Rapport
Mini projet 3
PROGRES

25/12/2021

Binôme :
- AKLI Mohammed Farouk.
- Eunbi Lee.

Table des matières

1 Introduction :

Le but de ce mini-projet est d'utiliser un simulateur fourni pour mener une campagne de simulation : exécuter le simulateur avec de nombreux paramètres différents, collecter et manipuler les données expérimentales produites par le simulateur, obtenir des graphiques résumant l'impact des paramètres suivis sur les indicateurs choisis.

Ce simulateur est écrit en code Java et modélise la propagation d'un virus dans une population. Il est fourni sous la forme d'un fichier **Virus.jar** qui intègre toutes les bibliothèques nécessaires.

Dans le premier exercice on va réaliser en Python et en utilisant la bibliothèque **subprocess** un programme qui exécute plusieurs fois le simulateur avec les mêmes paramètres de simulation. Ces exécutions multiples seront utilisées pour obtenir des intervalles de confiance suffisants lors de la phase de visualisation des données expérimentales. Les résultats intermédiaires vont être stockés dans un fichier ou un ensemble de fichiers (texte) pour l'ensemble des simulations menées.

Dans la 2eme partie du même exercice on va réaliser un programme qui exécute le simulateur en faisant varier un paramètre de la simulation, en réalisant plusieurs simulations pour chaque ensemble de paramètres. Les résultats intermédiaires vont être aussi stockés dans un fichier ou un ensemble de fichiers (texte) pour l'ensemble des simulations menées.

Dans le deuxième exercice et à l'aide des fichiers produits lors de l'exercice 1, en utilisant les bibliothèques **numpy** et **pandas** on va réaliser un programme qui permette d'obtenir les informations d'impact des différents paramètres changé sur la durée de l'épidémie, la proportion maximale de la population qui est infectée, la distribution des multi-infections. Les informations ainsi produites seront stockées dans un ou plusieurs fichiers (csv).

Dans le troisième exercice et à l'aide des fichiers produits lors de l'exercice 2, réaliser en Python et en utilisant la bibliothèque **matplotlib** un programme qui produit des graphiques pertinents pour évaluer l'impact des paramètres suivis (nombre initial d'infectés, rayon de mobilité, proportion de vaccinés, efficacité vaccinale, durée de contagiosité, durée d'immunité, densité de population) sur les indicateurs sélectionnés (la durée de l'épidémie, la proportion de la population qui est infectée, la distribution des multi-infections).

2 Exercice 1 :

2.1 Explications des décisions techniques :

Dans cet exercice nous allons simuler le fichier java en différentes méthodes :

2.1.1 Simulations avec le nombre du nœud :

Au début nous avons varier pour un certain nombre de simulations (10-20) un choix adapté à nos machines les différents paramètres d'exécution du simulateur **Virus.jar.**

Le choix des paramètres à varier a été pris en considération des différentes questions du deuxième exercices pour permettre d'évaluer l'impact de chaque variable sur les résultats obtenus.

Pour implémenter cette technique nous avons programmés des boucles **for** qui nous a permis de gérer plusieurs simulations et aussi de limiter le nombre de simulations et pour chaque itération on varie le paramètre choisi exemple : (nb_infected, travel_distance, infection_period, …) le choix du départ c'était le même du code original (même paramètre) et l'itération qui suit on ajoute une valeur adaptée pour essayer au maximum de visualiser la différence et son impact.

Et le résultat du fichier généré va être sous la forme d'un ensemble de trois valeurs (sain, infecté, immunisé) et qui varie en fur à mesure du temps.

Et ensuite on stocke chaque résultat de simulations dans un fichier texte (choisi par le binôme).

Code :

```
nb_infected = 2
travel_distance = 200
nb_vaccinated = 0
vaccine_efficiency = 2
infection_period = 100
contagion_period = 200
immune_period = 300
nb_nodes = 100

for i in range(10):
    # 5 simulations varying number of infected
    #gui=0 ==> to eliminate interface snapshot ==> number of sample
    # capture_output to to capture both STDOUT and STDERR independently
    proc = run(['java','-jar','Virus.jar','-gui=0', '-nb_snapshots=100','show_parameters =
0',f'-nb_infected={nb_infected}'],capture_output=True)
    nb_infected += 2
    with open ('1_nb_infected.txt','a') as file:
        file.write(proc.stdout.decode('utf-8'))
```

2.1.2 Simulations avec l'état du nœud :

Dans la deuxième partie nous avons travaillé avec le même principe que la première partie le seul changement c'est ce qu'on était obligé de forcer le paramètre « printout » à 2 pour générer des simulations qui peuvent afficher pour chaque nœud son état en fur à mesure du temps de la simulation ce qui va être pratique pour les questions (iii) du deuxième exercice.

Code :

```
for i in range(10):
    # 5 simulations varying number of infected
    #gui=0 ==> to eliminate interface printout for multiple snapshot ==> number of
sample
    # capture_output to to capture both STDOUT and STDERR independently
    proc = run(['java','-jar','Virus.jar','-gui=0','-printout=2', '-
nb_snapshots=100','show_parameters = 0',f'-
nb_infected={nb_infected}'],capture_output=True)
    nb_infected += 2
    with open ('2_nb_infected.txt','a') as file:
        file.write(proc.stdout.decode('utf-8'))
```

3 Exercice 2 :

Avant de commencer l'évaluation nous avons extrait les valeurs en utilisant la bibliothèque des expressions régulières « re » on a pu extraire les valeurs en mettant le paramètre « show_parameters » à 0 mais de cette manière là ça nous permet de couper chaque colonne (sain, infecté, immunisé) dans une liste pour le traitement par la suite.
En suite nous avons couper la liste qui nous intéresse « infectedList » dans un dictionnaire en plusieurs séries de 100 valeurs dans chaque clé (index : Rang).

Code :

```
with open('1_nb_infected.txt','r') as file:
    for child in file:
        # we could retrieve values only with show_parameters = 0 but that's help us to cut
        # each part
        if re.match('\d{1,3} \d{1,3} \d{1,3}', child):
            statList.append(child)

#Retrieve the statistics generated in the nb_infected file

for line in statList:
    # first value represent sane people
    sane = line.split(' ')[0]
    saneList.append(int(sane))
    # second value represent infected people
    infected = line.split(' ')[1]
    infectedList.append(int(infected))
    # third value represent immune people
    immune = line.split(' ')[2]
    immuneList.append(immune)

infectedFrame ={
    0 : Series(infectedList[0:100]) , 1 : Series(infectedList[100:200]), 2 :
Series(infectedList[200:300]),
    3 : Series(infectedList[300:400]) , 4 : Series(infectedList[400:500]), 5 :
Series(infectedList[500:600]),
    6 : Series(infectedList[600:700]) , 7 : Series(infectedList[700:800]), 8 :
Series(infectedList[800:900]),
```

```
    9 : Series(infectedList[900:1000])
}
```

3.1 Explications des décisions techniques :

Dans cet exercice nous allons séparer l'évaluation de l'impact de chaque paramètre modifié dans l'exercice 1 sur les aspects suivants :

3.1.1 Durée de l'épidémie :

Le principe pour trouver la durée d'épidémie c'est de basculer toutes les valeurs de chaque série de 100 de la liste jusqu'à trouver que le nombre de nœuds des infectés égale à 0, alors il n'y a plus d'individus qui peut infecter les autres et là on avait besoin de mettre un boolean pour stopper la condition et compter jusqu'à où on s'est arrêté sinon la durée de l'épidémie ça sera égale tout le parcours fait qui veut dire 99.
Et ensuite affecter résultat dans le data frame pour réaliser le traçage dans l'exercice qui suit.

Code :

```
for cmpt in range(10):
    boolean = False
    for i in range(cmpt*100,100*(cmpt+1)):

        if (infectedList[i]== 0 and not boolean):
            epidemieDuration.append(i-100*cmpt)
            # valeur de test
            boolean = True
            # toute les individus sont infectés pendant toute la simulation
    if (not boolean):
        epidemieDuration.append(99)
df= DataFrame({'durée épidémie':epidemieDuration})
```

3.1.2 Proportion maximale de la population infectée :

Le principe de cette question est simple c'est juste de chercher le maximum du nombre infecté dans chaque population (série) de 100.
Et ensuite affecter dans le data frame pour réaliser le traçage dans l'exercice qui suit.

Code :

```
#la proportion maximale de la population qui est infectée (ii)
df['proportion maximale infectés']= Series(dfInfected.max())
```

3.1.3 Distribution des multi-infections :

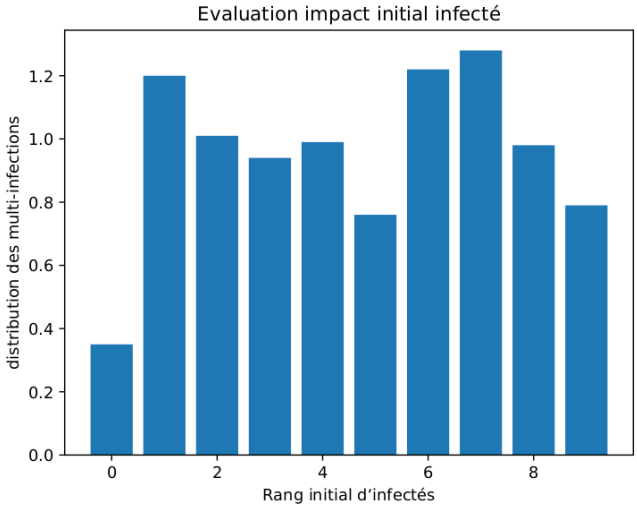
Le principe c'est qu'on lit chaque ligne du fichier et on place une nouvelle liste "a" avec un nombre représentant le statut d'infection de chacun. On crée une nouvelle liste "anew" qui est un moyen simple de représenter une séquence de nombres. Une séquence de 0,1 est considérée comme ayant été infectée une fois, et le nombre est stocké dans "number", et le nombre d'infections par tout le monde est stocké dans une liste nommée "howmanynumber". Le nombre moyen d'infections dans la liste "howmanynumber" est stocké dans une variable appelée "avg". Enfin, nous initialisons "avg" à 0 et l'appliquons à chaque simulation.

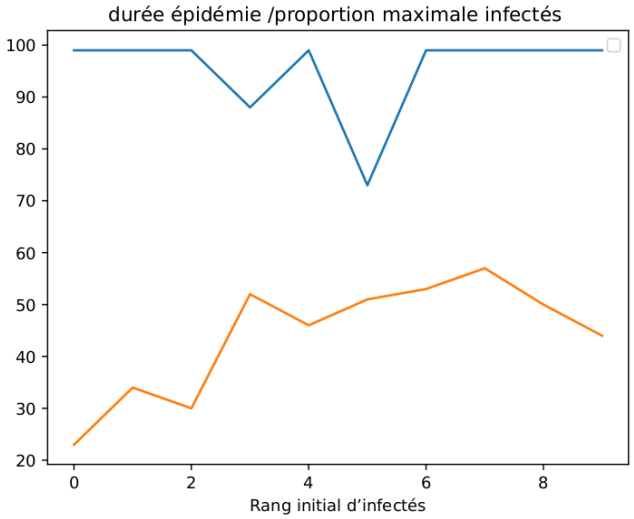
```
for simul in range (10) :
    for z in range (200) :
        try :
            a = []
            b = []

            with open('2_nb_infected.txt') as file:
                for child in file:
                    if re.match('[0-9]+', child) :
                        b.append(int(child[z])) #len(b) = 1000

            for c in range ( simul*100 , 100*(simul+1)) :
                a.append(b[c]) #len(a) = 100

            anew = []
            anew.append(a[0])

            for i in range (len(a)) :
                try :
                    if(a[i] != a[i+1] ) :
                        anew.append(a[i+1])
                except :
                    IndexError

            number = 0

            for j in range (len(anew)) :
                try :
                    if(anew[j] == 0 and anew[j+1] == 1) :
                        number += 1
                except :
                    IndexError

            #print(anew)
            #print("this {}th person was infected {} times".format(int((z+1)/2) +1 ,number))
            howmanynumber.append(number)
            #print("-"*30)
        except :
            ValueError

    #print("howmanynumber:", howmanynumber)
    sum = 0
    for s in range(len(howmanynumber)) :
        sum += howmanynumber[s]
    avg = sum/len(howmanynumber)
    #print("avg :", avg)
    listofavg.append(avg)
    howmanynumber.clear()
    avg = 0
```

4 Exercice 3 :

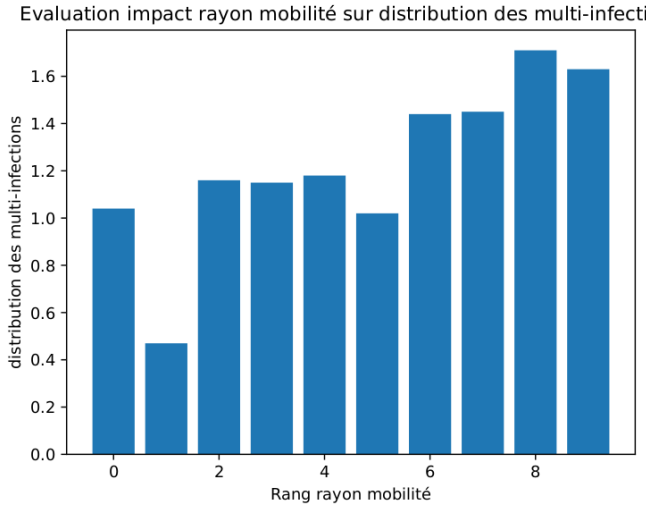
Pour procéder au graphe et tracer les données expérimentales, nous avons décider de mettre en œuvre deux graphes différents un graphe sur les indicateurs sur la durée de l'épidémie et la proportion maximale de la population infectée et un histogramme sur la distribution des multi-infections donc 16 graphes en tous.

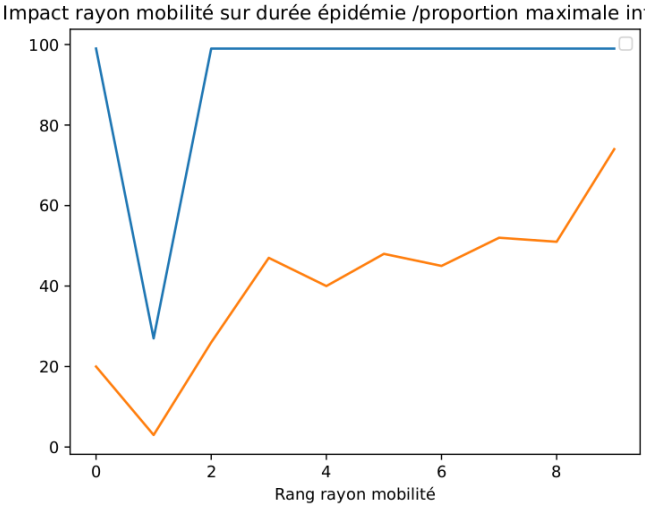
4.1 Nombre initial d'infectés :



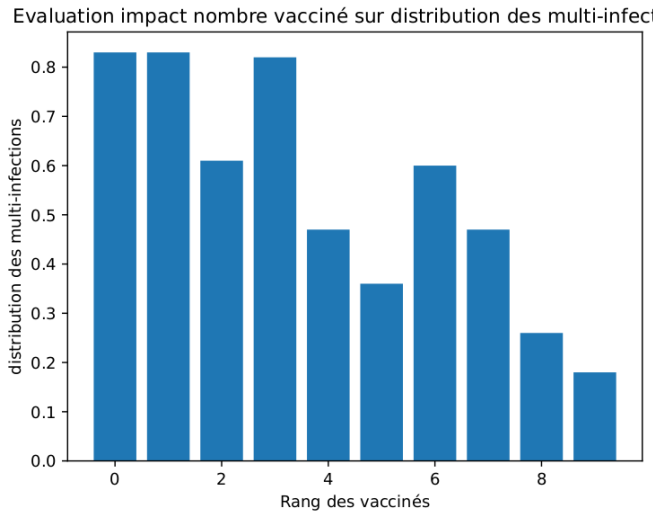


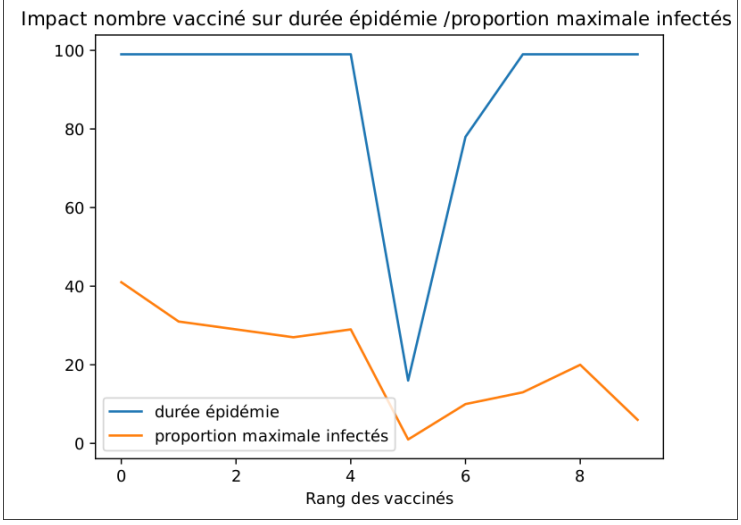
4.2 Rayon de mobilité :



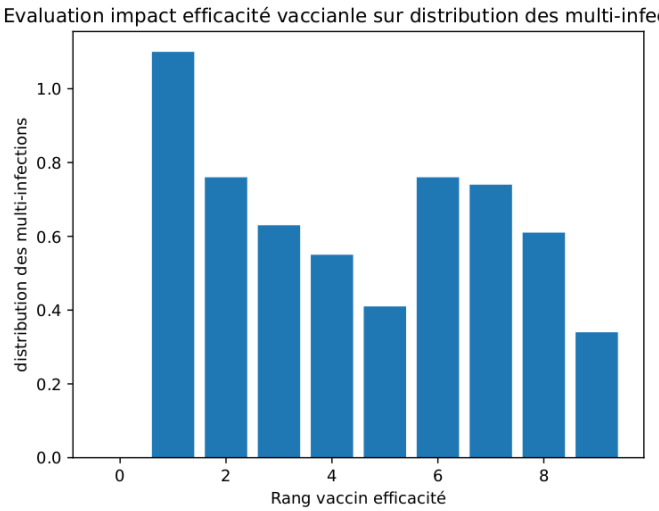


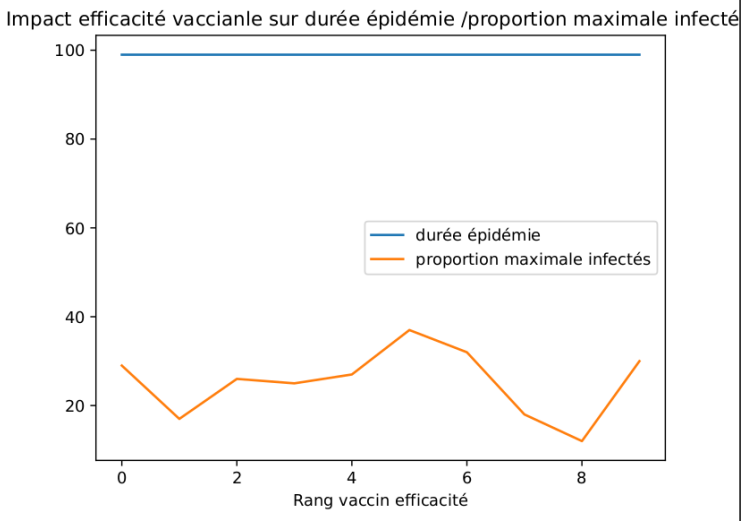
4.3 Proportion de vaccinés :



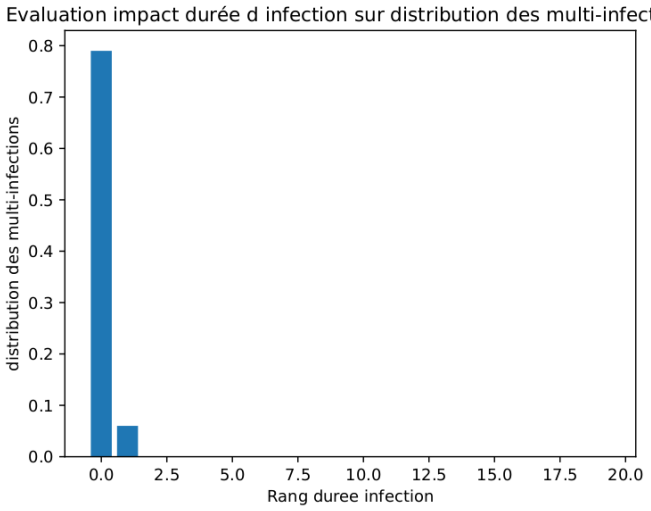


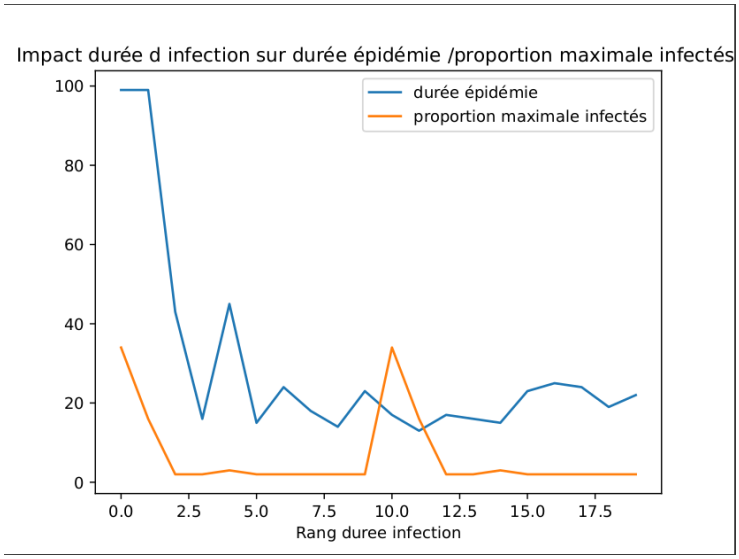
4.4 Efficacité vaccinale :



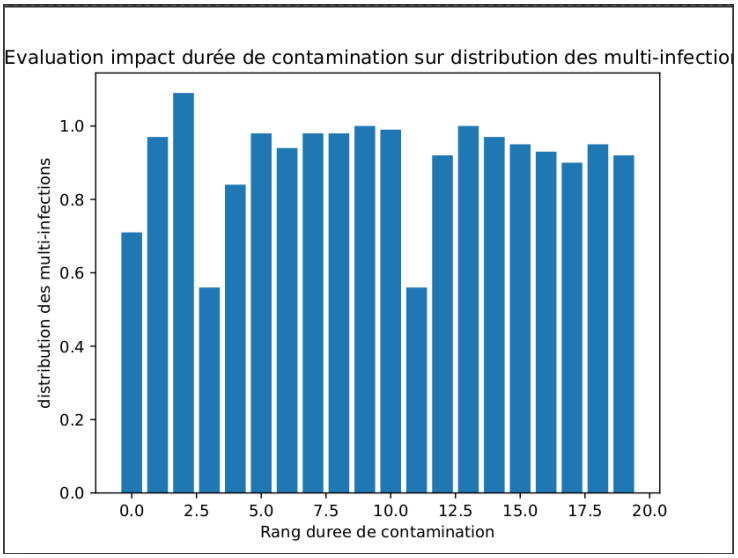


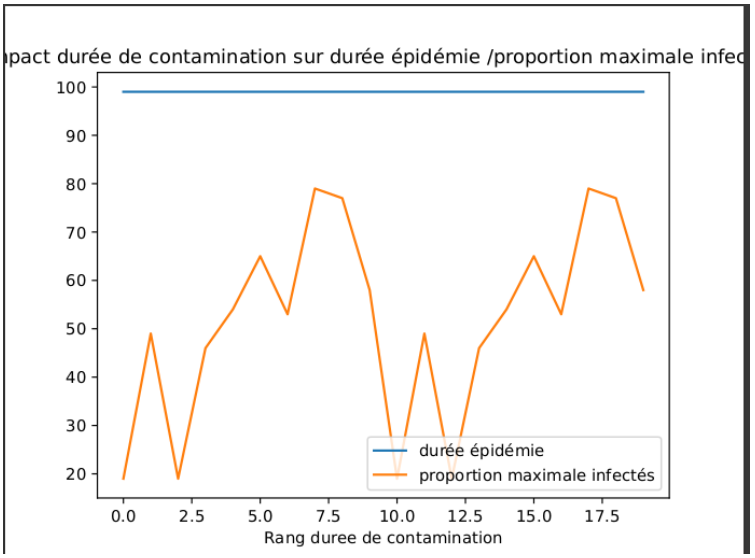
4.5 Durée d'infection :



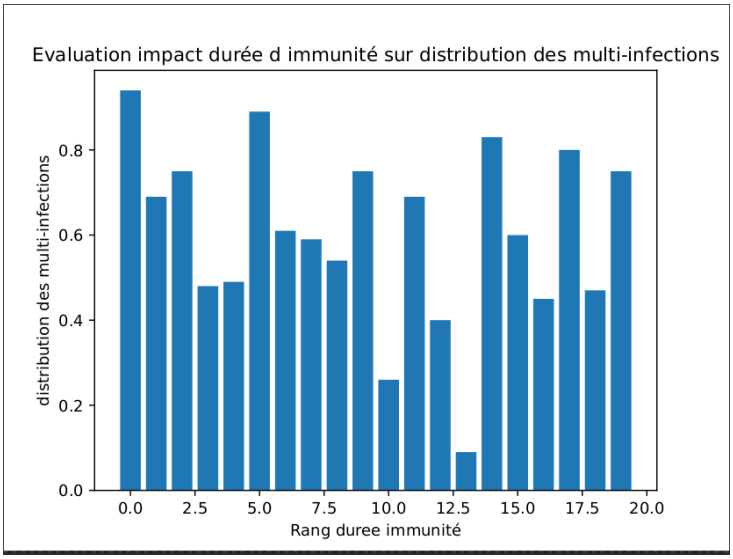


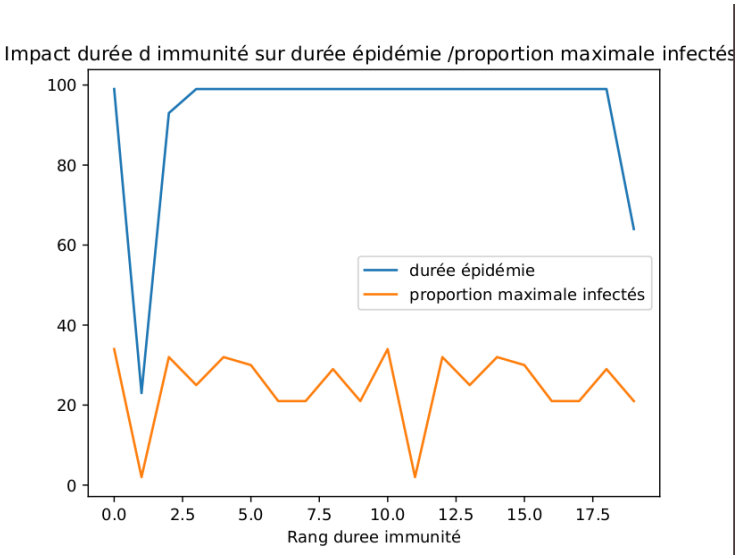
4.6 Durée de contagiosité :



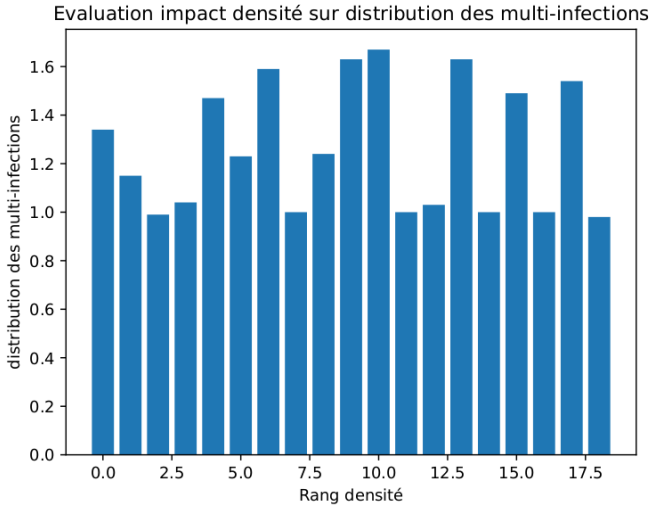


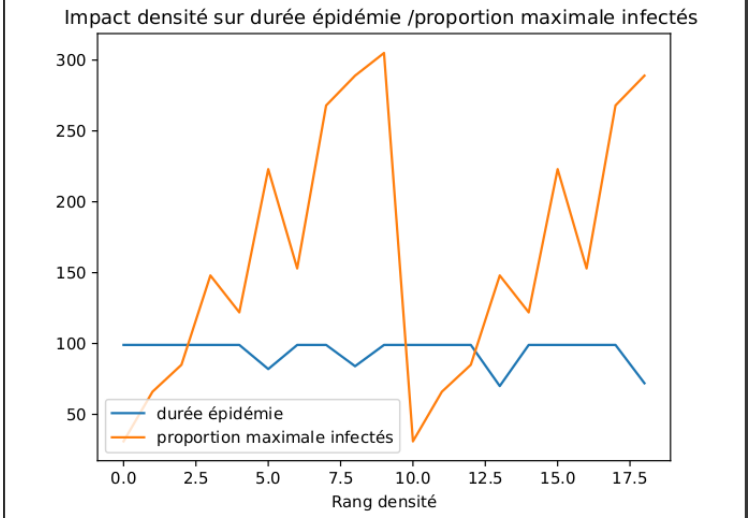
4.7 Durée d'immunité :





4.8 Densité de population :





5 Conclusion :

Dans notre projet on a rencontré le problème d'effectuer un nombre suffisant de simulation pour bien visualiser le résultat à cause que nos machines prennent beaucoup du temps pour simuler mais on a appris à utiliser les différentes bibliothèques python « subprocess », « numpy », « pandas », « matplotlib » qui permettent de faire une analyse complète de données.

6 Démonstration vidéo :

https://drive.google.com/file/d/1KfJjoMJUqpjG9Hs7KlZ7eztnEpkdoWlc/view

NB : Utilisation des écouteurs est recommandé pour une bonne qualité du son.