

## Impact des outils numériques sur le contrôle d'une épidémie

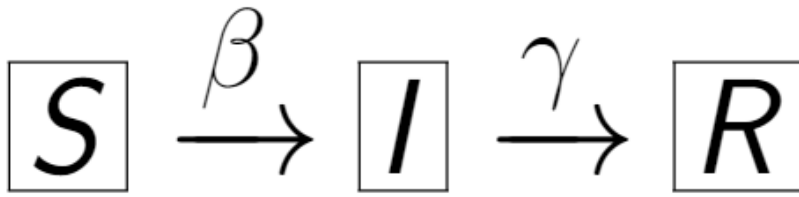
Quels outils numériques pouvons-nous proposer afin de restreindre la propagation d'un virus ?

# 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



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

- $\beta$  : le taux d'infection
- $\gamma$  : le taux de guérison

Résolution numérique grâce au langage Python

## Résolution numérique grâce à la méthode d'Euler

**Conditions initiales :**

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

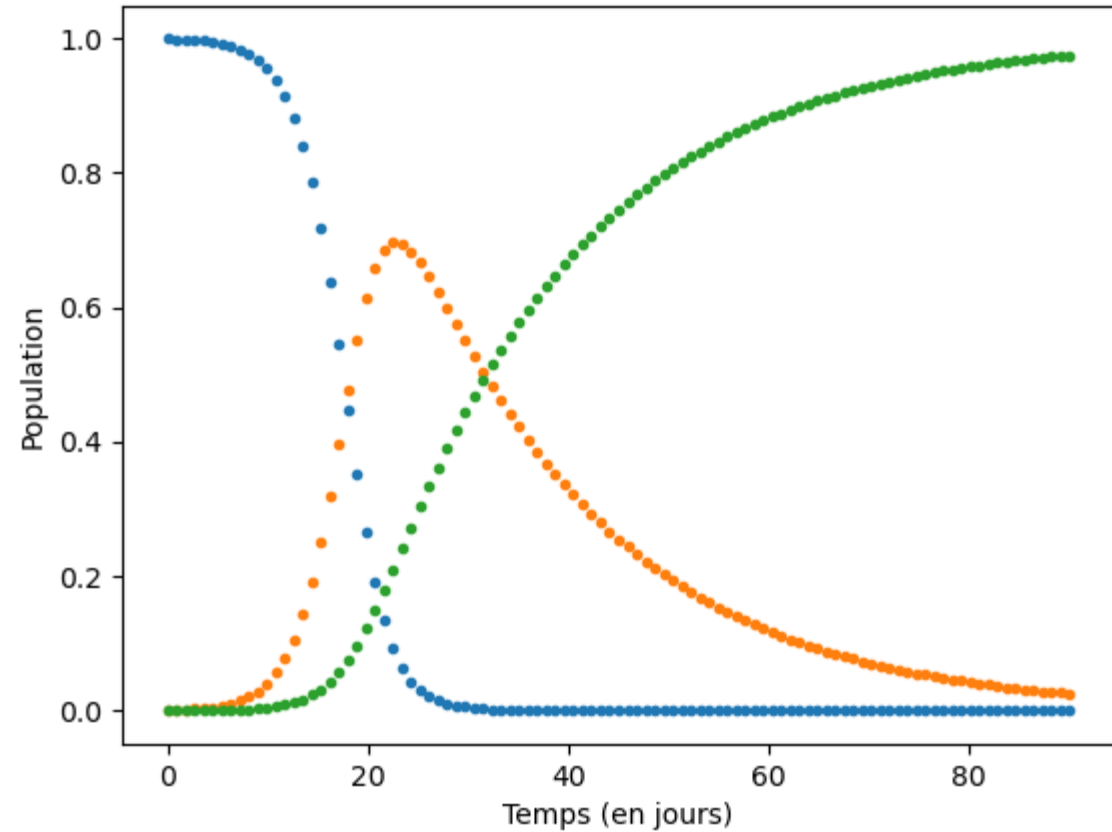
**Paramètres :**

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

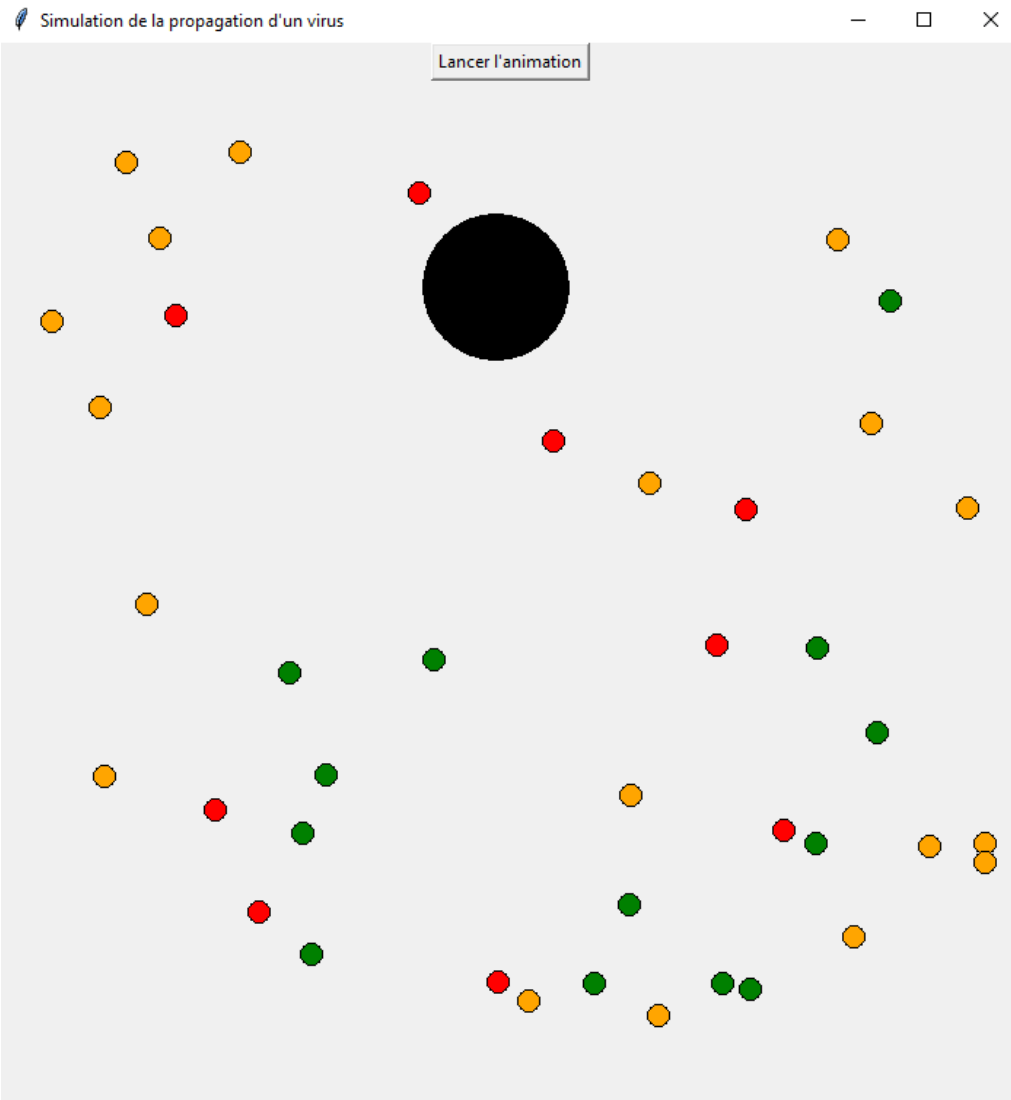
Modèle SIR



Evolution de la taille des 3 catégories de personnes au cours du temps



## II) Simulation d'un phénomène de foule



> Quelques définitions relatives à la simulation :

- Point attracteur
- Contact

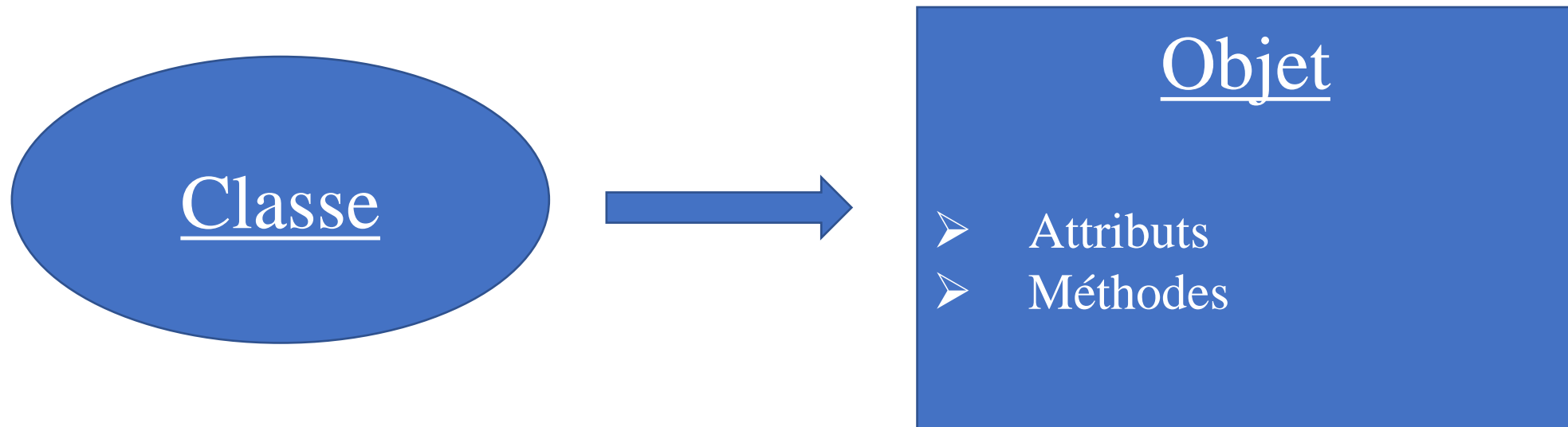
> Comment définir cette fenêtre ?

> Que se passe-t-il après avoir cliqué sur le bouton « Lancer l'animation » ?

```
7 height = 700
8 width = 700
9 scale = 15 # 25 pixels sur le dessin représente 1 mètre
10 point_dimension = scale # ODG Individu : 1m
11 n = 40 # Nombre d'individus à représenter
12
13 stantard_contact = {
14     "distance": 1, # distance inférieure ou égale à 1 mètre en cas de contact...
15     "durée": 5, # ... PENDANT une durée de 5 minutes
16     "beta": 1 # Probabilité d'être infecté après avoir été en contact avec un individu infecté
17 }
18 point_data = {
19     "diameter": point_dimension,
20     "colors": ["green", "orange", "red"],
21     "diameter_attractor": 100
22 }
28 sim = Simulation(height, width, stantard_contact, point_data, scale, db)
```

## 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



## > Classe simulation

```
1 import tkinter as tk
2 import random
3 from simulation.point import Point
4 from server.contact import Contact
5
6
7 class Simulation:
8     def __init__(self, height, width, strd_contact, point_data, scale, db):
9         """
10         Cette classe représente une fenêtre dans laquelle se déroulera la simulation.
11         La simulation correspondra a une succession d'états permettant de mettre en évidence la transmission du virus :
12             A un certain état n, chaque point sera caractérisé par sa couleur et par sa position.
13             Le mouvement d'un point est réalisé grace au changement de ses coordonnées lors de la transition d'un état n
14             à un état n+1.
15         """
16
17         # Paramètres de la fenêtre
18         self.height = height
19         self.width = width
20         self.window = tk.Tk()
21         self.window.title("Simulation de la propagation d'un virus")
22         self.canvas = tk.Canvas(self.window, width=700, height=700)
23         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
```

## > Classe point

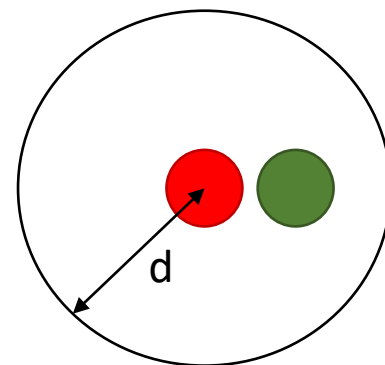
```
6 class Point:
7     """
8     Cette classe modélise un individu par un cercle coloré.
9     """
10    def __init__(self, x, y, diameter, color, canvas):
11        self.x = x
12        self.y = y
13        self.color = color
14        self.canvas = canvas
15        self.diameter = diameter
16        self.id = None
17        self.user = None
```

```

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 
```



## Contact





# 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** :

<i>utilisateurs</i>	<i>donnees_de_sante_categories</i>	<i>donnees_de_sante</i>	<i>utilisateurs_donnees_de_sante</i>	<i>contact</i>
<ul style="list-style-type: none"><li>- id</li><li>- etat</li></ul>	<ul style="list-style-type: none"><li>- id</li><li>- nom</li></ul>	<ul style="list-style-type: none"><li>- id</li><li>- nom</li><li>- categorie</li></ul>	<ul style="list-style-type: none"><li>- utilisateur_id</li><li>- donnee_id</li><li>- valeur</li><li>- info_supplementaire</li></ul>	<ul style="list-style-type: none"><li>- id</li><li>- utilisateur1_id</li><li>- utilisateur2_id</li><li>- contamination</li></ul>

### III.1) Représentation d'une base de données en Python

```
1  import mysql.connector
2
3
4  class Database:
5      def __init__(self, host, user, password, database_name):
6          """
7          Cette classe permet de gérer les actions effectuées sur la base de données sélectionnée.
8
9          :param host: Hôte du server MySQL
10         :param user: Nom d'utilisateur
11         :param password: Mot de passe
12         :param database_name: Nom de la base de données
13         """
14         self.connection = mysql.connector.connect(
15             host=host,
16             port=3306,
17             user=user,
18             database=database_name,
19             password=password
20         )
21
22         self.cursor = self.connection.cursor()
```

## Actions sur une base de données

### 1) Modifier une table

```
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
```

## Actions sur une base de données

### 2) Récupérer des informations

```
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
```

### III.2) Création de nouvelles classes

Une classe HealthData, dont les attributs sont générés dynamiquement grâce à la structure des tables relatives aux données de santé.

```
5 class HealthData:
6     def __init__(self, user_id, db):
7         """
8         Cette classe permet de générer pour tout individu un objet contenant les données de santé essentielles.
9         Pour ce travail on se limite à 3 types de données de santé, enregistrés dans une bdd :
10        - Le premier correspond aux données de santé à caractère personnel (antécédents médicaux, maladies, traitements,
11          handicap, etc...)
12
13        - Le second correspond à des données relatives aux facteurs extrinsèques environnementaux non-personnels de
14          santé (qualité de l'environnement évaluée en fonction de celle de l'eau ou de l'air, contexte régional)
15
16        - Le troisième correspond à des données de contextualisation de la santé relative à l'individu (IMC,
17          alimentation, consommation d'alcool/tabac, catégorie socioprofessionnelle, l'appartenance à une classe scolaire)
18
19        # Les objets issus de cette classe sont construits à partir de la structure de la BDD #
20        """
21
22        self.user_id = user_id
23        self.nb_types = len(db.select("id", "donnee_de_sante_categories"))
24        self.data_health_types = [{ } for _ in range(self.nb_types)]
25
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
```

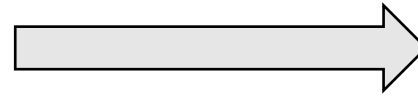
## Que veut dire généré dynamiquement ? Quel est l'intérêt ?

Table **donnees\_de\_sante\_categories**

id	nom
1	données de santé
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	traitement	1
4	handicap	1
5	environnement sain	2
6	IMC	3
7	alcool	3
8	tabac	3
9	école	3



Type de données de santé n°1

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

Type de données de santé n°2

```
{  
    5: None  
}
```

## III.2) Création de nouvelles classes

Une classe User

```
5  class User:
6      def __init__(self, id, state):
7          self.id = id
8          self.state = state
9          self.health_data = None
23  def initialize_health_data(self, db):
24      """
25      """
26      self.health_data = HealthData(self.id, db)
27      self.health_data.generate()
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 !")
```

## III.2) Création de nouvelles classes

Une classe Contact

```
1  class Contact:
2      def __init__(self, users, contamination):
3          self.users = users
4          self.contamination = contamination

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



### III.3) Lien avec la simulation

```
21     def create_user(self, db):
22         """
23         Chaque point modélise un individu. On associe alors à chaque objet de type point un individu possédant des
24         données de santé générées aléatoirement.
25         :param db:
26         :return:
27         """
28         if self.is_contaminated():
29             self.user = User(self.id, "Infecté")
30         elif self.is_healthy():
31             self.user = User(self.id, "Sain")
32         else:
33             self.user = User(self.id, "Rétabli")
34
35         self.user.initialize_health_data(db)
36
37         # On ajoute l'utilisateur dans la BDD
38         self.user.insert_in_db(db)
39         # On y ajoute également ses données de santé
40         self.user.get_health_data().insert_in_db(db)
```

```

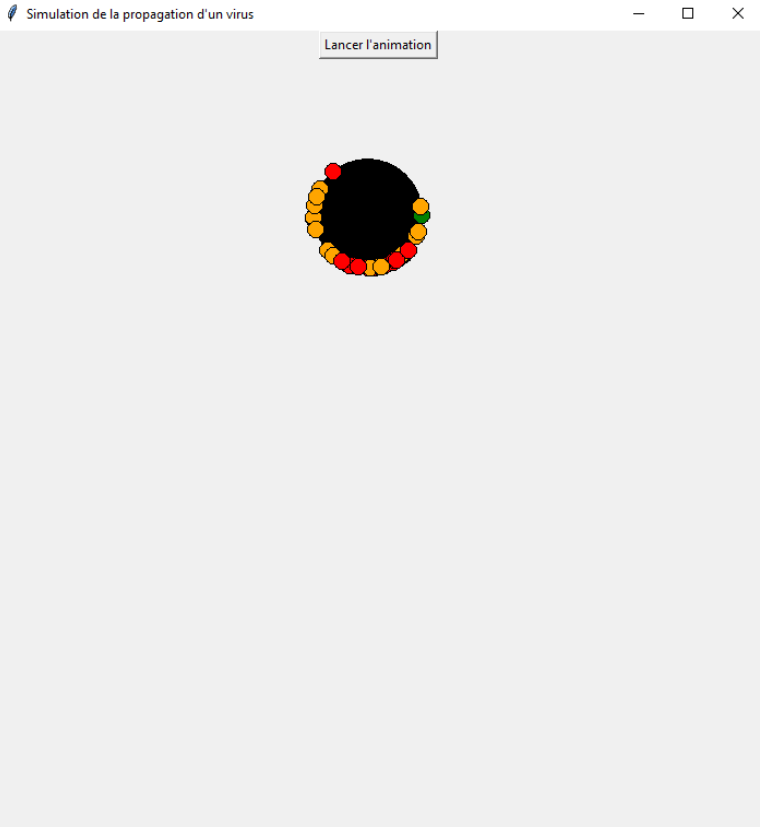
133     def run_animation(self):
134         """
135         Déplacer tous les points vers le point attracteur tant qu'ils n'y sont pas.
136         :return:
137         """
138         all_point_on_attractor = True
139         n = len(self.points)
140         for i in range(n):
141             point, 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 crée 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

```

Résultats à la fin de la simulation

id	etat	utilisateur_id	donnee_id	valeur	info_supplementaire
2	Infecté	2	1	0	
3	Infecté	2	2	1	
4	Sain	2	3	1	
5	Sain	2	4	0	
6	Sain	2	5	0	
7	Rétabli	2	6	0	
8	Infecté	2	7	0	
9	Sain	2	8	0	
10	Rétabli	2	9	1	
11	Rétabli	3	1	1	
12	Sain	3	2	0	
13	Sain	3	3	1	
14	Rétabli	3	4	0	
15	Rétabli	3	5	1	
16	Rétabli	3	6	0	
17	Infecté	3	7	0	
18	Sain	3	8	0	
19	Infecté	3	9	0	
20	Sain	4	1	0	
21	Sain	4	2	1	
22	Sain	4	3	0	
23	Rétabli	4	4	0	
24	Rétabli	4	5	0	
25	Sain	4	6	0	
26	Sain	4	7	0	

id	utilisateur1_id	utilisateur2_id	contamination
1	39	30	0
2	8	15	0
3	35	30	0
4	8	40	0
5	35	2	0
6	35	39	0
7	8	14	0
8	39	18	1
9	18	30	0
10	18	35	0
11	2	34	0
12	39	7	0
13	35	34	0
14	36	5	1
15	5	22	1
16	22	26	1
17	22	32	1
18	5	26	0
19	5	32	0
20	18	20	1
21	20	30	0
22	20	39	0
23	20	35	0
24	36	26	0



Conclusion, lien avec l'application TousAntiCovid

## main.py

```
1  from simulation.simulationc import Simulation
2  from server.database import Database
3
4
5  # ----- Paramètres ----- #
6
7  height = 700
8  width = 700
9  scale = 15 # 25 pixels sur le dessin représente 1 mètre
10 point_dimension = scale # ODG Individu : 1m
11 n = 40 # Nombre d'individus à représenter
12
13 stantard_contact = {
14     "distance": 1, # distance inférieure ou égale à 1 mètre en cas de contact...
15     "durée": 5, # ... PENDANT une durée de 5 minutes
16     "beta": 1 # Probabilité d'être infecté après avoir été en contact avec un individu infecté
17 }
18 point_data = {
19     "diameter": point_dimension,
20     "colors": ["green", "orange", "red"],
21     "diameter_attractor": 100
22 }
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

```
1  import math
2  from random import random
3  from server.user import User
4
5
6  class Point:
7      """
8      Cette classe modélise un individu par un cercle coloré.
9      """
10     def __init__(self, x, y, diameter, color, canvas):
11         self.x = x
12         self.y = y
13         self.color = color
14         self.canvas = canvas
15         self.diameter = diameter
16         self.id = None
17         self.user = None
18
19     def create_user(self, db):
20         """
21         Chaque point modélise un individu. On associe alors à chaque objet de type point un individu possédant des
22         données de santé générées aléatoirement.
23         :param db:
24         :return:
25         """
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

```
1  import tkinter as tk
2  import random
3  from simulation.point import Point
4  from server.contact import Contact
5
6
7  class Simulation:
8      def __init__(self, height, width, strd_contact, point_data, scale, db):
9          """
10             Cette classe représente une fenêtre dans laquelle se déroulera la simulation.
11             La simulation correspondra a une succession d'états permettant de mettre en évidence la transmission du virus :
12                 A un certain état n, chaque point sera caractérisé par sa couleur et par sa position.
13                 Le mouvement d'un point est réalisé grace au changement de ses coordonnées lors de la transition d'un état n
14                 à un état n+1.
15             """
16
17             # Paramètres de la fenêtre
18             self.height = height
19             self.width = width
20             self.window = tk.Tk()
21             self.window.title("Simulation de la propagation d'un virus")
22             self.canvas = tk.Canvas(self.window, width=700, height=700)
23             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 possèdera 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

```
1 import mysql.connector
2
3
4 class Database:
5     def __init__(self, host, user, password, database_name):
6         """
7         Cette classe permet de gérer les actions effectuées sur la base de données sélectionnée.
8
9         :param host: Hôte du server MySQL
10        :param user: Nom d'utilisateur
11        :param password: Mot de passe
12        :param database_name: Nom de la base de données
13        """
14        self.connection = mysql.connector.connect(
15            host=host,
16            port=3306,
17            user=user,
18            database=database_name,
19            password=password
20        )
21
22        self.cursor = self.connection.cursor()
23
24    def get_cursor(self):
25        """
26        Le curseur est un objet permettant d'exécuter 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
36        :param table_name:
37        :param attribute:
38        :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

```
1 from server.database import Database
2 from random import uniform
3
4
5 class HealthData:
6     def __init__(self, user_id, db):
7         """
8         Cette classe permet de générer pour tout individu un objet contenant les données de santé essentielles.
9         Pour ce travail on se limite à 3 types de données de santé, enregistrés dans une bdd :
10        - Le premier correspond aux données de santé à caractère personnel (antécédents médicaux, maladies, traitements,
11          handicap, etc...)
12
13        - Le second correspond à des données relatives aux facteurs extrinsèques environnementaux non-personnels de
14          santé (qualité de l'environnement évaluée en fonction de celle de l'eau ou de l'air, contexte régional)
15
16        - Le troisième correspond à des données de contextualisation de la santé relative à l'individu (IMC,
17          alimentation, consommation d'alcool/tabac, catégorie socioprofessionnelle, l'appartenance à une classe scolaire)
18
19        # Les objets issus de cette classe sont construits à partir de la structure de la BDD #
20        """
21
22        self.user_id = user_id
23        self.nb_types = len(db.select("id", "donnee_de_sante_categories"))
24        self.data_health_types = [{_ for _ in range(self.nb_types)]
25
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:
51             if health_data_id in health_data_type.keys():
52                 health_data_type[health_data_id] = new_value
53
54     def set_extra_data(self, db, health_data_id, info):
55         """
56         Permet d'ajouter des informations supplémentaires à un utilisateur selon une certaine donnée de santé.
57         :param db:
58         :param health_data_id:
59         :param info:
60         :return:
61         """
62         table_name = "utilisateurs_données_de_sante"
63         attribute = "info_supplementaire"
64         new_value = info
65         condition = "donnee_id = " + str(health_data_id) + " AND utilisateur_id = " + str(self.user_id)
66         db.update(table_name, attribute, new_value, condition)
67
68     @staticmethod
69     def get_db_attributes():
70         """
71         Retourne la chaîne de caractère correspondant aux attributs de la table 'utilisateurs_données_de_sante'.
72
73         :return: str
74         """

```

```

75         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)

```

```
1  class Contact:
2      def __init__(self, users, contamination):
3          self.users = users
4          self.contamination = contamination
5
6      @staticmethod
7      def get_db_attributes() -> str:
8          """
9          Retourne la chaîne de caractère correspondant aux attributs de la table 'users'.
10
11          :return: str
12          """
13          return "(utilisateur1_id, utilisateur2_id, contamination)"
14
15      def get_values(self):
16          return self.get_users()[0], self.get_users()[1], self.contamination
17
18      def get_users(self):
19          return self.users
20
21      def insert_in_db(self, db):
22          db.insert("contact", self.get_db_attributes(), self.get_values())
```

```
1  from server.database import Database
2  from server.health_data import HealthData
3
4
5  class User:
6      def __init__(self, id, state):
7          self.id = id
8          self.state = state
9          self.health_data = None
10
11     @staticmethod
12     def get_db_attributes() -> str:
13         """
14         Retourne la chaîne de caractère correspondant aux attributs de la table 'users'.
15
16         :return: str
17         """
18         return "(id, etat)"
19
20     def get_health_data(self):
21         return self.health_data
22
23     def initialize_health_data(self, db):
24         """
25         """
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 !")
```