FLU EPIDEMIC

2015-2016 / POLYTECH NICE SOPHIA - POO

Axel Aiello , Antoine Steyer

SI3 – GROUPE 1 |

PROJET SIMULATION D'EPIDEMIE

SOMMAIRE

1. PRESENTATION DU PROJET

Le projet consiste en la représentation d'une simulation d'une épidémie. Dans notre cas nous nous intéressons à l'épidémie « Flu ».

Nous devons simuler une épidémie avec quatre espèces, humains, cochons, poulets et canards. Les cochons seront à l'origine de l'épidémie de H1N1 alors que les poulets et canards seront à l'origine de l'épidémie de H5N1.

La simulation s'arrête quand tous les individus présents sont morts ou en bonne santé (et non porteur de maladie).

2. PARTIE GRAPHIQUE

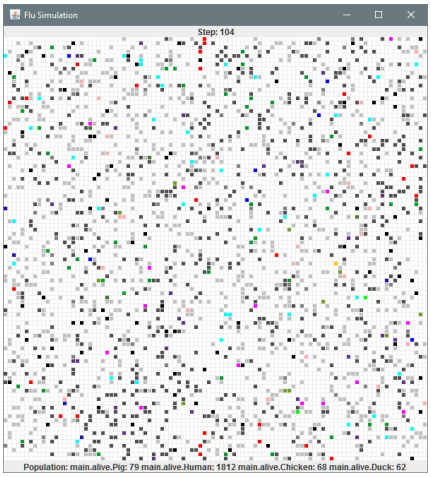
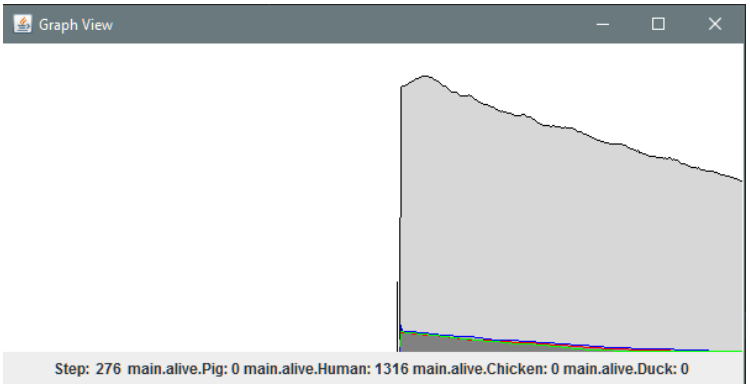


Image 1 : Représentation de la simulation



2. Graphique montrant l'évolution de la population des espèces

GRIDVIEW

Pour l'affichage graphique nous utilisons les classes déjà présentent dans le projet « Rabbits and Fox » de David J. Barnes et Michael Kölling. Nous avons adapté leur travail afin de l'adapter aux besoins de notre projet.

Dans la classe GridView, qui gère directement l'affichage de la simulation sous forme d'arène avec des rectangles de couleur, nous avons modifié la méthode getColor() afin de modifier les couleurs de chaque entité selon son état (Classe State). Pour cela nous avons une classe Colors qui définit les couleurs par défaut pour chaque état d'une espèce donnée.

Par exemple :

```
HumanColors.put(State.HEALTHY, Color.BLACK);
HumanColors.put(State.SICK, Color.LIGHT_GRAY);
HumanColors.put(State.CONTAGIOUS, Color.GRAY);
HumanColors.put(State.RECOVERING, DARK_GRAY);
```

Ainsi lors de l'exécution de la simulation on peut voir facilement l'évolution de l'épidémie.

GRAPHVIEW

Nous avons aussi modifié l'affichage de la fenêtre GraphView, construite dans la classe du même nom. Le graphique n'affichait que deux courbes, au lieu des quatre que nous voudrions observer. Après modification le graphique peut afficher les courbes et les conteurs de chaque espèce de la simulation, avec une couleur différente.

3. PARTIE SIMULATION

LES DIFFERENTES ESPECES

Les espèces sont au nombre de 4 dans cette simulation. Toutes les classes les représentant héritent de la classe Alive, qui les définit abstraitement. Chaque espèce a ses propres caractéristiques mais ont des méthodes en commun, notamment act(), qui détermine ce que doit faire l'animal ou humain à un moment donné.

Nous nous sommes basé sur les classes Animal, Rabbit and Fox du projet de David J. Barnes et Michael Kölling pour construire les notre.

La liste des attributs que nous avons gardés :

- Field : l'arène dans laquelle se trouve l'animal ou humain
- Location : la case où se trouve l'animal ou humain

Nous avons rajouté les attributs suivant :

- Resistance : La capacité d'un individu à résister à une maladie
- State : l'état dans lequel se trouve l'individu (En bonne santé, malade etc...)
- Disease : la maladie dont l'individu est porteur
- Speed : la vitesse de déplacement dans la simulation
- Immunities : Une liste des maladies indiquant pour laquelle l'individu est immunisé (HashMap)
- Age : l'âge de l'individu (attribut privé non présent dans Alive)
- nbDays : un compteur pour le nombre de jours écoulés, notamment utilisé lors de la période d'incubation (attribut privé non présent dans Alive)

Ces différents attributs nous permettent de créer une simulation qui se veut la plus réaliste possible.

ACTIONS DES ESPECES

Dans la méthode act() présente chez chaque Alive, nous faisons agir l'individu. Dans un premier temps nous augmentons son âge, pour le côté réaliste. Ensuite nous regardons si l'individu peut faire des enfants, et si oui ils sont placés sur des cases adjacentes au parent.

Ensuite nous appelons la méthode changeState(). Dans cette méthode nous regardons l'état de l'individu et modifions ses attributs selon celui-ci. Par exemple pour une personne malade, on regarde si elle a fini sa période d'incubation, dans ce cas elle devient contagieuse et on doit donc changer son état (setState()), sinon on augmente son compteur de jours et on passe à l'étape suivante.

```
case SICK:
    if (nbDays < getDisease().getIncubationTime()) {
        nbDays++;
    } else {
        setState(State.CONTAGIOUS);
        nbDays = 0;
    }
    break;
```

Dans la cas d'une personne saine, c'est-à-dire quand son état est HEALTHY, on doit regarder si l'un de ses voisins dans la simulation n'est pas contagieux. Si c'est le cas, on regarde si la personne devient malade, et ainsi on change son état en SICK.

Si une personne succombe suite à une maladie, on la retire de la simulation avec la méthode setDead().

3. VARIANTES

Nous avons implanté deux variantes dans notre projet.

La première est l'utilisation de l'âge comme représentation plus réaliste de la simulation. Un individu trop âgé meurt de vieillesse et on ne peut pas faire d'enfants avant un certain âge (BREEDING_AGE).

La deuxième variante consiste à faire muter la maladie de manière aléatoire, grâce à la méthode mutation() présente dans la classe Disease.