Modélisation et simulation individu-centrée

Feux de forêt, épidémiologie

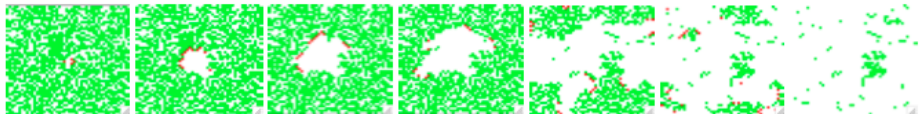
Mise à jour : Mars 2017

Mots clés: *automates cellulaires, simulation par agents*

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.

Les feux de forêt

On s'intéresse à la simulation du développement d'un feu de forêt. Pour mémoire, il s'agit d'un automate cellulaire 2D ou une cellule ne devient active que si un seul de ces voisins est actif (voisinage de von Neumann). Pour réaliser cet automate, on vous demande de télécharger le code source *exercice_forestfire.py* -- ce code source est incomplet, et annoté afin que vous sachiez ou répondre pour chaque question.



Les règles de mise à jour sont les suivantes (avec leur interprétation):
- une cellule blanche reste blanche
  - ie. un emplacement vide reste vide
- une cellule verte reste verte sauf si au moins une des cellules voisines est rouge
  - ie. un arbre prend feu si au moins un de ces voisins est en feu (droite, gauche, haut ou bas).
- une cellule rouge devient blanche
  - ie. un feu meurt en une itération

QUESTION 0: Lisez attentivement les commentaires "COMMENT CA MARCHE" et "PREMIER PAS" du code source.
- Vous modifierez **uniquement** les fonctions init() et step().
  - La fonction init() n'est appelée qu'une seule fois au tout début de l'exécution du programme
  - La fonction step() est appelée à chaque itération du programme pour mettre à jour le monde
- Remarques techniques:
  - La simulation de feux de forêt utilise une méthode dîte "double-buffer". C'est à dire que l'état courant de l'environnement est construit à partir de l'état précédent. A chaque instant, deux tableaux sont en mémoire (l'un contient l'état courant du monde, et l'autre contient l'état précédent)
  - Les fonctions setPixel et getPixel différents. setPixel écrit un pixel dans l'état courant du monde, getPixel lit la valeur d'un pixel dans l'état précédent du monde.
- Vous pouvez, si vous le souhaitez, modifier les "variables globales" pour changer les paramètres de la simulation (vitesse d'affichage, taille du monde)

QUESTION 1: Dans init(), initialisez l'environnement afin que chaque case contienne soit un arbre (=vert), soit rien (=blanc). Si vous lancez le programme, vous verrez votre écran clignoter.

QUESTION 2: Dans init(), ajoutez un arbre en feu (ie. une case rouge)

QUESTION 3: Dans step(), recopiez l'ensemble du tableau précédent (fonctions getPixel et setPixel)

QUESTION 4: Dans step(), étendez la question précédente pour mettre à jour (si nécessaire) l'état des cases contenant un arbre (ie. si un voisin est en feu, alors l'arbre prend feu).

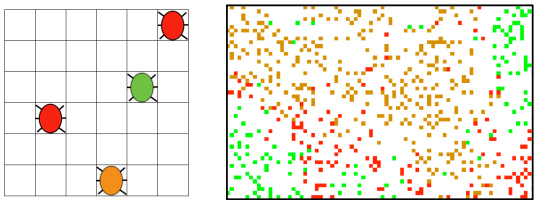
QUESTION 5: Dans step(), étendez la question précédente pour afficher dans le terminal le nombre d'arbres vivants, et le nombre d'arbre en feu.

QUESTION 6: Lancer plusieurs fois le programme pour estimer le seuil de percolation (il faut lancer plusieurs simulations pour chaque densité). Vous pouvez utilisez un tableur pour montrer vos résultats.
Astuce: vous devez afficher le nombre d'arbres restants lorsque tous les feux sont éteints.

QUESTION 7: (a) Modifiez votre modélisation pour prendre en compte les 8 (et non 4) voisins d'un arbre lors de la propagation d'un feu (b) Modifiez votre modélisation pour ajouter, pour chaque case vide, une probabilité pour qu'un arbre repousse.

Epidémiologie

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 est le même que précédemment, à ceci prêt que maintenant vous allez 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).*

QUESTION 3: modifiez le programme pour que les agents continuent à se déplacer lorsqu'il sont "guéris". Modifiez le programme pour qu'un agent "guéri" redevienne sain après quelques temps (ex.: 30 itérations).