TP2: Projet Informatique

Jacques KOZIK

April 4, 2024

Contents

1 Introduction **3**

2 Description du modèle **3**
2.1 Générateur Mersenne Twister 3
2.2 Implémentation de Classes 4

1 Introduction

La modélisation de la propagation des maladies dans une population est un outil essentiel pour comprendre et gérer les épidémies.

Les modèles multi-agents offrent une approche puissante pour simuler les interactions individuelles au sein d'une communauté et étudier l'émergence de phénomènes collectifs tels que la propagation d'une maladie infectieuse. Dans cette étude, nous développons un modèle multi-agent simple pour simuler la propagation d'une maladie au sein d'une population, en mettant en œuvre des mécanismes tels que l'infection, l'exposition, la récupération et la perte d'immunité.

Le modèle repose sur une grille en deux dimensions où chaque individu est caractérisé par son statut de santé, représenté par les états Susceptible (S), Exposé (E), Infecté (I) et Rétabli (R), ainsi que par les durées de vie associées à chaque état. Nous utilisons un espace toroïdal pour éviter les artefacts de bordure et permettre une interaction fluide entre les individus.

L'évolution de la maladie est modélisée sur une échelle de temps discrète, où chaque pas de temps correspond à un jour. Les individus se déplacent aléatoirement à chaque pas de temps, simulant ainsi les interactions spatiales au sein de la population. L'infection se produit lorsque des individus susceptibles entrent en contact avec des individus infectés dans leur voisinage, avec une probabilité calculée en fonction du nombre d'infectieux présents.

Nous intégrons également des mécanismes de transition d'état asynchrones, où les individus passent de l'état exposé à l'état infecté après une période d'exposition, puis de l'état infecté à l'état rétabli après une période infectieuse. De plus, nous prenons en compte la perte d'immunité avec le temps, où les individus rétablis peuvent redevenir susceptibles après une période donnée.

En initialisant la simulation avec un nombre spécifique d'individus dans chaque état de santé et en définissant les paramètres de durée de vie de manière aléatoire, nous examinons comment différents scénarios affectent la dynamique de la maladie au fil du temps.

Dans cette étude, nous présentons le cadre général de notre modèle multi-agent de propagation de la maladie et discutons de son application potentielle pour comprendre et prévoir les épidémies dans des contextes réels.

2 Description du modèle

Pour mettre en œuvre cette simulation, nous avons opté pour une approche combinant des classes Java avec l'utilisation du générateur Mersenne Twister. Et pour le rendu, nous avons inscrits nos résultats dans des fichiers CSV puis avons, à l'aide d'un notebook Jupyter, généré des graphiques pour les illustrer.

2.1 Générateur Mersenne Twister

Le générateur Mersenne Twister, largement utilisé en Java pour la génération de nombres aléatoires, est apprécié pour sa haute qualité et sa performance. En Java, il est implémenté

sous forme de classe, offrant une interface simple et efficace pour la génération de nombres aléatoires. Cette méthode de génération de nombres aléatoires est basée sur un algorithme développé par Matsumoto et Nishimura, qui produit des séquences de nombres pseudo-aléatoires présentant d'excellentes propriétés statistiques. En utilisant le générateur Mersenne Twister en Java, les développeurs peuvent facilement introduire des éléments de stochasticité dans leurs simulations, ce qui est crucial pour modéliser des phénomènes complexes tels que la propagation des maladies dans une population. La robustesse et la fiabilité de Mersenne Twister en font un choix populaire pour de nombreuses applications nécessitant une génération de nombres aléatoires de haute qualité en Java.

Nous avons également ajouter une méthode negExp dans la classe MTrandom, qui prend en paramètre un réel mean, et qui renvoie un réel entre 0 et mean.

```
/*method negExp, qui genere un nombre entre 0 et inMean*/
public double negExp(double inMean) {
    return -inMean * Math.log(1 - this.nextDouble());
 }
```

2.2 Implémentation de Classes

Nous avons décidé de créer une classes