Numéro de candidat : **36157**
Thème : Santé, prévention

Impact des outils numériques sur le contrôle d'une épidémie

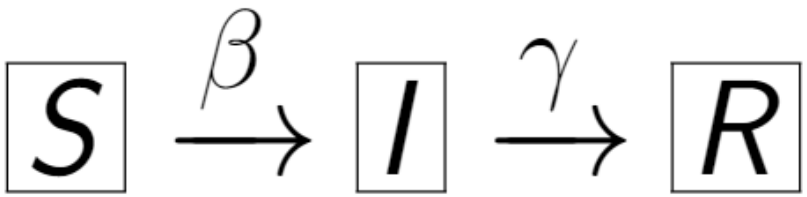
Quels outils numériques pouvons-nous proposer afin de restreindre la propagation d'un virus ?

Sommaire

1. Modélisation d’une pandémie grâce au modèle SIR
2. Simulation d’un phénomène de foule
3. Mise en relation de la simulation avec une base de données

I) Modélisation

Choix du modèle : **SIR**



$$\begin{cases} \dfrac{dS(t)}{dt} = -ßS(t)I(t) \\ \dfrac{dI(t)}{dt} = ßS(t)I(t) - \gamma I(t) \\ \dfrac{dR(t)}{dt} = \gamma I(t) \end{cases}$$

- ß : le taux d'infection
- γ : le taux de guérison

Résolution numérique grâce au langage Python

Code

```
# Dérivées au points 0
deriv_s = - beta * i * s
deriv_i = beta * i * s - gamma * i
deriv_r = gamma * i

delta = tf / points
```

```
# Méthode d'Euler
for k in range(1, points + 1):
    print(k)
    x = x + delta

    # Calcul approximatif des images s, i et r, connaissant les dérivées
    s = s + delta * deriv_s
    i = i + delta * deriv_i
    r = r + delta * deriv_r
    t.append(x)
    y.append([s, i, r])

    # Dérivées aux points de coordonnées (x, s), (x, i) et (x, r)
    deriv_s = - beta * i * s
    deriv_i = beta * i * s - gamma * i
    deriv_r = gamma * i
```

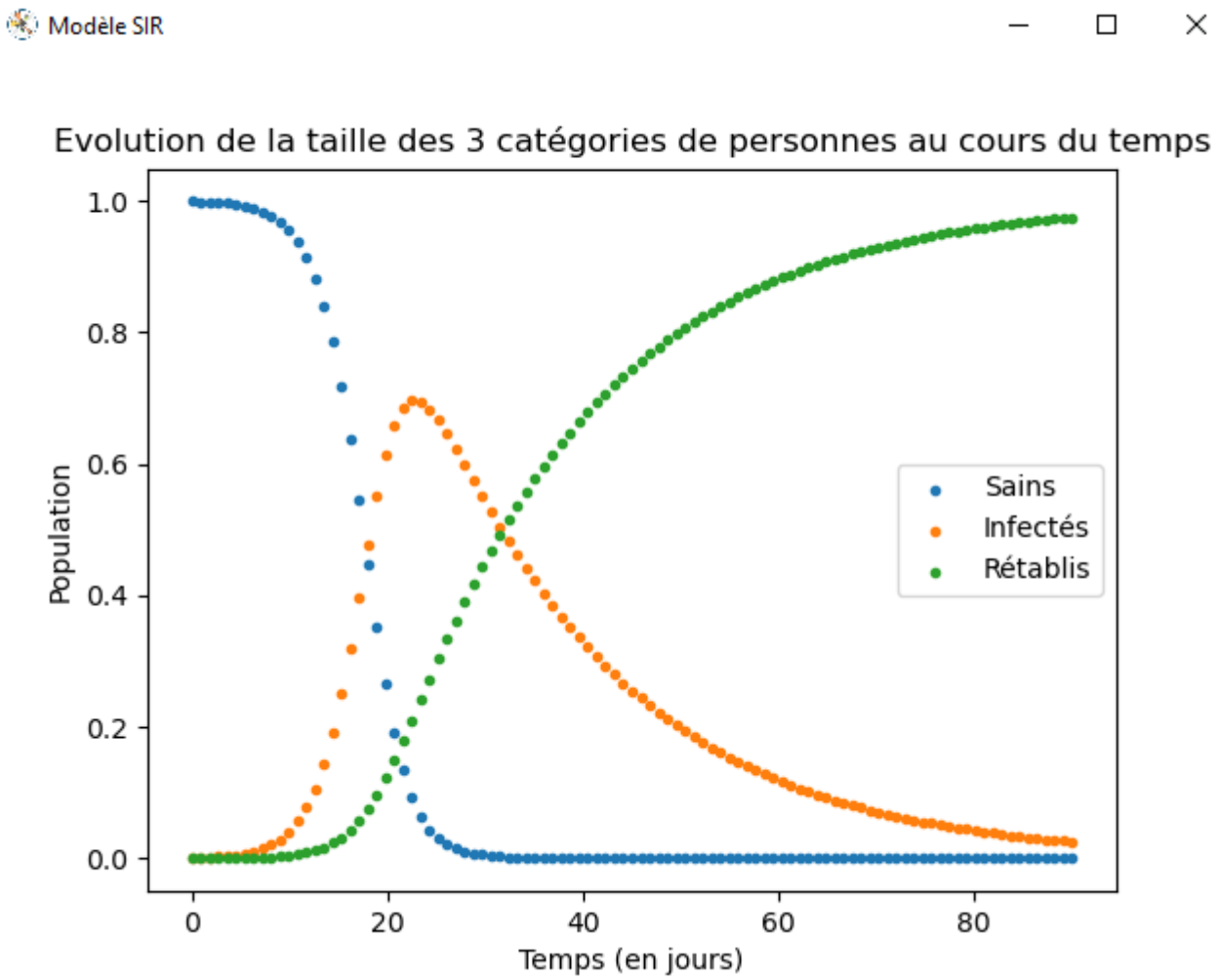
Résolution numérique grâce à la méthode d'Euler

Conditions initiales :

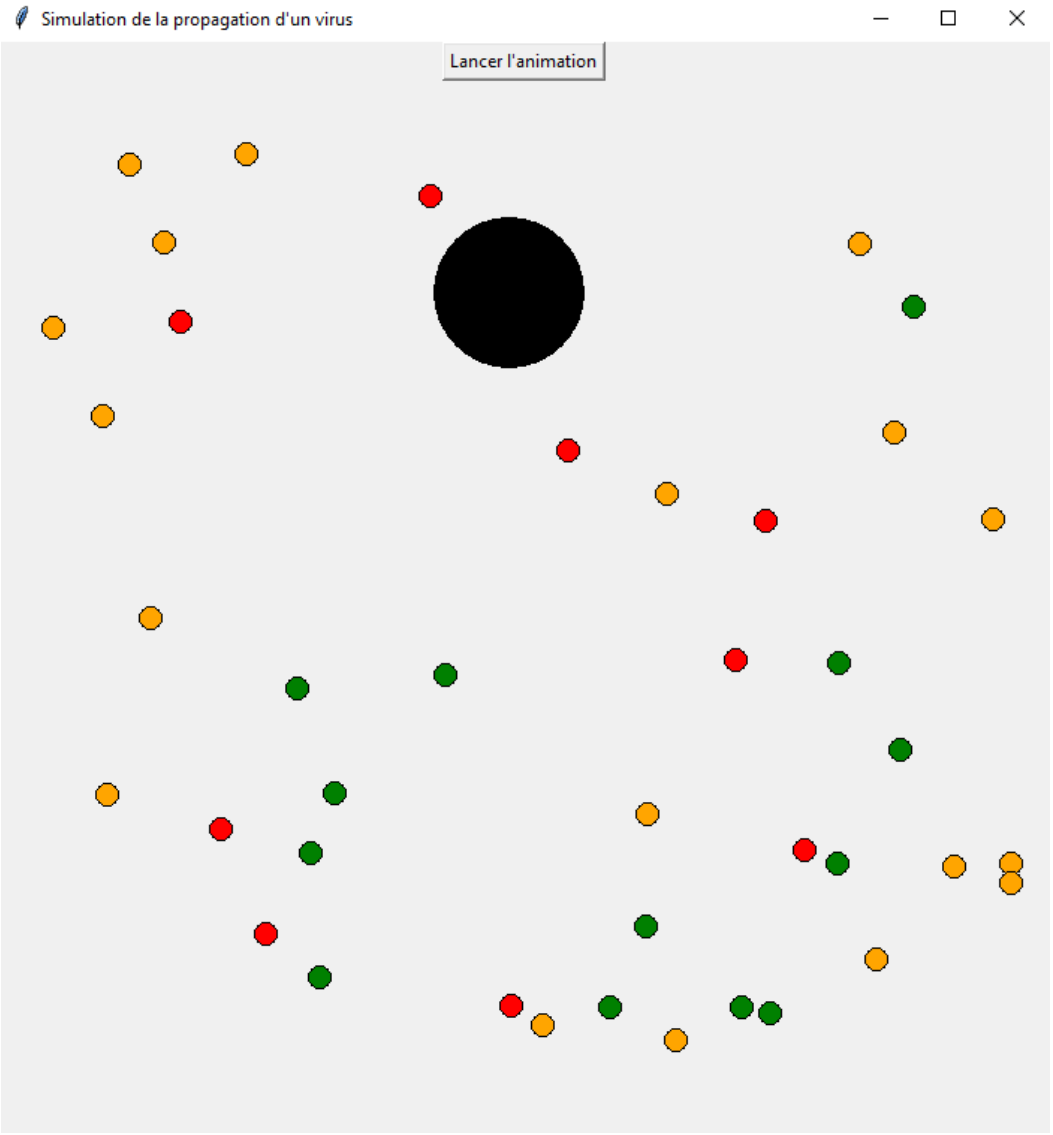
$$\begin{cases} n = 1000 \\ I_0 = 1/n \\ R_0 = 0 \\ S_0 = 999/n \end{cases}$$

Paramètres :

$$\begin{cases} ß = 0{,}5 \\ \gamma = 0{,}05 \end{cases}$$



II) Simulation d'un phénomène de foule



> **Quelques définitions relatives à la simulation :**

- Point attracteur
- Contact

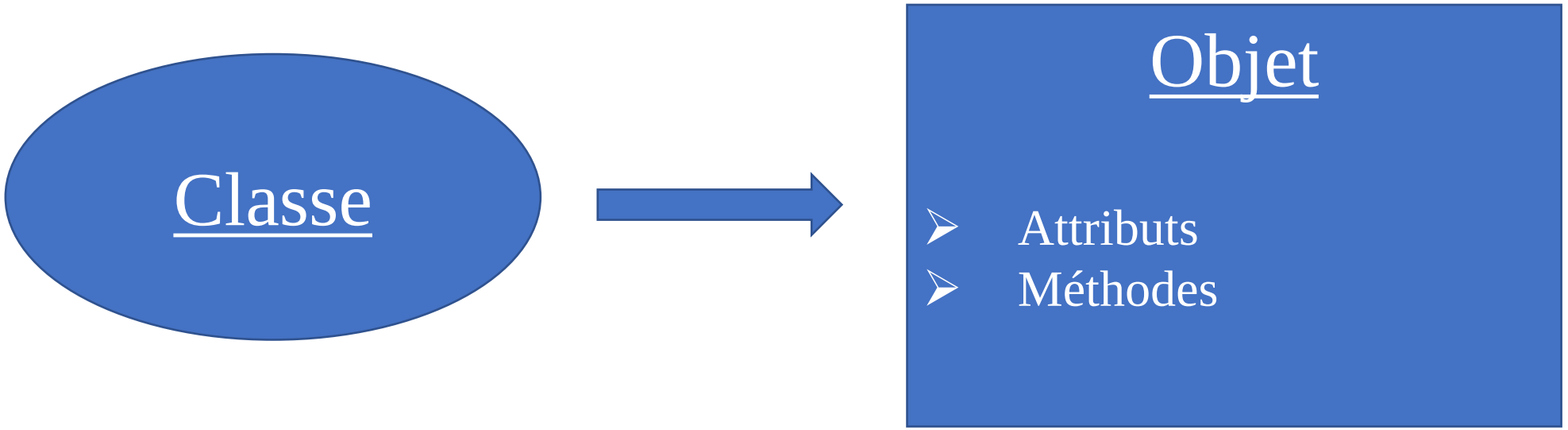
```
stantard_contact = {
    "distance": 1,
    "durée": 5,
    "beta": 1
}
```

> **Comment définir cette fenêtre ?**

```
sim = Simulation(height, width, stantard_contact, point_data, scale, db)
sim.put_points(n)
sim.display()
```

Qu’est ce que la Programmation Orientée Objet ?

- Un paradigme de la programmation informatique
- Objets (concept) : structure interne / mise en relation avec d’autres objets
- Classes
- Attributs
- Méthodes



Exemple : quelques éléments définissant la classe *Simulation*

```
class Simulation:
    def __init__(self, height, width, strd_contact, point_data, scale, db):
      self.standard_contact = strd_contact
      self.point_data = point_data
      self.attractor_point = None
      self.points = []
      self.vectors = []



      self.contacts = []
      self.scale = scale

   def add_contact(self, point1_id, point2_id):
     self.contacts.append((point1_id, point2_id))

   def contact_exist(self, point1_id, point2_id):
     for couple in self.contacts:
         if couple == (point1_id, point2_id) or couple == (point2_id, point1_id):
             return True
     return False
```

Attributs

Méthodes

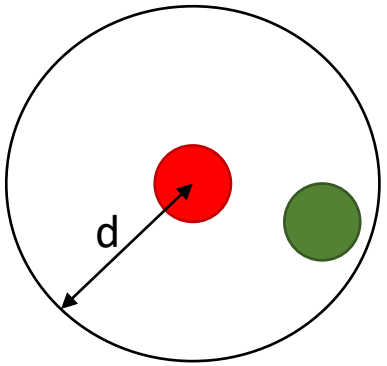
Que se passe-t-il lorsqu'on lance la simulation ?

```
if not point.is_on_point(self.attractor_point):
    point.move(vector)
    all_point_on_attractor = False
```

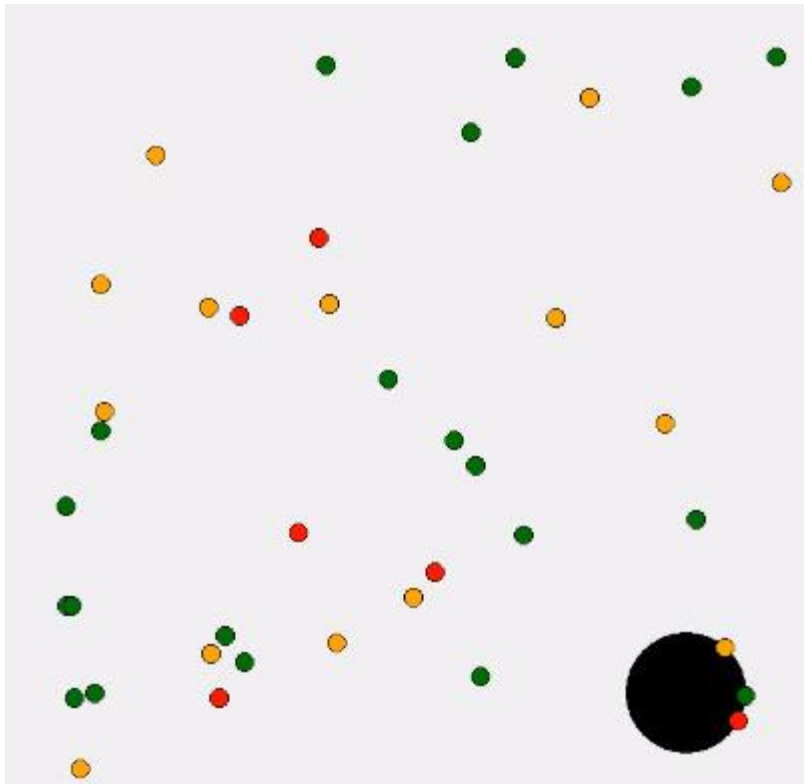
Si le point considéré n'est pas sur le point attracteur, le déplacer dans sa direction

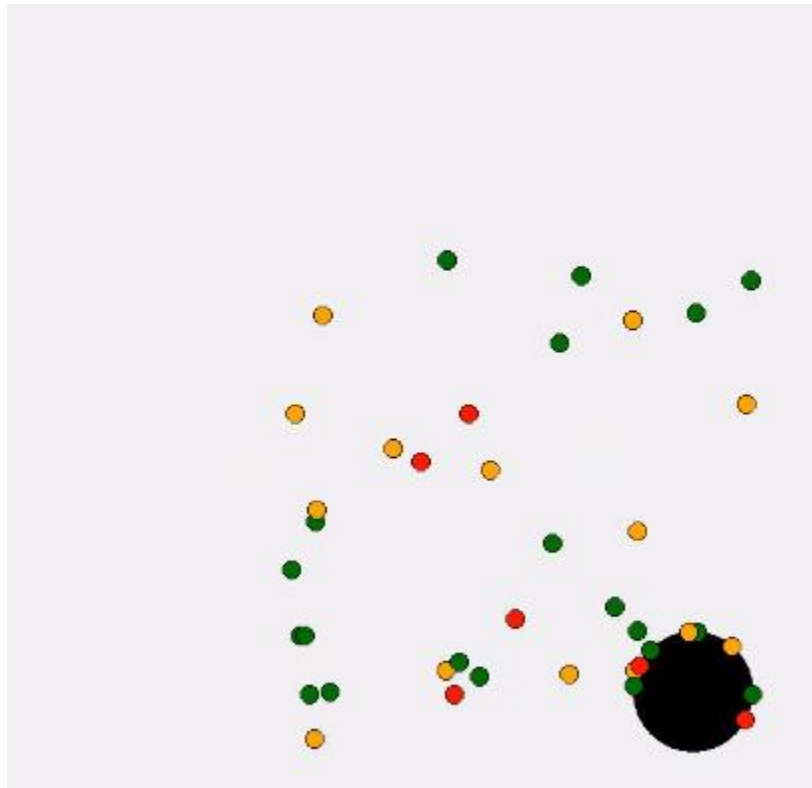
```
if point.is_contaminated():
    for p in self.points:
        condition1 = p.is_in_ball(point, self.standard_contact["distance"] * self.scale)
        condition2 = p != point
        condition3 = not self.contact_exist(point.id, p.id)
        if condition1 and condition2 and condition3:
            contamination = p.contaminate(self.standard_contact["beta"])
            self.add_contact(p.id, point.id)
```

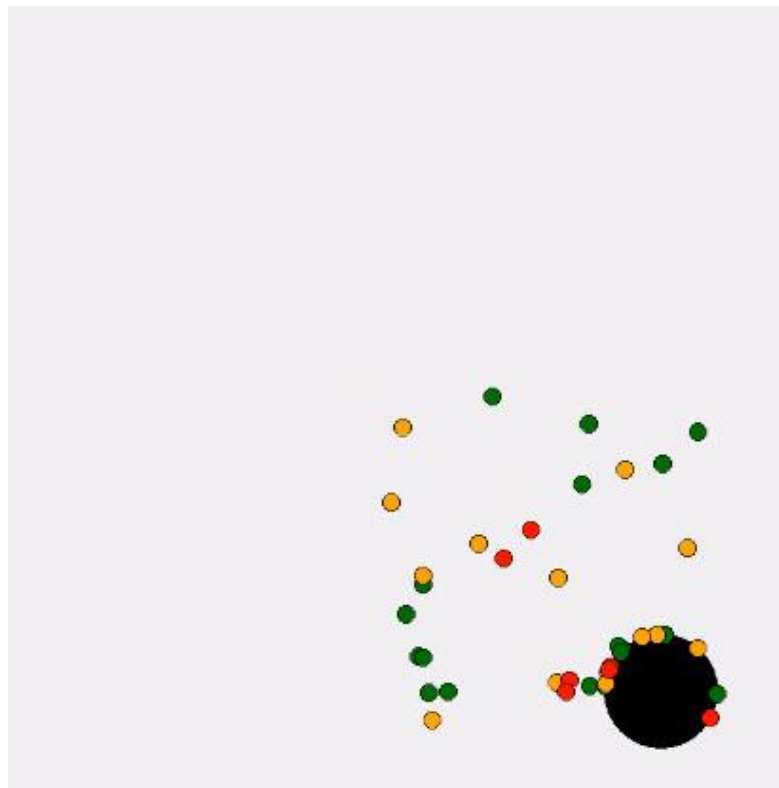
Contact



Illustrations







III) Mise en relation de la simulation avec une BDD

Objectif : pouvoir déterminer les tranches de la population les plus exposées aux risques.

- Données de santé pour chaque individu
- 3 types de données de santé :
- Celles à ***caractère personnel***
- Celles relatives aux ***facteurs extrinsèques environnementaux***
- Celles de ***contextualisation de la santé***

Différentes tables :

utilisateurs	***donnees_de_sante_categories***	***donnees_de_sante***	**utilisateurs_donnees_de_sante**	***contact***
- id - etat	- id - nom	- id - nom - categorie	- utilisateur_id - donnee_id - valeur - info_supplementaire	- id - utilisateur1_id - utilisateur2_id - contamination

Représentation d’une base de données (BDD) en Python

```
class Database:
    def __init__(self, host, user, password, database_name):
        self.connection = mysql.connector.connect(
            host=host,
            port=3306,
            user=user,
            database=database_name,
            password=password
        )

        self.cursor = self.connection.cursor()
```


Que représente cet attribut ?

Actions sur une base de données (méthodes de la classe *Database)*

1) Modifier une table

```
def update(self, table_name, attribute, new_value, condition):
    sql = "UPDATE " + table_name + " SET " + attribute + " = '" + new_value + "' WHERE " + condition
    self.get_cursor().execute(sql)
    self.connection.commit()

def insert(self, table_name, structure, values):
    sql = "INSERT INTO " + table_name + " " + structure + " VALUES " + str(values)
    self.get_cursor().execute(sql)
    self.connection.commit()
```

2) Récupérer des informations

```
def select(self, attributes, table_name, condition=None):
    if condition is not None:
        sql = "SELECT " + attributes + " FROM " + table_name + " WHERE " + condition
    else:
        sql = "SELECT " + attributes + " FROM " + table_name

    self.get_cursor().execute(sql)
    result = self.get_cursor().fetchall()
    return result
```

Création de nouvelles classes

1) *HealthData*

```
class HealthData:
    def __init__(self, user_id, db):
        self.user_id = user_id
        self.nb_types = len(db.select("id", "donnee_de_sante_categories"))
        self.data_health_types = [{} for _ in range(self.nb_types)]

        for i in range(self.nb_types):
            category_id = i + 1
            health_data = db.select("id", "données_de_sante", "categorie = " + str(category_id))
            for n_tuple in health_data:
                id = n_tuple[0]
                self.data_health_types[i][id] = None
```

Attributs générés **dynamiquement** grâce à la **structure des tables** relatives aux données de santé.

Que veut dire « générés dynamiquement » ? Quel est l'intérêt ?

> Table *donnees_de_sante_categories*

id	*nom*
1	Données de santé à caractère personnel
2	Facteurs extrinsèques
3	Données de contextualisation relative à la santé

> Table *donnees_de_sante*

id	*nom*	*categorie*
1	Antécédents médicaux	1
2	Maladie	1
3	Handicap	1
4	Environnement sain	2
5	Alcool	3
6	Tabac	3

Catégorie de données de santé n°1

```
{
    1: None,
    2: None,
    3: None,
}
```

Création de nouvelles classes

1) ***HealthData***
2) ***User***

```python
class User:
    def __init__(self, id, state):
        self.id = id
        self.state = state
        self.health_data = None

    def initialize_health_data(self, db):
        self.health_data = HealthData(self.id, db)
        self.health_data.generate()

    def insert_in_db(self, db: Database):
        db.insert("utilisateurs", self.get_db_attributes(), self.get_values())
```

Génération d'un objet de type *HealthData*

Insertion de l'individu dans la base de données

Création de nouvelles classes

1) ***HealthData***
2) ***User***
3) ***Contact***

```
class Contact:
    def __init__(self, users, contamination):
        self.users = users
        self.contamination = contamination

    def insert_in_db(self, db):
        db.insert("contact", self.get_db_attributes(), self.get_values())
```

Insertion du contact dans la base de données

A quoi correspond l'attribut *contamination* ?

Lien avec la simulation

1) Introduction d'une méthode de la classe *Point*

```
def create_user(self, db):
    if self.is_contaminated():
        self.user = User(self.id, "Infecté")
    elif self.is_healthy():
        self.user = User(self.id, "Sain")
    else:
        self.user = User(self.id, "Rétabli")

    self.user.initialize_health_data(db)

    # On ajoute l'utilisateur dans la BDD
    self.user.insert_in_db(db)
    # On y ajoute également ses données de santé
    self.user.get_health_data().insert_in_db(db)
```

2) S'il y a contact entre deux individus

```
contact = Contact((point.id, p.id), contamination)
contact.insert_in_db(self.db)
```

Résultats à la fin de la simulation

1) Extrait de la table *utilisateurs* (représentant les individus)

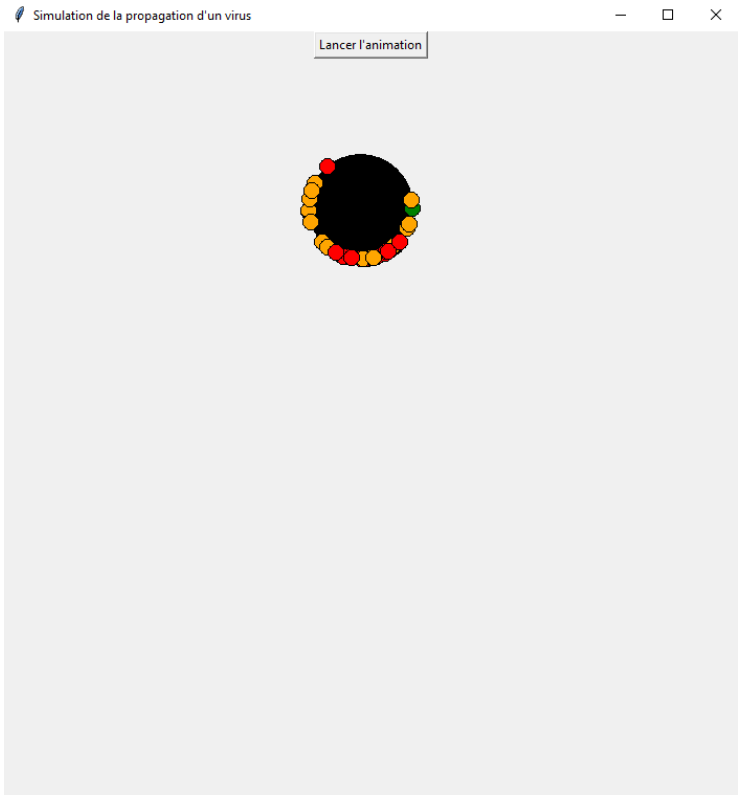
id	*etat*
2	Infecté
3	Infecté
4	Sain

2) Extrait de la table *donnees_de_sante_utilisateurs*

utilisateur_id	*donnee_id*	*valeur*
2	1	0
2	2	1
2	3	1
2	4	0
2	5	0

3) Extrait de la table *contacts* et rendu visuel

id	*utilisateur1_id*	*utilisateur2_id*	*contamination*
13	35	34	0
14	36	5	1
15	5	22	1



Conclusion, lien avec l’application TousAntiCovid

main.py

```python
from simulation.simulationc import Simulation
from server.database import Database


# -------------------- Paramètres -------------------- #

height = 700
width = 700
scale = 15  # 25 pixels sur le dessin représente 1 mètre
point_dimension = scale  # ODG Individu : 1m
n = 40  # Nombre d'individus à représenter

stantard_contact = {
    "distance": 1,  # distance inférieure ou égale à 1 mètre en cas de contact...
    "durée": 5,  # ... PENDANT une durée de 5 minutes
    "beta": 1  # Probabilité d'être infecté après avoir été en contact avec un individu infecté
}
point_data = {
    "diameter": point_dimension,
    "colors": ["green", "orange", "red"],
    "diameter_attractor": 100
}
```

```
23
24  db = Database("localhost", "root", "", "tipe")
25
26  # ---------------------- Simulation ---------------------- #
27
28  sim = Simulation(height, width, stantard_contact, point_data, scale, db)
29  sim.put_points(n)
30  sim.display()
```

point.py

```python
import math
from random import random
from server.user import User


class Point:
    """
    Cette classe modélise un individu par un cercle coloré.
    """
    def __init__(self, x, y, diameter, color, canvas):
        self.x = x
        self.y = y
        self.color = color
        self.canvas = canvas
        self.diameter = diameter
        self.id = None
        self.user = None

    def create_user(self, db):
        """
        Chaque point modélise un individu. On associe alors à chaque objet de type point un individu possédant des
        données de santé générées aléatoirement.
        :param db:
        :return:
        """
```

```
26          if self.is_contaminated():
27              self.user = User(self.id, "Infecté")
28          elif self.is_healthy():
29              self.user = User(self.id, "Sain")
30          else:
31              self.user = User(self.id, "Rétabli")
32
33          self.user.initialize_health_data(db)
34
35          # On ajoute l'utilisateur dans la BDD
36          self.user.insert_in_db(db)
37          # On y ajoute également ses données de santé
38          self.user.get_health_data().insert_in_db(db)
39
40      def is_contaminated(self):
41          return self.color == "red"
42
43      def is_healthy(self):
44          return self.color == "green"
45
46      def is_recovered(self):
47          return self.color == "orange"
48
```

```
49      def get_vector(self, point):
50          """
51          Retourne le vecteur directeur : il indique la direction et le sens de déplacement vers le point indiqué
52          en paramètre.
53          C'est un vecteur unitaire.
54          :return:
55          """
56          dx = point.x - self.x
57          dy = point.y - self.y
58          norm = self.distance(point)
59
60          return dx/norm, dy/norm
61
62      def distance(self, point):
63          """
64          Retourne la distance séparant le point indiqué en paramètre et CE point (désigné par l'objet self)
65          """
66          dx = point.x - self.x
67          dy = point.y - self.y
68          dist = math.sqrt(dx ** 2 + dy ** 2)
69          return dist
70
71      def is_in_ball(self, center, radius):
72          """
```

```
73          Vérifie si le point self est dans la boule définie par son centre et son rayon.
74          :param center:
75          :param radius:
76          :return:
77          """
78          return center.distance(self) <= radius
79
80      def is_on_point(self, point):
81          """
82          Vérifie si le point self est sur le point indiqué en paramètre.
83          :param point:
84          :return:
85          """
86          radius = point.get_diameter() / 2
87          return self.is_in_ball(point, radius)
88
89      def move(self, vector):
90          """
91          Translation du point selon le vecteur entré en paramètre
92          :param vector:
93          :return:
94          """
95          # On met à jour ses nouvelles coordonnées
```

```
96          self.x = self.x + vector[0]
97          self.y = self.y + vector[1]
98
99          # On déplace le point (dx <- vector[0] ; dy <- vector[1])
100         self.canvas.move(self.id, vector[0], vector[1])
101
102     def get_diameter(self):
103         """
104         Retourne le diamètre du cercle
105         :return:
106         """
107         return self.diameter
108
109     def get_color(self):
110         """
111         Retourne la couleur du point
112         :return:
113         """
114         return self.color
115
116     def get_x(self):
117         return self.x
118
```

```
119         def get_y(self):
120             return self.y
121 
122         def draw(self):
123             """
124             Dessine le disque représentant l'individu dans le canvas sélectionné
125             :return:
126             """
127             x0 = self.get_x() - self.get_diameter()/2
128             y0 = self.get_y() - self.get_diameter()/2
129             x1 = self.get_x() + self.get_diameter()/2
130             y1 = self.get_y() + self.get_diameter()/2
131 
132             self.id = self.canvas.create_oval(x0, y0, x1, y1, fill=self.get_color())
133             # self.canvas.create_text(x0, y0, text=self.id)
134 
135         def change_color(self, color):
136             self.color = color
137             self.canvas.itemconfig(self.id, fill=color)
138 
139         def contaminate(self, beta):
140             """
```

```
141         Cette fonction modélise la contamination du point.
142         Retourne 1 s'il y a eu contamination, 0 sinon.
143         :return:
144         """
145         # On ne contamine que les individus sains...
146         if self.is_healthy():
147             k = random()
148             # ... avec une probabilité beta
149             if k <= beta:
150                 self.change_color("red")
151                 print(str(self.id) + " a été contaminé !")
152                 return 1
153             else:
154                 print(str(self.id) + " n'a pas été contaminé !")
155                 return 0
156         else:
157             return 0
```

simulationc.py

```python
import tkinter as tk
import random
from simulation.point import Point
from server.contact import Contact


class Simulation:
    def __init__(self, height, width, strd_contact, point_data, scale, db):
        """
        Cette classe représente une fenêtre dans laquelle se déroulera la simulation.
        La simulation correspondra a une succession d'états permettant de mettre en évidence la transmission du virus :
            A un certain état n, chaque point sera caractérisé par sa couleur et par sa position.
            Le mouvement d'un point est réalisé grace au changement de ses coordonnées lors de la transition d'un état n
            à un état n+1.
        """

        # Paramètres de la fenêtre
        self.height = height
        self.width = width
        self.window = tk.Tk()
        self.window.title("Simulation de la propagation d'un virus")
        self.canvas = tk.Canvas(self.window, width=700, height=700)
        self.button = tk.Button(self.window, text="Lancer l'animation", command=self.run animation)
```

```
24          self.button.pack()
25
26          # Paramètres de la simulation
27          self.standard_contact = strd_contact  # Conditions nécessaires pour décrire un contact
28          self.point_data = point_data  # Informations relatives à un point (son diamètre, sa couleur...)
29          self.attractor_point = None
30          self.points = []
31          self.vectors = []  # Contient les vecteurs déplacement de chaque point de la simulation.
32                             # Il sont unitaires, dirigé et orienté vers le point attracteur.
33
34          self.contacts = []
35          self.scale = scale
36
37          # Base de données
38          self.db = db
39
40      def SIR_data(self):
41          s = 0
42          i = 0
43          r = 0
44          for point in self.points:
45              if point.is_contaminated():
46                  i = i + 1
```

```
47              elif point.is_healthy():
48                  s = s + 1
49              else:
50                  r = r + 1
51
52          return s, i, r
53
54      def generate_color(self):
55          """
56          Génère aléatoirement une couleur parmi le rouge le vert et le orange.
57          :return: une couleur (de type string)
58          """
59          colors = self.point_data["colors"]
60          return random.choice(colors)
61
62      def add_contact(self, point1_id, point2_id):
63          """
64          Ajoute un contact à l'ensemble des contacts qui ont lieu au cours de la simulation.
65          :param point1_id:
66          :param point2_id:
67          :return:
68          """
69          self.contacts.append((point1_id, point2_id))
```

```
70
71      def contact_exist(self, point1_id, point2_id):
72          """
73          Cette méthode vérifie si les points dont les id sont rentrés en paramètres ont été en contact.
74
75          -> En effet, si à un certain état n il n'y a pas eu de contamination entre 2 points, il ne peut pas y en avoir à
76          l'état n+1 d'où l'utilité de vérifier si un contact a déjà eu lieu.
77          :param point2_id:
78          :param point1_id:
79          :return: booléen
80          """
81          for couple in self.contacts:
82              if couple == (point1_id, point2_id) or couple == (point2_id, point1_id):
83                  return True
84          return False
85
86      def generate_coord(self):
87          """
88          Retourne des coordonnées aléatoires pour un point de la fenêtre (contraintes en fonction des dimensions
89          de la fenêtre)
90          :return: tuple sous la forme (x, y)
91          """
92          diameter = self.point_data["diameter"]  # Récupération du diamètre d'un point
93          x = int(random.uniform(diameter, self.width - diameter))
94          y = int(random.uniform(diameter, self.height - diameter))
```

```
95
96          return x, y
97
98      def create_attractor_point(self):
99          """
100         Positionne sur la fenêtre le point attracteur, ie le point qui génère des phénomènes de foule.
101         Exemple : un concert, une école, etc...
102         :return: None
103         """
104         (x0, y0) = self.generate_coord()
105         self.attractor_point = Point(x0, y0, self.point_data["diameter_attractor"], "black", self.canvas)
106         self.attractor_point.draw()
107
108     def put_points(self, n):
109         """
110         Positionne sur la fenêtre n points de coordonnées générées aléatoirement.
111         :param n:
112         :return: None
113         """
114         diameter = self.point_data["diameter"]
115         self.create_attractor_point()
116
117         for i in range(n):
118             # On génère de manière aléatoire des coordonnées
119             (x, y) = self.generate_coord()
```

```
120
121             # On crée le point pour ensuite le dessiner dans la fenêtre..
122             point = Point(x, y, diameter, self.generate_color(), self.canvas)
123             point.draw()
124             # ..Après l'avoir dessiné, on peut lui associer un utilisateur qui possedera un ensemble de données de santé
125             point.create_user(self.db)
126
127             # On met à jour les listes caractérisant la simulation
128             self.points.append(point)
129             self.vectors.append(point.get_vector(self.attractor_point))
130
131         print("----------------  Début de la simulation...  ----------------")
132
133     def run_animation(self):
134         """
135         Déplacer tous les points vers le point attracteur tant qu'ils n'y sont pas.
136         :return:
137         """
138         all_point_on_attractor = True
139         n = len(self.points)
140         for i in range(n):
141             point, vector = self.points[i], self.vectors[i]
142
143             # 1) Si le point d'indice i n'est pas dans la zone du point attracteur, le faire avancer dans sa direction.
144             if not point.is_on_point(self.attractor_point):
```

```
145                 point.move(vector)
146                 all_point_on_attractor = False
147 
148             # 2) Si le point d'indice i est rouge, mettre les points voisins en rouge sous certaines conditions...
149             if point.is_contaminated():
150                 for p in self.points:
151                     condition1 = p.is_in_ball(point, self.standard_contact["distance"] * self.scale)
152                     condition2 = p != point
153                     condition3 = not self.contact_exist(point.id, p.id)
154                     if condition1 and condition2 and condition3:
155                         contamination = p.contaminate(self.standard_contact["beta"])
156                         self.add_contact(p.id, point.id)
157 
158                         # On créé un objet permettant de symboliser le contact, pour ensuite l'enregistrer dans la BDD
159                         contact = Contact((point.id, p.id), contamination)
160                         contact.insert_in_db(self.db)
161 
162         # Continuer la simulation tant que tous les points ne sont pas vers le point attracteur
163         if not all_point_on_attractor:
164             self.canvas.after(10, self.run_animation)
165         else:
166             print("Il y a eu " + str(len(self.contacts)) + " contacts !")
167             print("----------------  Fin de la simulation !  ----------------")
168 
169     def display(self):
170         self.canvas.pack()
171         self.window.mainloop()
```

database.py

```python
import mysql.connector


class Database:
    def __init__(self, host, user, password, database_name):
        """
        Cette classe permet de gèrer les actions effectuées sur la base de données sélectionnée.

        :param host: Hôte du server MySQL
        :param user: Nom d'utilisateur
        :param password: Mot de passe
        :param database_name: Nom de la base de données
        """
        self.connection = mysql.connector.connect(
            host=host,
            port=3306,
            user=user,
            database=database_name,
            password=password
        )

        self.cursor = self.connection.cursor()

    def get_cursor(self):
        """
```

```
26          Le curseur est un objet permettant d'éxécuter des requêtes SQL.
27
28          :return: Le curseur de la BDD sur laquelle on est connecté.
29          """
30          return self.cursor
31
32      def update(self, table_name, attribute, new_value, condition):
33          """
34          Permet de mettre à jour des données.
35          :param table_name:
36          :param attribute:
37          :param new_value:
38          :param condition:
39          :return:
40          """
41          sql = "UPDATE " + table_name + " SET " + attribute + " = '" + new_value + "' WHERE " + condition
42          self.get_cursor().execute(sql)
43          self.connection.commit()
44
45      def insert(self, table_name, structure, values):
46          """
47          Permet d'enregistrer des valeurs dans la table table_name.
48
49          :param table_name: Nom de la table
```

```
50          :param structure: Structure de la table
51          :param values: Les attributs sous forme de liste ou de tuple
52          :return: None
53          """
54          sql = "INSERT INTO " + table_name + " " + structure + " VALUES " + str(values)
55          self.get_cursor().execute(sql)
56          self.connection.commit()
57
58      def select(self, attributes, table_name, condition=None):
59          """
60          Retourne une liste correspondant aux enregistrements retournés par la requête.
61          Cette liste contient des n-uplets si n attributs sont sélectionnés dans la requête.
62
63          :param condition:
64          :param table_name:
65          :param attributes:
66          :return: list
67          """
68          if condition is not None:
69              sql = "SELECT " + attributes + " FROM " + table_name + " WHERE " + condition
70          else:
71              sql = "SELECT " + attributes + " FROM " + table_name
72
73          self.get_cursor().execute(sql)
74          result = self.get_cursor().fetchall()
75          return result
```

health_data.py

```python
from server.database import Database
from random import uniform


class HealthData:
    def __init__(self, user_id, db):
        """
        Cette classe permet de générer pour tout individu un objet contenant les données de santé essentielles.
        Pour ce travail on se limite à 3 types de données de santé, enregistrés dans une bdd :
        - Le premier correspond aux données de santé à caractère personnel (antécédents médicaux, maladies, traitements,
          handicap, etc...)

        - Le second correspond à des données relatives aux facteurs extrinsèques environnementaux non-personnels de
        santé (qualité de l'environnement évaluée en fonction de celle de l'eau ou de l'air, contexte régional)

        - Le troisième correspond à des données de contextualisation de la santé relative à l'individu (IMC,
        alimentation, consommation d'alcool/tabac, catégorie socioprofessionnelle, l'appartenance à une classe scolaire)

        # Les objets issus de cette classe sont construits à partir de la structure de la BDD #
        """

        self.user_id = user_id
        self.nb_types = len(db.select("id", "donnee_de_sante_categories"))
        self.data_health_types = [{} for _ in range(self.nb_types)]

```

```
26          for i in range(self.nb_types):
27              category_id = i + 1
28              health_data = db.select("id", "données_de_sante", "categorie = " + str(category_id))
29              for n_tuple in health_data:
30                  id = n_tuple[0]
31                  self.data_health_types[i][id] = None
32
33      def generate(self):
34          """
35          TODO:
36          Cette fonction génère des valeurs correspondant au types 1, 2 et 3 de manière aléatoire
37          :return:
38          """
39          for data_health_type in self.data_health_types:
40              for id in data_health_type.keys():
41                  data_health_type[id] = round(uniform(0, 1))
42
43      def set_health_data(self, health_data_id, new_value):
44          """
45          Permet d'assigner une nouvelle valeur correspondant à la donnée de santé dont l'id est renseigné en paramètre.
46          :param health_data_id:
47          :param new_value:
48          :return:
49          """
50          for health_data_type in self.data_health_types:
```

```
            if health_data_id in health_data_type.keys():
                health_data_type[health_data_id] = new_value

    def set_extra_data(self, db, health_data_id, info):
        """
        Permet d'ajouter des informations supplémentaires à un utilisateur selon une certaine donnée de santé.
        :param db:
        :param health_data_id:
        :param info:
        :return:
        """
        table_name = "utilisateurs_données_de_sante"
        attribute = "info_supplementaire"
        new_value = info
        condition = "donnee_id = " + str(health_data_id) + " AND utilisateur_id = " + str(self.user_id)
        db.update(table_name, attribute, new_value, condition)

    @staticmethod
    def get_db_attributes():
        """
        Retourne la chaîne de caractère correspondant aux attributs de la table 'utilisateurs_données_de_sante'.

        :return: str
        """
        return "(utilisateur_id, donnee_id, valeur, info_supplementaire)"
```

```
76
77      def insert_in_db(self, db):
78          """
79          Cette méthode permet d'ajouter les données de santé enregistrées dans cet objet dans la table
80          utilisateurs_données_de_sante.
81          :param db:
82          :return:
83          """
84          table_name = "utilisateurs_données_de_sante"
85          structure = self.get_db_attributes()
86          for data_health_type in self.data_health_types:
87              for health_data_id, health_data_value in data_health_type.items():
88                  value = (self.user_id, health_data_id, health_data_value, "")
89                  db.insert(table_name, structure, value)
```

contact.py

```python
class Contact:
    def __init__(self, users, contamination):
        self.users = users
        self.contamination = contamination

    @staticmethod
    def get_db_attributes() -> str:
        """
        Retourne la chaîne de caractère correspondant aux attributs de la table 'users'.

        :return: str
        """
        return "(utilisateur1_id, utilisateur2_id, contamination)"

    def get_values(self):
        return self.get_users()[0], self.get_users()[1], self.contamination

    def get_users(self):
        return self.users

    def insert_in_db(self, db):
        db.insert("contact", self.get_db_attributes(), self.get_values())
```

user.py

```python
from server.database import Database
from server.health_data import HealthData


class User:
    def __init__(self, id, state):
        self.id = id
        self.state = state
        self.health_data = None

    @staticmethod
    def get_db_attributes() -> str:
        """
        Retourne la chaîne de caractère correspondant aux attributs de la table 'users'.

        :return: str
        """
        return "(id, etat)"

    def get_health_data(self):
        return self.health_data

    def initialize_health_data(self, db):
        """
        """
```

```
26          self.health_data = HealthData(self.id, db)
27          self.health_data.generate()
28
29      def get_values(self):
30          """
31          Retourne les valeurs des attributs correspondant à cet objet.
32
33          :return: tuple
34          """
35          return self.id, self.state
36
37      def insert_in_db(self, db: Database):
38          """
39          Permet d'insérer les données de cet utilisateur dans la base de données indiquée en paramètre.
40
41          :param db: Base de données
42          :return:
43          """
44          db.insert("utilisateurs", self.get_db_attributes(), self.get_values())
45          print("L'utilisateur " + str(self.id) + " a bien été enregistré dans la BDD !")
```

euler_method.py

```
1   import consts
2   import matplotlib.pyplot as plt
3
4   # Sains
5
6
7   def sir(points, tf, beta, gamma, y):
8
9       # CI
10      x = 0
11      s = y[0]
12      i = y[1]
13      r = y[2]
14
15      # Listes
16      t = [0]
17      y = [[s, i, r]]
18
19      # Dérivées au points 0
20      deriv_s = - beta * i * s
21      deriv_i = beta * i * s - gamma * i
22      deriv_r = gamma * i
```

```
23
24      delta = tf / points
25
26      # Méthode d'Euler
27      for k in range(1, points + 1):
28          print(k)
29          x = x + delta
30
31          # Calcul approximatif des images s, i et r, connaissant les dérivées
32          s = s + delta * deriv_s
33          i = i + delta * deriv_i
34          r = r + delta * deriv_r
35          t.append(x)
36          y.append([s, i, r])
37
38          # Dérivées aux points de coordonnées (x, s), (x, i) et (x, r)
39          deriv_s = - beta * i * s
40          deriv_i = beta * i * s - gamma * i
41          deriv_r = gamma * i
42
43      return t, y
44
45
```

```
46  sir_euler_method = sir(100, 90, consts.BETA, consts.GAMMA, [consts.S0, consts.I0, consts.R0])
47  t, y = sir_euler_method
48  s = [i[0] for i in y]
49  i = [i[1] for i in y]
50  r = [i[2] for i in y]
51
52  plt.figure("Modèle SIR")
53  plt.title("Evolution de la taille des 3 catégories de personnes au cours du temps")
54  plt.xlabel("Temps (en jours)")
55  plt.ylabel("Population")
56
57  plt.scatter(t, s, marker=".", label="Sains")
58  plt.scatter(t, i, marker=".", label="Infectés")
59  plt.scatter(t, r, marker=".", label="Rétablis")
60
61  plt.legend()
62  plt.show()
```

consts.py

```
1   # Conditions initiales sur la population
2
3   I0 = 1/1000
4   R0 = 0
5   S0 = 999/1000
6
7   # Les taux
8
9   # BETA correspond au taux d'infection / probabilité d'être infecté après avoir été en contact avec un individu infecté
10  BETA = 0.5
11
12  # GAMMA correspond à la probabilité de ne plus pouvoir transmettre le virus
13  GAMMA = 0.05
```