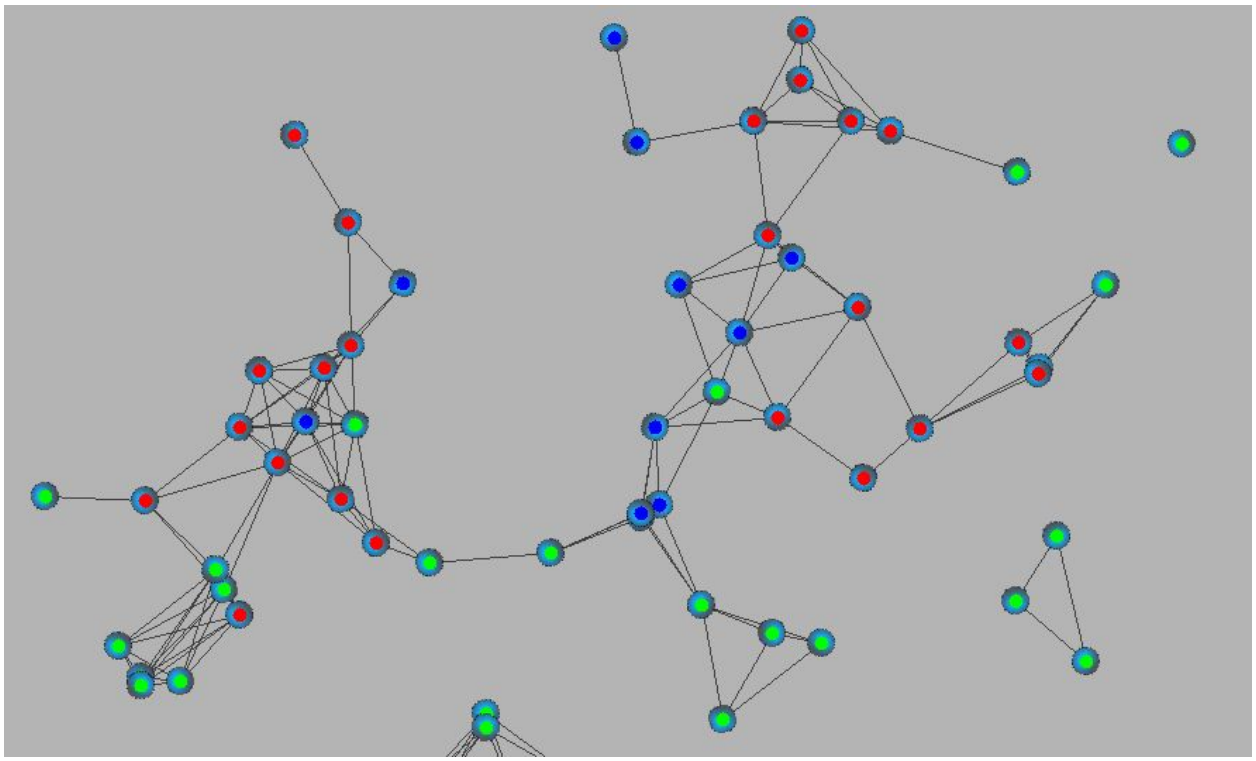
RAPPORT DE MINI-PROJET 3



<u>Binôme</u> : Alberto PETROSSI - Edgar OBLETTE

(Lien vers la vidéo de démonstration à la fin du document)

INTRODUCTION

Ce document est destiné à la maîtrise d'œuvre. Il peut donc contenir certains termes techniques que nous ne définirons pas dans un glossaire. Comme indiqué dans les consignes, il doit recenser les choix techniques effectués, ce qui a été implémenté, ce qui ne l'a pas été… et un lien dirigeant vers une vidéo de démonstration. Le but de ce 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 produites par le simulateur, obtenir des graphiques résumant l'impact des paramètres suivis sur les indicateurs choisis. Ce simulateur est écrit en 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.

Ainsi il faudra dans un premier exercice faire tourner un certain nombre de simulations, et stocker les données de celles-ci dans des fichiers afin de pouvoir les récupérer par la suite. Dans un second exercice, il faudra utiliser les données des simulations pour faire une étude de l'impact de certains paramètres sur le résultat des simulations. Enfin dans un troisième exercice, il faudra générer différents graphiques afin d'être en mesure d'observer visuellement l'impact de ces paramètres sur plusieurs indicateurs fournis par les résultats des simulations.

CHOIX TECHNIQUES EFFECTUÉS ET PRÉSENTATION GÉNÉRALE

L'objectif de ce projet était avant tout de réussir en python à exécuter un simulateur programmé en un autre langage (en Java dans notre cas) à l'aide de la bibliothèque "**subprocess**", puis de stocker dans des fichiers les données de ces simulations, et de les récupérer et manipuler en utilisant les bibliothèques "**pandas**" et "**numpy**", l'une permettant la création de dataframes, l'autre permettant de définir des matrices de données. Enfin, la bibliothèque "**matplotlib**" permet l'affichage des données une fois manipulées sous la forme de graphiques de différentes formes.

Dans un premier temps, notre groupe a fait tourner différentes simulations grâce à “**subprocess**”, et nous avons fait varier la valeur d’un paramètre en particulier à chaque simulation lancée. Cela nous aurait permis par la suite de pouvoir répondre aux différentes questions du deuxième exercice.

Ainsi nous avons choisi de faire varier les paramètres dont il est nécessaire d’évaluer l’impact sur différents aspects des résultats des simulations au cours du deuxième exercice.

Pour cela nous avons programmé des boucles for ayant pour limite le nombre de simulations que l’on souhaite exécuter, et à chaque itération nous avons fait augmenter la valeur d’un paramètre en particulier (par exemple du rayon de mobilité, ou du nombre initial de nœuds infectés). Sur la capture d’écran ci-dessous par exemple, nous avons choisi, en partant d’un nombre initial de nœuds infectés égal à 0 et stocké dans une variable, de faire augmenter de 10 sa valeur à chaque simulation lancée. Nous avons lancé dans ce cas 10 simulations successives.

```
#10 simulations en faisant varier le nombre d'infectés
for i in range(10):
    res = run(['java', '-jar', 'Virus_old.jar', '-gui=0', '-nb_snapshots=100', f'-nb_infected={nb_infected}'], capture_output=True)
    nb_infected+=10 #on augmente le nombre d'infectés de 10 à chaque itération
    with open('res_infected.txt', 'a') as f:
        f.write(res.stdout.decode('utf-8'))
```

Nous avons également décidé d’exécuter **deux types de simulations**: celles qui affichent l’évolution de l’état de chaque nœud de la topologie, et celles qui affichent le nombre de nœuds de la topologie à tel état (infecté, sain, immunisé) au cours de la simulation. Nous avons donc créé deux fichiers python distincts pour le premier exercice: le fichier **exercice1.py** permet d’exécuter les simulations affichant le nombre de noeuds infectés, sains et immunisés au cours de la simulation, et le fichier **exercice1_multiinfect.py** permet d’afficher pour chaque noeud son état au cours de la simulation. Pour obtenir le résultat généré par **exercice1_multiinfect.py** il a été nécessaire de modifier le paramètre “printout” de chaque simulation en lui attribuant la valeur 2.

Nous avons stocké les résultats des simulations dans divers fichiers “txt”. Les fichiers contenant le préfixe “res” désignent les résultats générés par **exercice1.py** pour chaque paramètre que l’on fait varier, ceux contenant le préfixe “multi” désignent les résultats générés par **exercice1_multiinfect.py**, et seront utiles pour comprendre l’impact des paramètres étudiés sur la distribution des multi-infections.

Ainsi une fois nos résultats générés, il a été nécessaire de les récupérer et de les manipuler dans un second exercice afin de pouvoir les analyser de manière pertinente.

Nous avons procédé de la manière suivante: nous sommes allés lire dans chaque fichier généré par les simulations, et grâce à l'utilisation d'expressions régulières, nous avons pu récupérer pour chaque fichier les données des simulations et les avons stockées dans une liste. Par exemple, supposons que l'on étudie les données des simulations obtenues dans le cas où on fait varier le paramètre "travel_distance" (donc le rayon de mobilité), on récupère dans une liste les données de toutes les simulations exécutées pour chaque valeur de "travel_distance".

Ensuite, dans "**exercice2.py**", on extrait de cette liste obtenue en lisant un des fichiers "txt" avec un préfixe "res" trois sets de données différents grâce à la bibliothèque "**pandas**": le premier set contient l'évolution du nombre de noeuds sains, le deuxième set contient l'évolution du nombre de noeuds infectés, et enfin le dernier contient l'évolution du nombre de noeuds immunisés.

Une fois ces trois sets établis, on les convertit en dataframes. Enfin, on extrait de ces dataframes des données utiles à notre étude que l'on place dans un nouveau dataframe. Il n'y a en réalité que le dataframe contenant le nombre de nœuds infectés au cours des simulations qui nous a été utile. Dans le cadre de l'exercice 2, les données utiles à extraire sont (i) la durée de l'épidémie, (ii) la proportion maximale de la population infectée, et (iii) la distribution des multi-infections.

Dans "**exercice2.py**" nous traitons uniquement la récupération de la durée de l'épidémie et de la proportion max infectée. Ainsi pour la proportion maximale, nous calculons le maximum du dataframe contenant les données du nombre de personnes infectées au cours des simulations, et nous plaçons le maximum obtenu pour chaque simulation dans le dataframe final. Ensuite pour la durée de l'épidémie, pour chaque simulation on observe le moment où plus aucune personne n'est infectée (toujours en se basant sur le dataframe avec les données des personnes infectées) et on sauvegarde ce moment. Ainsi on place dans le dataframe final le numéro du snapshot de la simulation où il n'y a plus de nœuds infectés pour chaque simulation lancée. Le dataframe final obtenu se présente donc de la manière suivante : la première colonne indique la valeur de la variable que l'on étudie (par exemple le nombre de nœuds initiaux infectés) pour chaque simulation lancée, la deuxième la proportion maximale infectée, la troisième la durée de l'épidémie.

Ci-dessous un exemple de dataframe obtenu lorsque l'on étudie les simulations où l'on fait varier la durée de contagiosité.

Durée de conta	Proportion ma	Durée Epidemi
10	2	1
20	2	3
30	2	4
40	2	5
50	2	6
60	2	7
70	2	8
80	2	8
90	10	35
100	14	76
110	5	46
120	5	48
130	16	72
140	11	75
150	17	99
160	24	99
170	27	99
180	28	92
190	27	95
200	34	99

Cependant il manque une colonne de données concernant la distribution des multi-infections. Pour obtenir ces données nous avons créé un fichier "**exercice2_multiinfect.py**" qui nous permettra de les récupérer. Ainsi cette fois-ci il a été nécessaire de faire appel à la bibliothèque "**numpy**", puisque nous étudions les fichiers "txt" commençant par "multi", qui eux contiennent l'évolution de l'état de chaque nœud, individuellement, dans la topologie. Ainsi, étant donné que les données à récupérer sont présentées sous la forme d'une matrice où chaque colonne représente l'évolution de l'état d'un des noeuds de la topologie, nous avons jugé plus pertinent de faire appel à "numpy" pour structurer les informations récupérées plutôt que d'utiliser

une simple liste.

Ainsi grâce aux matrices récupérées, il a fallu dans un deuxième temps les parcourir afin de détecter pour chaque nœud le nombre de fois où il passe de l'état sain à l'état infecté au cours d'une simulation, et ensuite calculer la moyenne des valeurs obtenues. Ces moyennes sont placées dans une liste, puis on ajoute aux dataframes déjà existants crées dans "**exercice2.py**" une nouvelle colonne contenant donc l'ensemble des moyennes obtenues en fonction des simulations, donc de la valeur de la variable que l'on fait varier.

Le rendu final des dataframes, que l'on enregistre dans des fichiers "csv" est présenté sous cette forme (ici on prend l'exemple des simulations où l'on fait varier la densité de la population):

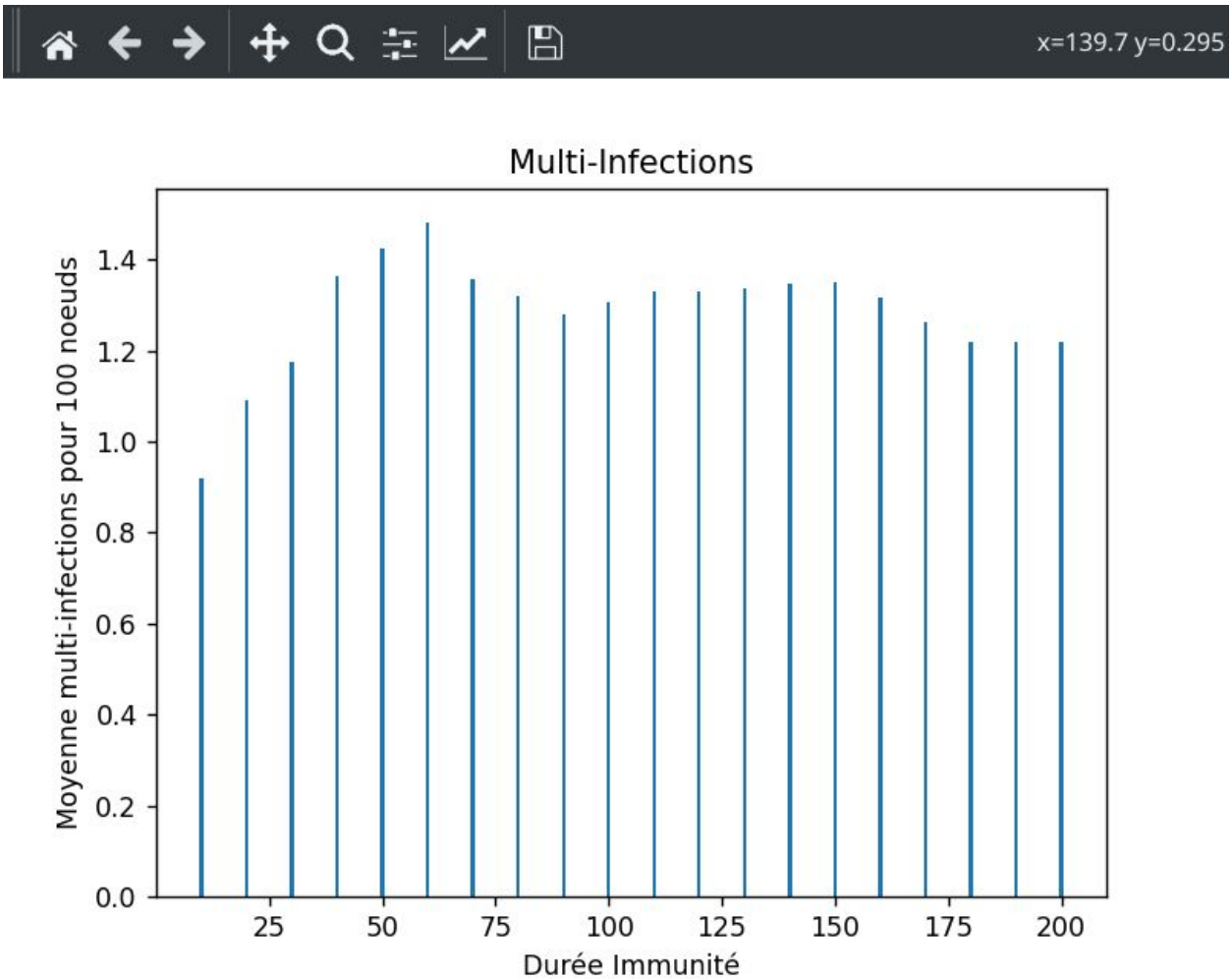
	Densité de la p	Proportion ma	Durée Epidemi	Distribution Mu
	1	1	19	1
	6	2	21	0.43
	11	2	23	0.33
	16	2	19	0.24
	21	2	59	0.24
	26	3	27	0.28
	31	8	63	0.25
	36	6	56	0.26
	41	7	65	0.23
	46	14	99	0.28
	51	5	67	0.33
	56	5	38	0.33
	61	12	99	0.29
	66	14	99	0.26
	71	23	99	0.29
	76	19	99	0.31
	81	30	99	0.35
	86	25	99	0.38
	91	17	99	0.43
	96	22	99	0.45

Ainsi ces dataframes obtenus (qui sont du nombre de 6, un pour chaque paramètre étudié) nous ont permis de traiter l'exercice 3.

Dans le troisième et dernier exercice, le but était de réutiliser les dataframes que l'on a généré et de produire à l'aide de la librairie "matplotlib" des graphiques permettant d'évaluer visuellement l'impact d'un paramètre sur les trois aspects des simulations que l'on étudie (durée de l'épidémie, proportion maximale infectée, distribution des multi-infections).

Ainsi nous avons décidé de représenter **sur un histogramme**, pour chaque dataframe étudié, l'évolution de la distribution des multi-infections selon la valeur du paramètre qui varie au cours des simulations.

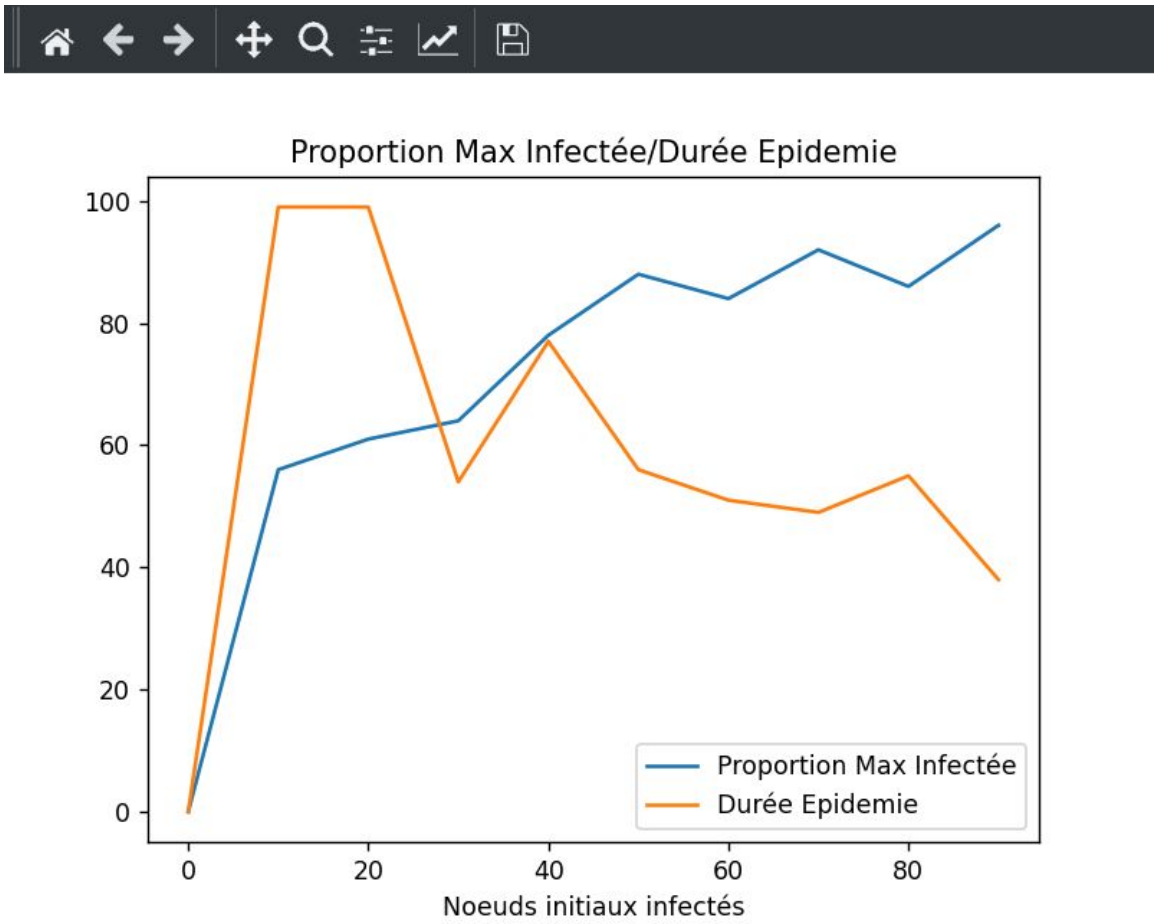
Par exemple pour le fichier csv "**immune_period.csv**", qui contient les données de l'impact du paramètre concernant la période d'immunité (immune_period) sur la durée de l'épidémie, la proportion max infectée et la distribution des multi-infections, l'histogramme généré va être le suivant :



Comme on peut l'observer, en abscisse est indiquée la valeur de la durée d'immunité (chaque valeur testée au cours d'une simulation) et en ordonnée la distribution des multi-infections qui s'exprime sous la forme d'une moyenne calculée pour les 100 noeuds de la topologie.

Ensuite pour les deux autres aspects étudiés dans les résultats des simulations, qui sont la durée de l'épidémie et la proportion maximale infectée, nous avons généré des graphiques où ces deux valeurs sont représentées ensemble sous la forme de courbes qui évoluent selon le paramètre que l'on fait varier pour chaque simulation.

Ainsi, si par exemple on choisit le dataframe "**infectedNodes.csv**", qui sert à étudier l'impact du nombre initial de noeuds infectés dans la topologie de 100 noeuds (celle de base), le graphique généré pour étudier son impact sur la durée de l'épidémie et la proportion maximale infectée est le suivant :



Ainsi nous avons répété cette représentation graphique choisie pour l'ensemble des dataframes "csv" étudiés, donc pour chaque paramètre dont on cherche à montrer l'impact sur la proportion max infectée, la durée de l'épidémie et la distribution des multi-infections. Après exécution de notre programme dans "**exercice3.py**", 12 graphiques (de la figure 1 à la figure 12) sont au total générés et il est possible pour l'utilisateur de les consulter. La bibliothèque "matplotlib" permet également de sauvegarder les graphiques obtenus sous format pdf mais pour ne pas alourdir notre projet nous n'avons pas fait recours à ce stratagème.

Ainsi à travers ces différentes étapes, nous avons été en mesure de lancer différentes simulations en faisant varier un paramètre particulier, de récupérer dans des fichiers texte les données de ces simulations, d'extraire d'autres données à partir de ces simulations ((i), (ii) et (iii)), de les classer sous forme de dataframe en utilisant entre autres la bibliothèque "**pandas**", de générer ces dataframes en format "csv" et enfin de les réutiliser pour représenter graphiquement sous deux formes pertinentes (histogramme et courbes) l'impact des différents paramètre étudiés sur la durée de l'épidémie, la proportion max infectée et la distribution des multi-infections.

Il est donc maintenant possible pour l'utilisateur d'en conclure quant à l'impact de ces paramètres sur ces indicateurs grâce à notre programme.

PROBLÈMES RENCONTRÉS

Lors de l'élaboration de notre projet, nous n'avons malheureusement pas pu effectuer autant de simulations que prévu. En effet, nous avons constaté que nos machines prenaient un temps excessif à exécuter les simulations si nous voulions en effectuer plusieurs pour un même ensemble de paramètres, et en le faisant varier à chaque fois.

Nous avons donc jugé préférable pour la bonne avancée de notre projet de ne réaliser qu'une seule simulation par paramètre que l'on choisit de faire varier. Par exemple, on exécute 10 simulations avec à chaque itération, une augmentation de valeur pour le paramètre qu'on étudie. C'est la solution qui nous a semblé la plus efficace pour surmonter ce problème: malheureusement la conséquence est que nous n'avons pas été en mesure de générer d'intervalles de confiance, étant donné qu'il aurait fallu diverses simulations pour chaque ensemble de paramètres afin de pouvoir en générer. Les données obtenues ne sont donc pas les plus pertinentes possibles, mais étant donné le temps pris par nos machines pour réaliser les simulations, nous avons préféré opter pour cette méthode qui reste tout de même valable.

Un second problème concerne la lecture fastidieuse dans les fichiers “txt” pour récupérer les données des simulations, qui nous a pris beaucoup de temps: pour se compliquer moins la tâche, il aurait peut-être été préférable de stocker les données sous un autre format plutôt qu’un simple format textuel.

Nous avons également généré un très grand nombre de fichiers “txt” afin de récupérer nos données, puisque nous avons estimé trop chaotique le stockage des données dans un fichier unique. Nous avons donc opté pour un stockage réparti dans plusieurs fichiers en fonction des simulations, ce qui nous a permis d’obtenir une meilleure vision et un accès plus rapide aux résultats des différentes simulations.

Enfin, pour ce qui est du calcul de la durée de l’épidémie, étant donné que nous avons exécuté des simulations de 100 snapshots, lorsque l’épidémie durait plus que cette valeur, nous avons estimé que l’épidémie durait plus de 99, représentant le 100e snapshot. Ce qui explique la valeur 99 dans les résultats obtenus pour certaines simulations.

CONCLUSION

Au cours de ce projet nous avons appris à exécuter en python des sous-programmes grâce à “**subprocess**” (ici un programme Java), à y récupérer des informations, à les placer dans des matrices et des dataframes (“**numpy**” et “**pandas**”), afin de pouvoir dans un dernier temps les représenter graphiquement en utilisant la librairie “**matplotlib**”.

Ainsi **c’est une analyse complète de données** que nous avons pu réaliser grâce au langage python.

LIEN VERS LA VIDÉO DE DÉMONSTRATION

https://youtu.be/vYc_8jh7PZw