TD Netlogo 1

Objectif

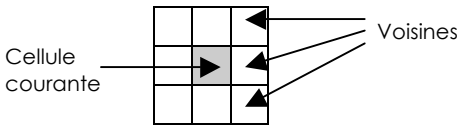
Le but de ce premier TD est de prendre en main l'environnement de la plate-forme Netlogo, et de programmer des premiers modèles simples en faisant bien la distinction entre ce qui concerne l'observer, les patchs et les tortues.

Les automates cellulaires

Les automates cellulaires constituent une classe de systèmes dans laquelle une grille de cellules évolue de cycle en cycle en fonction de règles définies au niveau de chaque cellule et spécifiant quel doit être l'état de la cellule au cycle suivant en fonction de son état courant et de l'état de ses plus proches voisines. A chaque cycle, toutes les cellules recalculent leur nouvel état puis elles changent toutes d'état simultanément, de manière synchrone.
Dans la plate-forme Netlogo, nous considérerons les patchs comme une grille de cellules dont l'état sera recalculé à chaque cycle.

Le jeu de la vie

Le jeu de la vie, proposé par J. Conway dans les années 1970 est l'exemple le plus connu et le plus étudié de ce type de systèmes. Dans cet exemple, chaque cellule ne peut se trouver à chaque instant que dans l'un des 2 états **on** (allumée) ou **off** (éteinte). Lorsqu'elle est allumée, la cellule ne restera allumée au cycle suivant que si 2 ou 3 de ses 8 voisines sont elles-mêmes allumées. Lorsqu'elle est éteinte, la cellule ne s'allume au cycle suivant que si exactement 3 de ses 8 voisines sont allumées.

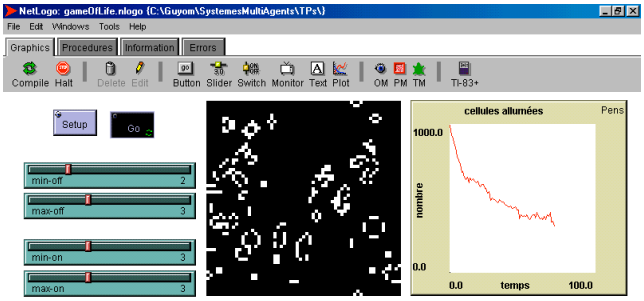


Travail demandé

- **Setup**
  - Ajouter une variable pour les patchs qui indique si le patch est dans l'état on ou off
  - Ecrire la procédure `setup` qui initialise aléatoirement chaque patch dans l'état on ou off et l'affiche en blanc ou noir suivant les cas
  - Ajouter à l'interface un bouton **Setup** qui exécute la procédure `setup`
- **Go**
  - Ecrire la procédure `go` qui demande aux patchs de :
    - compter le nombre de leur voisins dans l'état on (voir `sum…of` ou `count…with`) et `neighbors`)
    - calculer leur nouvel état
    - se réafficher en blanc ou noir suivant l'état
  - Ajouter à l'interface un bouton **Go** qui exécute la procédure `go` de manière répétitive
- **Observation**
  - Observer les différents types de comportements de groupes de cellules ;
  - Ecrire une procédure qui permette à l'utilisateur de modifier l'état des cellules grâce à la souris (voir `mouse-down?`, `mouse-xcor`, `mouse-ycor`, `patch`)
  - Ajouter un bouton pour activer ou désactiver la fonction de dessin
- **Expérimentation**
  - Ajouter des sliders pour pouvoir paramétrer les conditions de changement d'état des cellules ; tester différentes règles ;
  - En s'inspirant du modèle « Wolf Sheep Predation », ajouter un graphique pour tracer le nombre de cellules allumées à chaque cycle ;
  - Expérimenter l'outil « Behavior Space »

Application à un système de vote

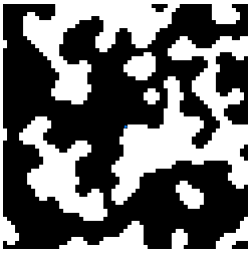
Appliquer le modèle précédant à un modèle de vote : on considère que la couleur d'une cellule correspond à un choix de vote. Pour savoir comment voter, une cellule compte le nombre de ses voisines qui ont voté d'une certaine manière. Elle vote alors de la même manière que le majorité des cellules, ou elle maintien son vote précédant en cas d'égalité.



TD Netlogo 2

Objectif

Le but de TD est de commencer à programmer des tortues et de voir comment elles peuvent interagir entre elles et avec leur environnement. Ce TD s'appuie sur le TD précédent, qui constitue l'environnement dans lequel vont évoluer les tortues. On partira notamment du système de vote qui se stabilise en produisant des motifs analogues à ceux de la figure ci-dessous :



La création du monde !

La première étape consiste à initialiser les patchs sous la forme ci-dessus, c'est-à-dire intégrer dans la procédure `setup` une dizaine de cycles de l'automate cellulaire défini au TD précédent (voir la commande `repeat`)

Ma première tortue !

On va maintenant manipuler des tortues. On souhaite dans un premier temps créer des tortues ayant un déplacement complètement aléatoire

- créer une première tortue (voir la commande `crt`), lui associer la forme « turtle » et la couleur rouge
- définir une procédure `move` qui donne à la tortue un mouvement aléatoire : à chaque cycle, la tortue change son orientation d'un angle choisi aléatoirement entre –45° et + 45° (voir `lt`, `rt` et `random`), puis avance d'un pas.
- Ajouter un slider `nb-turtles` qui définit le nombre de tortues à créer

Ma première tortue marine !

On souhaite maintenant que les tortues tiennent compte de la couleur des patchs sur lesquels elles se déplacent. On souhaite en particulier que les tortues ne se déplacent que sur les zones coloriées en blanc. Pour ce faire, on définit les règles suivantes :

- une tortue se trouvant sur un patch noir doit avancer droit devant elle
- une tortue sur un patch blanc peut changer son orientation comme précédemment mais elle ne pourra avancer que si le patch devant est également blanc (voir le mot-clé `of`, ainsi que `patch-ahead`). Dans le cas contraire, elle devra faire un demi-tour et n'avancera pas. De cette manière, la tortue ne sort jamais de la zone blanche.

- Adapter la procédure `move` pour prendre en compte ces règles

Tortues terrestres et tortues marines !

On souhaite distinguer deux types de tortues, suivant qu'elles se déplacent sur les zones blanches ou les zones noires. Les tortues blanches ne pourront se déplacer que sur les patchs noirs tandis que les tortues noires ne pourront se déplacer que sur les patchs blancs.

- ajouter un slider `%-white` définissant le pourcentage de tortues blanches
- créer 2 espèces de tortues (voir `breed`), `tortue-marine` et `tortue-terrestre`. Les premières sont noires et se déplacent sur fond blanc. Les secondes sont blanches et se déplacent sur fond noir.
- Modifier la procédure `move` en lui transmettant un argument `white?` qui vaudra 1 lorsqu'on appellera la procédure pour des tortues blanches et 0 pour des tortues noires (voir `to`)

La guerre des tortues !

Chacune des 2 espèces de tortues souhaite étendre son territoire. Pour ce faire elles adoptent la stratégie suivante : si une tortue, au moment où elle essaye d'avancer d'un pas, se trouve face à un patch de la mauvaise couleur, elle le colorie de la bonne couleur avant de faire un demi-tour.

- adapter la procédure `move`
- examiner ce qui se passe pour différentes proportions de tortues de chaque espèce. Que peut-on observer ? Comment peut-on l'expliquer ?
- utiliser le BehaviorSpace pour examiner ce qui se passe lorsque cette proportion varie : faire varier le pourcentage de tortues blanches de 0 à 100% et compter le nombre de patchs blancs après 100 cycles de simulation