if 18<=self.heure:
            if len(self.liste_aprem)>=180:
                t=180
            else:
                t=len(self.liste_aprem)
            for i in range(t):
                x=self.liste_aprem.pop()
                agent=self.agents[x]#agent d'id x
                if agent.health!="mort" and not agent.positif:
                    if agent.function=="mineur":
                        self.classes[x//40][0].remove_agent(agent)
                        self.classes[x//40][1].remove(agent)
                        self.arret_bus[1].add(agent)
                    else:
                        self.usine[0].remove_agent(agent)
                        self.usine[1].remove(agent)
```

```
                    self.arret_bus[1].add(agent)
                else:
                    self.usine[0].remove_agent(agent)
                    self.usine[1].remove(agent)
                    self.arret_bus[2].add(agent)

        self.arret_bus[1].step()
        self.arret_bus[2].step()
        for i in range(self.nb_classes):
            self.classes[i][1].step()
        self.usine[1].step()
        for i in range(self.nb_familles):
            self.menages[i][1].step()
    [x,y,z,w]=self.bus_pos
    self.bus_pos=[y,z,w,x]
    for i in range(12):
            self.bus[i][1].step()
```

```
m=ville(1200,1)
import matplotlib.pyplot as plt
x=[]
y=[]
y2=[]
y3=[]
y4=[]
y5=[]
for i in range(5000):
    m.step()
    x.append(i)
    y.append(m.sains)
    y2.append(m.infecte)
    y3.append(m.infectieux)
    y4.append(m.gueris)
    y5.append(m.morts)
pylab.figure(figsize=(6,6))
plt.plot(x,y, label="sains")
plt.plot(x,y2, label="potentiels")
plt.plot(x,y3, label="infectieux")
plt.plot(x,y4, label="gueris")
plt.plot(x,y5, label="morts")
plt.legend()
plt.show()
print("contaminés tot: "+ str(m.contaminé_tot) +
      " bus contamination: "+ str(m.bus_contagion) +
      " ecole contamination :"+ str(m.ecole_contagion) +
      " usine contamination :"+ str(m.usine_contagion) +
      " famille contamination :"+ str(m.famille_contagion) +
      " morts :"+ str(1200-len(m.popu)))
```