M1 Info - Résolution Collective de Problèmes
TP 1 - Introduction à NetLogo

Tom Jorquera - tom.jorquera@irit.fr

Première partie
Présentation de NetLogo

NetLogo est un environnement de modélisation programmable basé sur le paradigme multi-agent. Son objectif est de fournir un environnement simple permettant de modéliser des systèmes complexes, et sa bibliothèque contient de nombreux exemples tirés de domaines très divers, aussi bien des sciences naturelles que des sciences sociales.
Développé en Java et Scala, NetLogo tourne sur la Java Virtual Machine.

Ressources utiles

Les ressources suivantes vous seront utiles tout au long des tps. N'hésitez pas à vous y référer en cas de besoin.

- Le site officiel de NetLogo (eng) :
  http ://ccl.northwestern.edu/netlogo/index.shtml
- Le guide de l'interface (eng) :
  http ://ccl.northwestern.edu/netlogo/docs/interface.html
- Le guide de programmation (eng) :
  http ://ccl.northwestern.edu/netlogo/docs/programming.html
- Le dictionnaire des primitives NetLogo (eng) :
  http ://ccl.northwestern.edu/netlogo/docs/dictionary.html

Prise en main

Pour s'initier au fonctionnement de NetLogo, nous allons tout d'abord voir la manipulation basique du logiciel sur un exemple simple. Nous allons tirer parti de sa bibliothèque et jouer un peu avec un modèle biologique bien connu : l'écosystème Prédateur-Proie (à ne pas confondre avec la technique de résolution de problème du même nom).

Dans NetLogo, ouvrez la bibliothèque de modèles (*File -> Models Library*, ou Ctrl+M). Dans le répertoire *Sample Models*, ouvrez le répertoire *Biology*, puis sélectionnez le modèle "Wolf Sheep Predation" et ouvrez-le.

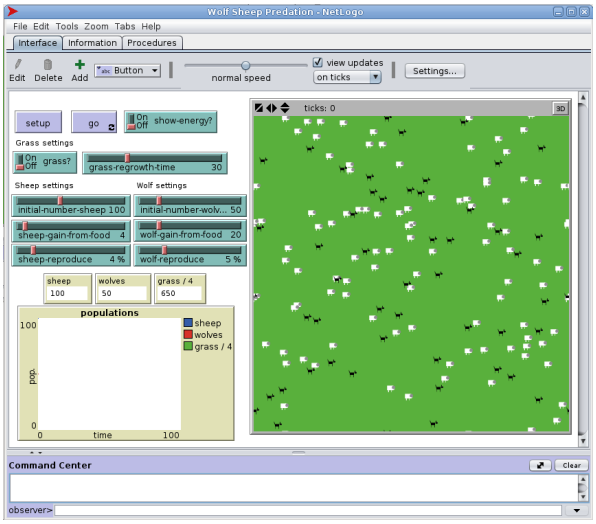

FIGURE 1 – L'interface NetLogo

Chaque modèle NetLogo dispose classiquement de trois vues : *Interface*, *Information* et *Procedures*. L'onglet Interface contient l'interface graphique qui permet à l'utilisateur de paramétrer et exécuter le modèle. L'onglet Information contient la description du modèle, de ses paramètres ainsi que d'autres informations intéressantes à son sujet. Enfin, l'onglet Procedures contient le code du modèle. Si vous observez le code du modèle que nous venons de charger, vous pourrez noter que celui-ci est relativement succin et explicite. En effet, nous verrons que NetLogo propose des primitives de haut niveau permettant de décrire simplement le comportement des agents.

Exécution du modèle

L'onglet Interface possède généralement au moins deux boutons : setup et go. Setup permet d'initialiser le modèle avec les paramètres définis par l'utilisateur, puis go permet de le lancer.

Amusez-vous à exécuter plusieurs fois le modèle. Si vous changez certains paramètres, pensez que vous devez utiliser le bouton reset pour que le changement soit pris en compte. Pour remettre les paramètres par défaut, il suffit de recharger le modèle. Vous pouvez modifier la vitesse d'exécution du modèle à l'aide de la slidebar située en haut de l'interface.

Observation de la sensibilité du système à ses paramètres

1. Que constatez-vous quand vous exécutez le modèle avec les paramètres par défaut ? Le système vous parait-il stable ?
2. Activez maintenant l'option *grass ?*, qui gère la repousse de l'herbe. Quelles sont les conséquences de l'introduction de ce paramètre au niveau de l'évolution des différentes populations ? Comment pouvez-vous intuitivement expliquer cet effet ?
3. Modifiez maintenant les paramètres suivants : passez *sheep-gain-from-food* à 2 et *sheep-reproduce* à 1%. Sachant que ces paramètres représentent le strict minimum "viable", quels vont être selon vous les effets de ces changements sur la population de moutons ? Sur la population de loups ? Les différentes exécutions du modèle correspondent-elles à vos prévisions ?
4. Rétablissez maintenant les paramètres des moutons à leur valeur par défaut (gain d'énergie de 4 et taux de reproduction de 4%), puis passez le paramètre *grass-regrowth-time* à une valeur de 60. En vous basant sur vos précédentes observations, quels vont être selon vous les effets de ce changement sur la population de moutons ? Sur la population de loups ? Les différentes exécutions du modèle correspondent-elles à vos prévisions ?
5. Essayez de déterminer les paramètres adéquats pour obtenir des populations d'herbe, de moutons et de loups les plus stables possibles, tout en restant non-nulles.

Deuxième partie

Programmation sous NetLogo

Notions basiques de NetLogo

Le NetLogo est un dérivé du language Logo. Ses principales caractéristiques sont les suivantes :

- procédural : l'exécution d'un programme se structure en blocs procéduraux (de manière un peu similaire au language C par exemple), qui peuvent éventuellement prendre des paramètres. Certaines procédures, nommées *reporter procedures*, permettent de retourner une valeur (équivalent des fonctions).
- typage faible : les variables peuvent être utilisées pour stocker n'importe quel type de valeur. Il est possible d'assigner une valeur d'un autre type que celui initialement assigné à la variable.
- multi-agent : toute instruction du language NetLogo est exécutée par un ou plusieurs agents. Il existe quatre types d'agents aux particularités bien distinctes : l'observeur, les turtles, les patches et les links. Nous allons maintenant voir en détail les caractéristiques de ces types d'agents.

Les agents NetLogo

L'agent *observer*

L'agent observer permet d'observer la simulation et ses données dans leur ensemble et peut donner des commandes aux autres types d'agents. Par défaut c'est lui qui exécute les commandes que vous donnez au modèle. Cet agent n'a pas de représentation graphique dans le modèle. Il ne peut y avoir qu'un seul observer

Quelques exemples de procédures associées à cet agent :

- *create-turtles/crt* <number> : permet de créer des agents turtles
- *clear-turtles/ct* : detruit tous les agents turtles

Les agents *turtles*

Ces agents sont les seuls dotés d'une capacité à se déplacer manière autonome. Par défaut le modèle ne contient aucun agent turtle, ils sont explicitement crées par l'observer (avec la commande create-turtles) ou par d'autres agents (avec la commande hatch pour les agents turtles et la commande sprout pour les agents patches).

Quelques exemples de procédures associées à ces agents :

- *forward/fd* <number> : fait avancer l'agent du nombre de pas spécifié
- *left/lt* <number> : fait tourner l'agent du nombre de degrés spécifié vers la gauche
- *right/rt* : fait tourner l'agent du nombre de degrés spécifié vers la droite
- *patch-here* : renvoie le patch sur lequel se trouve l'agent turtle
- *hatch* <number> : crée autant d'agents turtles que demandé, ces agents hériteront de toutes les caractéristiques de leur parent (y compris la position)
- *xcor* et *ycor* : renvoient les coordonnées actuelles de l'agent turtle
- *set color* [R G B] : change la couleur de l'agent turtles

Les agents *patches*

Les agents patches sont les agents qui constituent le fond sur lequel les agents turtles évoluent. A la différence des agents turtles, ces agents ne peuvent se déplacer et leur nombre est fixé. Un agent patch est caractérisé par ses coordonnées.

Quelques exemples de procédures associées à ces agents :

- *turtles-here* : renvoie l'ensemble des agents turtles présents sur le patch
- *sprout* <number> : crée autant d'agents turtles que demandé
- *pxcor* et *pycor* : renvoient les coordonées de l'agent patch
- *set pcolor* [R G B] : change la couleur de l'agent patch

Les agents *links*

Les agents links servent à représenter un lien autres deux autres agents. Dans ce premier TP nous ne nous attarderons pas sur ce type particulier d'agent.

> Note - Bien que nous ayons ici présenté les commandes comme étant rattachées à un type d'agent, une partie d'entre elles peuvent être appelées par des agents d'autre type. Par exemple un agent turtle peut utiliser la commande set pcolor, ce qui aura pour effet de directement modifier la couleur de l'agent patch sur lequel il est situé. En pratique, il est rarement nécessaire de se préoccuper de si la commande est bien adressée à un agent du bon type, NetLogo se charge d'interpréter au mieux vos instructions.

Développement d'un modèle de Swarm

Nous allons a présent commencer à développer un modèle simplifié de swarm sous NetLogo. Pour cela, créez un nouveau modèle (Files -> New, ou Ctrl+N).

Vos premières procédures

Comme vous l'avez vu précédemment, les modèles NetLogo possèdent généralement au moins deux procédures : setup, qui initialise le modèle, et go, qui l'exécute. Commencez par vous rendre dans l'onglet Procedures de votre nouveau modèle.

La déclaration d'une procédure se fait sous la forme suivante :

```
to <nom_procedure>
...
end
```

Nous allons commencer par définir la méthode setup comme suit :

```
to setup
        clear-all ;remet le modèle à zero
        create-turtles 10 ;;cree 10 agents turtle
end
```

Cette procédure permettra de créer dix agents turtles.

> Note - le caractère *;* sert à débuter un commentaire, il n'est pas nécessaire pour finir la ligne. Bien qu'un seul *;* suffise, les commentaires sont traditionnellement indiqués par *;;*

A présent nous allons définir la procédure go, dans laquelle nous allons faire déplacer nos agents ainsi crées.

```
to go
        ask turtles [
                left random 10 ;;tourne entre 0 et 10 degres vers la gauche
                right random 10 ;;tourne entre 0 et 10 degres vers la droite
                forward 1 ;;avance de 1
        ]
end
```

Cette procédure est légèrement plus complexe. La première ligne, "ask turtles", demande aux agents turtles d'exécuter le code entre crochets. L'instruction "left random 10" signifie "tourne vers la gauche d'une valeur aléatoire entre 0 et 10 degrés". A partir de là, vous aurez donc compris qu'il s'agit d'une procédure donnant aux agents une trajectoire relativement rectiligne, mais susceptible de varier légèrement.

Pour tester ces procédures, revenez à l'onglet Interface.
Au bas de la fenêtre, se trouve le Command Center, qui permet d'exécuter directement des commandes dans le modèle.

Assurez-vous que l'interprète de commande est bien sur *observer*>, si ce n'est pas le cas, cliquez dessus pour le modifier (ou appuyez sur la touche tabulation lorsque le curseur est dans l'interprète de commande).
Saisissez la commande setup et validez en appuyant sur entrée, vous devriez voir vos agents apparaitre sur le modèle. Faites ensuite de même avec la commande go pour les voir se déplacer d'un pas.

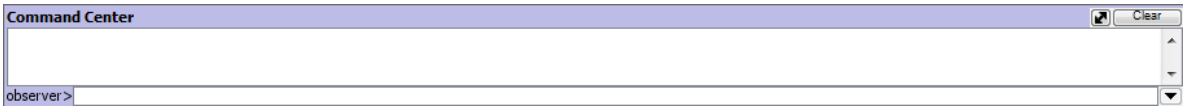

FIGURE 2 – Le Command Center

Bien entendu, cette méthode d'interaction n'est pas très pratique, c'est pourquoi nous allons maintenant ajouter des éléments à notre interface graphique afin de faciliter l'exécution des procédures.

Ajout d'éléments à l'interface graphique

L'ajout de nouveaux éléments d'interface se fait très simplement. Sous l'onglet Interface, un menu permet de choisir et d'ajouter les éléments prédéfinis dans NetLogo. Pour l'instant, contentez-vous de sélectionner l'objet bouton.

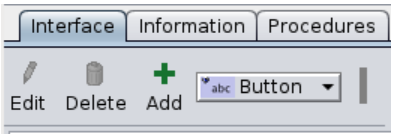

FIGURE 3 – palette d'ajout d'éléments graphiques

Lors de la création du boutons, insérez simplement le nom de la procédure à exécuter dans le champ *Commands*, sans remplir le champ *Display name* (NetLogo affichera alors le nom de la commande sur le bouton). Commencez par créer un bouton qui exécutera la procédure setup.

Nous allons ensuite créer le bouton pour exécuter la procédure go. Lorsque vous avez exécuté cette commande dans le Command Center, vous avez pu constater que les agents n'avancaient que d'un pas (une seule exécution de la procédure). Cependant il serait peu pratique d'avoir à faire des clics répétés sur le bouton juste pour faire avancer nos agents. C'est pourquoi lors de la création du bouton go, sélectionnez la case *Forever*, qui executera la commande en boucle (jusqu'à ce que l'utilisateur clique à nouveau sur le bouton). Les boutons forevers se distinguent par l'icone présente dans leur coin en bas à droite.

> Note - Rappelez-vous que vous pouvez régler la vitesse d'exécution de la simulation à l'aide du slider en haut de l'interface.

Nous allons maintenant ajouter un slider pour nous permettre de modifier le nombre d'agents crées lors du setup. Sélectionnez et ajouter un élément de type slider. Dans le champ *Global variable*, saisissez "nb-agents". Vous pouvez modifier les autres valeurs à votre convenance.
Retournez maintenant dans l'onglet Procedures et, dans la procédure setup, remplacez la ligne "create-turtles 10" par "create-turtles nb-agents".
Aucune autre modification n'est nécessaire. Vous pouvez constater que NetLogo vous permet de faire très simplement la liaison entre votre code et les éléments de votre interface graphique.

Utilisation des variables

Vous pouvez déclarer et réassigner des variables très simplement à l'aide des primitives *let* et *set*.

```
to exemple
        let mavariable 0 ;;assigne une valeur a une variable
        set mavariable "zero" ;;change le contenu d'une variable existante
end
```

Vous remarquerez rapidement que, grâce aux primitives haut niveau de NetLogo, vous aurez moins souvent besoin de déclarer des variables que dans d'autres langages procéduraux.

Troisième partie

A vous de jouer

Vous allez maintenant devoir développer vos propres procédures. N'oubliez pas d'utiliser les liens qui vous ont été fournis en début de sujet pour vous aider sur l'utilisation des fonctions. A l'aide des fonctions *set pcolor* et *scale-color*, générez un terrain dont les différences de couleur représentent un relief (dans l'image qui vous est donnée en illustration, une couleur plus claire de vert indique une hauteur plus élevée, une couleur sombre indique une hauteur basse).
Utilisez ensuite la fonction diffuse pour fournir à l'utilisateur un moyen d'ajuster les variations de relief (terrain plus accidenté ou plus homogène).

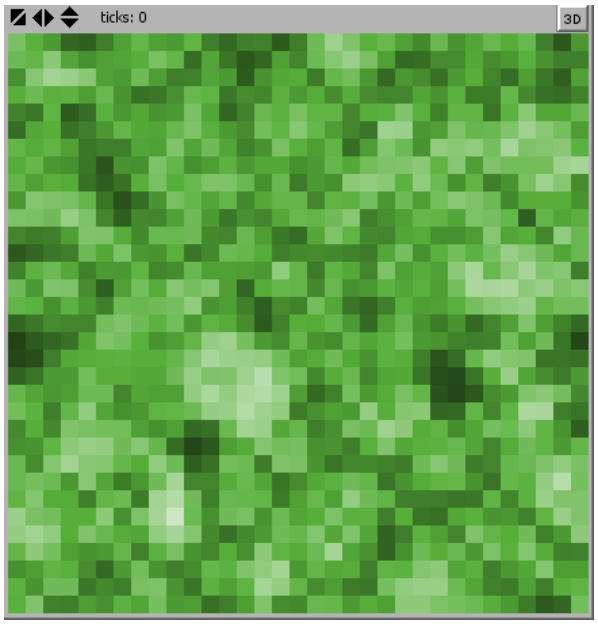

FIGURE 4 – Exemple de relief

Modifiez maintenant le code de vos agents turtle afin qu'ils essaient de se diriger vers le sommet le plus élevé.
Dans un premier temps, vous pouvez utiliser la fonction *uphill*.

> Note - Pour visualiser les trajectoires de vos agents, vous pouvez utiliser les commandes *pen-down* et *pen-up* qui servent respectivement à débuter et à arrêter le tracé du parcourt des agents turtles. Pensez que vous pouvez passer des commandes pendant l'exécution directement via le Command Center.

Vous vous apercevrez vite que se baser sur une méthode déterministe telle que *uphill* donne des résultats assez limités. A partir de la procédure qui vous est donnée en exemple ci-dessous, proposez une procédure pour que vos agents turtles sélectionnent l'agent patch sur lequel ils se déplacent de manière stochastique.

```
;;Exemple de procedure pour selectionner un agent turtle en
;;fonction de son son identifiant (le "who")
;;les agents ayant un identifiant plus grand on plus de chance d'etre selectionne
to exemple
        let candidats [ self ] of turtles ;;obtenir les candidats sous forme de liste
        let sumC sum map [ [ who ] of ? ] candidats ;;faire le total des criteres
        let rand random-float sumC ;;tirer un nombre entre zero et la somme des criteres

        ;;selection du patch
        let accumulator 0
        while [ rand > (accumulator + [ who ] of first candidats) ]
        ;;si le candidat n'a pas ete selectionne
        [
                ;; ajouter la valeur du critere du candidat a l'accumulateur
                set accumulator accumulator + [ who ] of first candidats

                ;;retirer le candidat de la liste des candidats
                set candidats but-first candidats
        ]

        ;;le candidat selectionne est en tete de liste des candidats restants
        let selectionne first candidats
end
```

FIGURE 5 – Exemple de tirage de Monte Carlo

Vous pourrez constater une nette amélioration dans le déplacement de vos agents. Cependant, une telle méthode reste très inefficace. Proposez maintenant vos propres solutions pour améliorer l'algorithme.

> Note - il est possible d'ajouter des variables à vos agents en plus des variables existantes (e.g. *color*) en utilisant les mots-clés *turtles-own*, *patches-own* et *links-own*. Par ailleurs vous pouvez déclarer des variables globales avec le mot clé *globals*. Documentez-vous sur ces mots-clés, et voyez si vous pouvez les exploiter dans vos améliorations.