Projet modélisation individu-centrée

Modèle SIR et variations

Mise à jour : Mars 2017

Mots clés: *simulation par agents, épidémiologie*

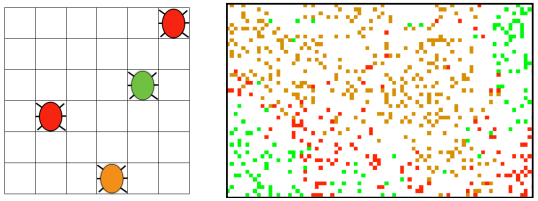
Rendez-vous sur la page http://pages.isir.upmc.fr/~bredeche/Teaching/3v681 pour télécharger le code source Python à partir duquel vous pourrez faire les exercices ci-dessous.

Ce projet prolonge l'exercice 2 vu en TP. Celui-ci est repris partiellement dans la première partie de l'énoncé ci-dessous, et sa réalisation sera prise en compte dans le projet. La seconde partie est un prolongement de cet exercice.

Vous pouvez utiliser la version graphique du code vu en TP. Vous pouvez la faire tourner sous Windows si elle ne fonctionne pas sous Linux. La version texte, donnée en TP fonctionne sur les deux systèmes. Vous devez utiliser Python 2.x

Epidémiologie: le modèle SIR — Reprise partielle de l'exercice 2 du TP

On s'intéresse à la simulation de la virulence d'un virus dans une population plus ou moins dense, à partir d'un ou plusieurs individus infectés. Chaque individu est placé dans l'environnement et bouge au hasard (une case dans une des quatre directions voisines, nord-sud-est-ouest). Il ne peut y avoir qu'un individu par case.



Il s'agit d'un modèle SIR ou un individu sain est en **vert**, un individu infecté est en **rouge** et un individu guéri/mort est en **orange**. Un individu sain le reste tant qu'il n'est pas à coté d'un individu infecté (voisinage: nord-sud-est-ouest). Un individu reste infecté pendant N itérations (on prendra N=20 pour commencer). Un individu guéri/mort ne bouge pas et reste éternellement dans cet état.

QUESTION 0: Le fonctionnement du programme permet de programmer le comportement d'un agent. A cet fin vous modifierez **uniquement** la fonction **step(self)**, qui défini le comportement d'un agent et permet de décider de l'action à faire et de l'état de l'agent.

QUESTION 1: programmez le déplacement de l'agent. C'est à dire un déplacement au hasard sur une case libre autour de l'agent. Attention à ne pas sortir de l'écran! (cela provoquerait une erreur)

QUESTION 2: programmez le changement d'état de l'agent. En fonction de l'état courant (et du voisinage dans le cas "sain"), l'état sera mis à jour (ou non). Un agent reste infecté pendant N itérations -- vous pouvez utilisez la variable ***infectionCountdown*** pour calculer le temps d'infection *(la variable est initialisée à 20)*.

Epidémiologie: extension du modèle SIR — exercice original

On souhaite maintenant étudier le modèle SIR que l'on vient de programmer. Pour commencer, on va prendre les paramètres suivants:

Taille de l'environnement: 80x50

Nombre d'agents au début de la simulation: 600 agents sains, 1 agent infecté.

QUESTION 1: Etudiez l'effet de la densité, c'est à dire du nombre d'individus dans l'environnement, sur le nombre d'individus finalement infectés. Pour cela vous devrez lancer votre programme pour plusieurs tailles de population, entre 10 et 1000, afin de mettre en évidence un éventuel seuil de percolation dans la diffusion de l'épidémie. Pour chaque taille de population, vous devrez lancer plusieurs simulations pour obtenir une bonne approximation du comportement moyen *(p.ex. 10)*.

Vous devrez tracer sur un graphique vos résultats. Ce graphe figurera la taille de la population initiale en abscisse, et la taille de la population survivante (état "S") après stabilisation (ie. lorsqu'il n'y a plus que des agents "S" ou "R") en ordonnée. Chaque simulation sera donc un point dans ce graphe.

Remarque:

1. A vous de choisir les tailles de population que vous souhaitez tester.

2. Pour générer les données du graphe, vous devrez modifier le code afin d'afficher pour chaque état le nombre d'agents concernés (cf. remarque suivante). Ce graphe montrera donc un ensemble de points, chaque point résumant une simulation pour une taille de population donnée. Il y aura donc plusieurs points pour chaque taille de population (correspondant à plusieurs runs).

3. Une manière simple pour récupérer les données d'un run et de mettre à jour la fonction *step()* avec le code ci-dessous (attention, il ne s'agit pas de *step(self)*, mais bien de *step()*). Cette nouvelle fonction reprend la précédente mais affiche à chaque étape le numéro de l'iteration, le nombre d'agents sains, infectés puis "guéris". Vous devrez pour chaque simulation attendre que le système se soit stabilisé pour prendre en compte les données (ie. les points du graphe représentent des simulations après stabilisation). Le choix vous est laissé libre pour faire votre graphe (Excel, libreoffice, python avec matplotlib, etc.)

```
def step():
  shuffledIndexes = [i for i in range(len(agents))]
  nbSains = nbInfected = nbRecover = 0
  random.shuffle(shuffledIndexes)
  for i in range(len(agents)):
      agents[shuffledIndexes[i]].step()
      if agents[shuffledIndexes[i]].state == "S":
          nbSains = nbSains + 1
      elif agents[shuffledIndexes[i]].state == "R":
          nbRecover = nbRecover + 1
      elif agents[shuffledIndexes[i]].state == "I":
          nbInfected = nbInfected + 1
  print iterations,",",nbSains,",",nbInfected,",",nbRecover
  return
```

QUESTION 2: Reprenez les paramètres par défaut donnés en début d'exercice (population initiale: 600 agents sains + 1 infecté), et modifiez le code pour les agents (1) continuent à bouger même dans l'état "R" et (2) qu'un agent dans l'état "R" retourne dans l'état "S" après *n* itérations. Il s'agit du modèle dît "SIRS".

Vous devez maintenant chercher un éventuel seuil de percolation pour *n*, tel que l'épidémie se résorbe (ie. convergence de la simulation vers un état stable ou tous les agents sont sains) ou non (ie. pas de convergence car ré-infections constantes).

Pour présenter ces données, vous pouvez reprendre le graphe fait à la question précédentePuisque les simulations peuvent ne pas converger, vous fixerez un nombre d'itérations maximum (suffisamment grand) pour arrêter la simulation. Vous tracerez cette le nombre d'agents infectés en ordonnée.