Modélisation et simulation économiques
Chap 3 – NetLogo comme plateforme de Modélisation multi-agents (MMA)

Murat Yıldızoğlu
http ://yildizoglu.fr
Université de Bordeaux
GREΘA (UMR CNRS 5113)

NetLogo ?

- Plateforme de programmation avec interface graphique adaptée
- formée de trois panneaux :
  - **Interface** : une interface d'interaction avec le modèle et de représentation des résultats et des états du « monde » ;
  - **Information** : une interface d'information des utilisateurs du modèle ;
  - **Procedures** : une interface pour écrire les procédures du modèle (caractéristiques et comportements des agents et du systèmes, leurs interactions).
- Les choses importantes se passent dans « Procedures » et les choses *amusantes* dans « Interface »

Types d'agents

Un programme NetLogo peut être composé de différentes types d'agents :

- les tortues (*turtles*) : ce sont nos agents individuels, ils ont des caractéristiques, ils se déplacent dans le monde, ils naissent et meurent ; ils peuvent être de plusieurs types (*breeds*, consommateurs, firmes)
- les patches : ce sont les composantes spatiales du « monde » qui est modélisé ; les agents se déplacent sur les patches et les patches peuvent stocker des variables
- les liens (*links*) : un type particulier d'agent qui relie deux agents et qui est représenté comme une ligne dessinée entre ces agents. Ce lien peut être orienté ou non
- + le « monde » (*World*) : il correspond à la représentation spatiale (en 2d ou 3D) de l'environnement modélisé.

Autres éléments

Le modèle peut aussi contenir :

- des variables globales (accessibles par tous les agents) ;
- des ensembles d'agents (*agent sets*) : des listes d'agents de même type ;
- des procédures spécialisées (dont `setup` et `go`) ;
- des commentaires qui doivent suivre un point-virgule `;` :

```
; Ceci est un commentaire
```

Aide

Bien profiter de :

- L'aide de NetLogo (Menu Help, User manual) et surtout,
- du dictionnaire du NetLogo (Menu Help, Dictionary)

Listes

- Les listes occupent une place centrale dans les programmes NetLogo
- On les délimite avec des crochets et on sépare leurs éléments avec un espace (pas de virgule !) :
  - Une liste de trois nombres : [0.5 1 2]
  - Une liste contenant deux listes : [[0.5 1 2] [2.5 1.33 25]]
  - Une liste hétérogène : [0.5 [2.5 1.33 25] ["*Paul*" "*Jacques*" "*Aliye*"]]
  - Une liste vide : []
- On peut aussi les construire en utilisant l'instruction *list* : `(list 0.5 1 2)`
- On fait référence aux éléments d'une liste en utilisant l'instruction *item* : `item 0` [0.5 1 2] → 0.5
- Autres instructions utiles : `replace-item`, `fput`, `lput`, `sentence`, `map`, `reduce`, `sort`, `sort-by`

Déclaration des types de tortues et de leurs variables

- Déclaration des types d'agents :

```
breed [firms firm]
```

- Déclaration des variables individuelles des agents :

```
firms-own [
prix
profit
]
```

chaque firme a un prix et un profit

Déclaration et utilisation des variables

Les variables se déclarent

- Dans l'interface graphique, attachées aux contrôles (champ de texte, sélecteur de liste, curseur,etc.),
- Au début du programme, comme variable globale , dans la liste *globals* :

```
globals [
prix
nbAgents
]
```

- Au début du programme, comme variable appartenant à un type d'agent (turtle)

```
firmes-own [
cout
profit
mes-prix
]
```

Variables 2

- Comme variable locale dans une procédure ou un bloc de code : `let prix-initial`
- On change leur valeur avec l'instruction *set* :
  - `set prix-initial 0.5`
  - `set mes-prix [1.5 10 25]`
  - `set mon-nom "Toto"`
  - `set prix ( item 0 mes-prix )`
  - `set mes-prix ( replace-item 1 mes-prix 15 )`
    `show mes-prix` → [1.5 **15** 25]

Création des agents

- Pour pouvoir utiliser des agents d'un type donné,
- Il faut déclarer leur type dans le début du programme
  `breed [firms firm]`
- déclarer, le cas échéant, les variables qui vont leur être propres
- et créer autant d'agents de ce type que nécessaire au début du programme (en général dans `setup`)
  `create-firms 25`
- pour en créer 25 exemplaires, par exemple
- Les tortues (*turtles*) constituent un type qui est présent par défaut
  `create-turtles 25` ou
  `ct 25`

Une variable qui sait compter : ticks

- Les modèles NetLogo se déroulent dans un temps discret qui avance pas à pas.
- NL contient un compteur qui mémorise le nombre de périodes déjà exécutées : *ticks*
- Pour obtenir le nombre de ce compteur, on exécute l'instruction de même nom
  `let` cette-periode `ticks`
- Pour initialiser la valeur de ce compteur, on exécute l'instruction *reset-ticks* (en général à la fin de `setup`)
- et pour augmenter sa valeur d'une unité (en général à la fin des instructions correspondant au déroulement complet d'une période : go), on exécute l'instruction *tick*

Déclaration de procédures

- Les procédures sont des fonctions spécialisées qui regroupent certaines instructions qu'on est amenées à exécuter plusieurs fois
- NetLogo en possède deux types :
  - Les *commandes* (commands) qu'on utilise pour modifier les variables globales et/ou les variables qu'on leur passe en argument ;
  - Les *rapporteurs* (reporter) qui sont des méthodes qui retourne une valeur précise à la fin de leurs opérations, valeur à stocker dans une variable ou à utiliser dans un calcul (exemple : `replace-item` qui retourne la liste modifiée).

Commandes

- Leur déclaration commence avec l'instruction *to* et se termine avec l'instruction *end*

```
to maprocedure [ argument1 argument2 ]
instruction1
instruction2
⋮
end
```

- Plus tard dans le code, on lui fait appel comme instruction :

```
⋮
maprodecure 6 10
⋮
```

Rapporteurs

- Leur déclaration commence avec l'instruction *to-report* et se termine avec l'instruction *end*

```
to-report momrapporteur [ argument1 argument2 ]
let resultat
instruction1
instruction2
⋮
set resultat ...
return resultat
end
```

- Plus tard dans le code on lui fait appel pour affecter une valeur à une variable

```
set ma-variable monrapporteur 1 5
```

- `ma-variable` contient alors la dernière valeur calculée pour `resultat`

Répéter des instructions

- Répéter des instructions *n* fois :

  ```
  repeat n [ instructions ]
  ```

- Répéter des instructions tant qu'une condition est remplie (donne la valeur vraie – `true`) et s'arrêter quand la valeur devient `false` :

  ```
  while [ condition ] [ instructions ]
  ```

  ```
  while [any? other turtles-here] [ fd 1 ]
  ```

  La tortue avance (fd = forward) tant qu'il trouve d'autres tortues sur les patchs où elle se déplace.

Tenir compte de conditions

- Exécution de certaines instructions seulement si certaines conditions sont remplies
- On utilise les commandes *if* ou *ifelse* :
- `if condition [instructions]` : instructions exécutées uniquement si `condition` = `true`
- `ifelse condition [instructions1] [instructions2]` :
  si `condition = true` → `instructions1` exécutées;
  sinon → `instructions2` exécutées.

Faire des calculs avec les listes d'agents

- Le nom pluriel d'un type d'agents (`firmes`) permet de faire référence à une liste de tous les agents de ce type
- On peut alors leur demander d'éxécuter des instructions chacun à son tour, dans un ordre aléatoire

```
ask firmes [
fixe-production ; appel procédure déterminant la production
calcule-profit ; appel procédure qui calcule le profit
]
```

Faire des calculs avec les agents – 2

- On peut aussi calculer des statistiques sur une propriété des agents d'un certain type
- On utilise :
  `opérateur [variable-type] of nom-pluriel-agents`
  - Le profit moyen des firmes : `mean [ profit ] of firmes`
  - Le profit maximal parmi les firmes : `max [ profit ] of firmes`
  - Aussi minimum (`min`), médiane (`median`), écart-type (`standard-deviation`), variance (`variance`) d'une variable des agents.

Faire des calculs avec les agents – 3

- Aussi tout autre calcul utilisant une variable commune des agents du même type :
- `show [ who ] of firmes` => [0 3 2 1] (ordre aléatoire)
- `show sort [who] of firmes` => [0 1 2 3]
- `show sort [who * who] of firmes` => [0 1 4 9]
- `show sort [recettes - couts ] of firmes` => [100 110 115 150]

Structure–type d'un modèle NetLogo

Un modèle NetLogo *typique* est composé de plusieurs blocs :

- **Déclarations** des variables globales, des types d'agents et des variables spécifiques de chaque type d'agent ;
- Procédure `setup` :
  - **Initialisation** des variables globales (lecture automatique au début de chaque exécution à partir de l'interface graphique ou fixation de leur valeur) ;
  - **Création** des populations de chaque type d'agents et **initialisation** de leur variables individuelles ;
  - Initialisation du compteur de périodes et des sorties diverses (graphiques et autres) de la simulation ;
- Une procédure (`go`) qui regroupe toutes les opérations qui ont lieu pendant une « période » d'exécution du modèle et qui incrémente le compteur de périodes (`ticks`) ;

Structure–type d'un modèle NetLogo, suite

- Des **procédures complémentaires** diverses
  - qui gèrent les comportements des différents types d'agents ;
  - qui collectent des statistiques sur ces comportements et sur les résultats agrégés du modèle ;
  - qui actualisent les sorties, notamment les graphiques.
- Une **interface graphique** qui contient des éléments qui permettent à l'utilisateur
  - de fixer les valeurs des paramètres du modèle et
  - d'observer les sorties, période par période.
- On utilise des commentaires pour documenter le modèle :

```
; ceci est un commentaire
```

Structure–type 1 : Déclarations

- Déclaration des variables globales :

```
globals [
    nbMoutons
    herbe
]
;Les variables fixées dans l'interface
;sont automatiquement des variables globales
```

- Création des types d'agents

```
breed [moutons mouton]
```

- Création des variables individuelles des agents

```
moutons-own [energie]
```

chaque mouton a un niveau d'énergie

```
patches-own [compteur]
```

Structure–type 2 : Initialisation

- Préparer le modèle pour l'exécution (procédure *setup*)

```
to setup ; execute par le bouton « setup »
(initialisations, instructions)
end
```

- ou plus spécifiquement

```
to setup ; execution par le bouton « setup »
    ; Effacer tous les agents, le monde
    ; et les variables
    clear-all
    ; we start with tick = 0 (NL 5)
    reset-ticks
    ; (procédure) initialiser les patches
    setup-patches
    ; (procédure) initialiser tous les moutons
    setup-moutons
end
```

Structure–type 3

```
to setup-patches
    ask patches [ set pcolor green ]
end
to setup-moutons
    create-moutons nombre-moutons
    ; nombre-moutons ← l'interface
    ask moutons [ setxy random-xcor random-ycor ]
end
```

Structure–type 4 : Exécution des périodes

```
to go   ; execution par le bouton « go »
        ; tant qu'il est enfonce
    if ticks >= 500 [ stop ]
    ;arret apres 500 periodes
    bouger-moutons ; (procedure)
    manger-herbe   ; (procedure)
    verifier-morts ; (procedure)
    reproduire     ; (procedure)
    set nbMoutons count moutons
    repousser-herbe  ; (procedure)
    set herbe count patches with [pcolor = green]
    tick             ; incremente la periode
end
```

Structure–type 5 : Finalisation du modèle

- Pour compléter le modèle, il faut :
- écrire les procédures qui manquent
  - bouger-moutons ;
  - manger-herbe ;
  - verifier-morts ;
  - reproduire ;
  - repousser-herbe ;

Structure–type 6 : Interface graphique

- Créer l'interface graphique avec au moins
  - un champ pour fixer nombre-moutons ;
  - un champ pour fixer chacun des autres paramètres
  - un bouton « setup » pour lancer l'initialisation
  - un bouton « go » pour démarrer le déroulement de l'histoire (ticks)
- On peut la compléter par des graphiques pour observer l'évolution de
  - Quantité de moutons (nbMoutons) ;
  - Quantité d'herbe (herbe).

Autres procédures : bouger-moutons

```
to bouger-moutons
    ask moutons [
        right random 360
        forward 1
        set energie energie - 1
        ]
end
```

Autres procédures : manger-herbe

```
to manger-herbe
    ask moutons [
        if pcolor = green [
        set pcolor black
        set energie (energie + energie-from-grass)
        ;energie-from-grass ← Interface
        ]
    ]
end
```

Autres procédures : reproduire

```
to reproduire
    ask moutons [

        if energie > energie-naissance [
          set energie energie - energie-naissance

        hatch 1 [ set energie energie-naissance ]
        ]
    ]
end
;energie-naissance ← Interface
```

Autres procédures : verifier-morts

```
to verifier-morts
    ask moutons [
        if energie <= 0 [ die ]
    ]
end
```

Autres procédures : repousser-herbe

```
to repousser-herbe
    ask patches [

        if random 100 < 3 [ set pcolor green ]
    ]
end
```

Vision d'ensemble du modèle

Nous avons donc un modèle herbe–moutons avec :

- de l'herbe qui repousse aléatoirement ;
- des moutons qui la mange ;
- des moutons qui se déplacent aléatoirement ;
- des moutons qui meurent et qui se reproduisent.
- le reste du code $\rightarrow$ dans NetLogo

Initialisation et actualisation des graphiques

- Les graphiques pour observer l'évolution des variables se créent d'abord dans l'interface utilisateur ;
- NetLogo 5 actualise automatiquement les données représentées sur les graphiques :
  - Les commandes d'initialisation de plot (plot setup commands) et de crayons (pen setup commands) sont exécutées quand les commandes *reset-ticks* ou *setup-plots* sont exécutés dans les procédures *setup* ou *go* ;
  - Les commandes d'actualisation de plot (plot update commands) et de crayons (pen update commands) sont exécutées quand les commandes *reset-ticks* (dans *setup*), *tick* ou *update-plots* (dans *go*) sont exécutées.