Ce guide de programmation présente le langage NetLogo en détail pour concevoir vos propres simulations multi-agents.

Exercice 1 : Les agents

Les agents sont des entités qui exécutent des instructions. On distingue 4 types d'agents : les tortues (*turtles*), les tuiles (*patches*), les liens (*links*) et l'observateur (*observer*). Les tortues sont des agents qui se déplacent dans l'environnement. L'environnement, qui est bidimensionnel, est divisé en une grille de tuiles. Chaque tuile est un carré de « terrain » sur lequel les tortues peuvent se déplacer. Les liens sont des agents qui relient deux tortues. L'observateur n'est pas situé. Il supervise les tortues et les tuiles en leur transmettant des instructions. Dans l'état inital, il n'y a pas de tortues. L'observateur peut créer de nouvelles tortues. Les tuiles peuvent également créer de nouvelles tortues. Bien qu'elles soient immobiles, les tuiles peuvent également exécuter des instructions.

Q1. Les **tuiles** ont des coordonnées. Par défaut, la tuile aux coordonnées $(0, 0)$ est appelé l'origine. Les coordonnées des autres tuiles correspondent aux distances horizontales et verticales par rapport à celle-ci. Les coordonnées d'une tuile sont appelées `pxcor` et `pycor`. Quand vous vous déplacez vers la droite `pxcor` augmente. vous vous déplacez vers le haut, `pycor` augmente. Le nombre total de tuiles est déterminé par les paramètres `min-pxcor`, `max-pxcor`, `min-pycor` et `max-pycor`.

Q1.1. Démarrez NetLogo.

Q1.2. Cliquez sur le bouton `Settings...` dans l'onglet `Interface`.

Q1.3. Quelles sont les valeurs initiales de `min-pxcor`, `max-pxcor`, `min-pycor` et `max-pycor` ? Combien y a-t-il de tuiles ?

Q1.4. Créez une grille de 10×5 tuiles.

Q2. Les **tortues** ont également des coordonnées : `xcor` et `ycor`. Alors que les coordonnées d'une tuile sont toujours des entières, celles d'une tortue peuvent être des réels.

Q2.1. Créez 1 tortues dans chaque coin,via le centre de commande.

```
create-turtles 1 [ setxy 0 0 ]
create-turtles 1 [ setxy 9 0 ]
create-turtles 1 [ setxy 0 4 ]
create-turtles 1 [ setxy 9 4 ]
```

Q3. Les **liens** n'ont pas de coordonnées. Chaque lien a deux extrémités, et chaque extrémité est une tortue. Si l'une des tortues meurt, le lien meurt aussi. Un lien est représenté visuellement comme une ligne reliant les deux tortues.

Q3.1. Créez un graphe de connexion complet via la commande
`ask turtles [ create-links-with other turtles ]`.

Exercice 2 : Les procédures

Les commandes et les rapporteurs (`reports`) sont des instructions qui s'adressent aux agents. Une commande est une instruction qui doit être exécutée par un agent. Un rapporteur est une instruction qui retourne une valeur. Alors qu'une commande est préfixée par un verbe d'action (`create`, `die`, `jump`, `inspect` ou `clear`), un rapporteur est généralement un nom ou une expression nominale. Les commandes et rapporteurs du langage NetLogo, qui sont appelés primitives, sont répertoriés dans un dictionnaire.

Q1. Les commandes et les rapporteurs que vous définissez sont appelés des procédures. Chaque procédure a un nom, précédé du mot-clé `to` pour les procédures de commande et `to-report` pour les procédures de

rapporteur. Le mot-clé end marque la fin de la procédure. Les commandes et rapporteurs peuvent avoir des arguments.

Q1.1. Créez un nouveau modèle.

Q1.2. Ajoutez un bouton setup et un bouton go.

Q1.3. Ajoutez les 2 procédures suivantes :

```
to setup                              ;; Définir une procédure intitulée setup
  clear-all                           ;; Réinitialiser l'environnement
  create-turtles 10                   ;; Créer 10 tortues sur la tuile 0,0
  [ setxy random-xcor random-ycor ]   ;; Déplacer aléatoirement chaque tortue
  reset-ticks                         ;; Redémarrer l'horloge
end                                   ;; Terminer la définition de la procédure

to go
  ask turtles [                       ;; Demander aux tortues d'exécuter les commandes
    fd 1                              ;; avance (forward) d'un pas
    rt random 10                      ;; tourne à droite (right turn) aléatoirement
    lt random 10                      ;; tourne à gauche (left turn) aléatoirement
  ]
  tick                                ;; Incrémenter le tic d'horloge
end
```

Notez que :

— setup et go sont des procédures de commande définies par l'utilisateur ;
— clear-all, create-turtles, reset-ticks, ask, lt, rt et tick sont des commandes primitives ;
— random et turtles sont des rapporteurs primitifs. random retourne un entier aléatoire inférieur à son argument (ici entre 0 et 9). turtles rapporte l'ensemble de toutes les tortues ;
— les commandes clear-all et create-turtles ne peuvent être exécutées que par l'observateur alors que la commande fd ne peut être exécutée que par les tortues. La commande tick peut être exécutée par différents types d'agents.

Q1.4. Configurez et exécutez le modèle.

Q2. Les procédures peuvent avoir des arguments.

Q2.1. Modifiez la procédure go pour qu'elle appelle une autre procédure :

```
to go
  ask turtles [                       ;; Demander aux tortues d'exécuter
    draw-polygon 8 who                ;; dessinez un octogone dont la longueur
  ]                                   ;; des côtés est égale à son numéro
  tick                                ;; Incrémenter le tic d'horloge
end

to draw-polygon [num-sides len]       ;; Dessiner un polygône
  pen-down                            ;; baisser crayon
  repeat num-sides [                  ;; pour chaque côté
    fd len                            ;; dessiner un côté
    rt 360 / num-sides                ;; tourner à droite
  ]
end
```

Q2.2. Configurez et exécutez le modèle.

Q3. Définissez votre propre rapporteur.

Q3.1. Modifiez la procédure go pour qu'elle utilise un rapporteur :

```
to draw-polygon [num-sides len]       ;; Dessiner un polygône
  pen-down                            ;; baisser crayon
  repeat num-sides [                  ;; pour chaque côté
    fd len                            ;; dessiner un côté
    rt 360 / num-sides                ;; tourner à droite
    set label absolute-value xcor     ;; affichez |xcor|
]
end

to-report absolute-value [number]     ;; Retourner la valeur assolue d'un nombre
  ifelse number >= 0                 ;;  si le nombre est positif
    [ report number ]                ;; retourner le nombre
    [ report (- number) ]            ;; sinon son opposé
end
```

Q3.2. Configurez et exécutez le modèle.

Exercice 3 : Les variables

Les variables d'agent permettent de stocker des valeurs pour chaque agent. Une variable d'agent peut être une variable globale, une variable de tortue, une variable de tuile ou une variable de lien. Les variables globales appartiennent à l'observateur : il n'y a qu'une seule valeur à laquelle chaque agent peut y accéder. Pour une variable de tortue, chaque tortue a sa propre valeur pour cette variable. Il en va de même pour les variables de tuile et de lien.

Q1. Toutes les tortues et tous les liens ont une variable définie par défaut `color` alors que toutes les tuiles ont une variable `pcolor`.

Q1.1. Configurez le modèle.

Q1.2. Demandez à toutes les tortues de changer pour la couleur rouge.

La liste des variables d'agent par défaut est disponible ici. Par exemple, `xcor`, `ycor` et `heading` (qui indique la direction vers laquelle la tortue est tournée) sont des variables de tortue alors que `pxcor` et `pycor` sont des variables de tuile.

Q2. Vous allez définir votre propre variable.

Q2.1. Créez une variable globale en utilisant le mot-clé globals au début de votre code, e.g. `globals [nbPolygon]`.

Q2.2. Modifiez votre code pour incrémenter cette variable quand un nouveau polygone est tracé.

Q2.3. Dans l'onglet `Interface`, ajoutez un moniteur pour afficher la valeur de cette variable au cours de la simulation.

Q2.4. Configurez et exécutez le modèle.

Q3. Pour définir des variables de tortue, de tuile et ou de lien, utilisez les mots-clés `turtles-own`, `patches-own` et `links-own`.

Q3.1. Créez une variables de tortue intitulée `eloignement` qui représente la distance entre la tortue et l'origine puis affichez là sous la tortue à chaque tic.

Q3.2. Configurez et exécutez le modèle.

Q4. Les variables globales sont accessibles à tout moment par n'importe quel agent. De même, une tortue peut lire et modifier les variables de la tuile sur laquelle elle se trouve.

Q4.1. Changez la couleur de la tuile à chaque angle de chaque octogone.

Q4.2. Configurez et exécutez le modèle.

Q5. Pour lire la valeur d'un variable d'un autre agent, utilisez le mot-clé `of`.

Q5.1. Dans la centre de commande, saisissez `show [color] of turtle 5` puis `show [xcor + ycor] of turtle 5`.

Q6. Pour créer une variable locale à une procédure, utilisez la commande `let`. au début de la procédure.

Q6.1. Ajoutez la procédure suivante :

```
to swap-colors [turtle1 turtle2]              ;; Echanger la couleur de 2 tortues
  let temp [color] of turtle1                 ;; definir une variable locale
  ask turtle1 [ set color [color] of turtle2 ] ;; modifier la couleur d'une tortue
  ask turtle2 [ set color temp ]              ;; puis de l'autre
end
```

Q6.2. Modifiez la procédure go pour que à chaque tic d'horloge, 2 tortues choisies aléatoirement échangent de couleur (`one-of agentset` rapporte de manière aléatoire un agent).

Q6.3. Configurez et exécutez le modèle.

Exercice 4 : Tic d'horloge

Dans la plupart des modèles, le temps est discrétisé en tics d'horloge (`ticks`). Affiché au-dessus de la vue, le compteur de tics est accessible depuis le code grâce au rapporteur primitif `ticks`. La commande primitive `tick` incrémente ce compteur alors que `clear-all` le réinitialise comme tout le reste. Utilisez la commande `reset-ticks` au terme de la configuration pour démarrer le compteur.

Q1. Modifiez la procédure go pour que les tortues dessinent en alternance des carrés et des octogones.

Exercice 5 : La commande Ask

La commande Ask permet d'adresser des instructions aux tortues (`turtles`), aux tuiles (`patches`) et aux liens (`link`).

Tout le code à exécuter par les tortues doit être situé dans un « contexte » de tortue. Vous pouvez établir un contexte de tortue :

1. dans un bouton, en choisissant `turtles` dans le menu contextuel. Le code que vous placez dans le bouton sera exécuté par toutes les tortues ;
2. dans le centre de commande, en choisissant `turtles` dans le menu contextuel. Toutes les commandes que vous saisissez seront exécutées par toutes les tortues ;
3. en utilisant les commandes ask `turtles`, hatch, ou d'autres commandes qui établissent un contexte de tortues.

Il en va de même pour les tuiles, les liens et l'observateur, sauf que vous ne pouvez pas utiliser la commande ask avec l'observateur. Le code qui n'est pas à l'intérieur d'une commande ask s'adresse à l'observateur.

Un `agentset` est un ensemble d'agents qui peut contenir des tortues, des tuiles ou des liens, mais pas plus d'un type à la fois. Par exemple la primitive turtles contient l'ensemble de toutes les tortues, alors que patches contient toutes les tuiles et links tous les liens. Pour éviter les biais de simulation, l'ordre des agents dans un `agentset` est par défaut aléatoire et différent à chaque fois que vous l'utilisez. Par conséquence les instructions transmises par la commande ask sont exécutées par les membres de l'`agentset` dans un ordre aléatoire. Si vous souhaitez que vos agents réalisent des instructions selon un ordre déterminé, utilisez une liste des agents.

Q1. Voici un exemple de l'utilisation de la commande ask dans une procédure :

```
to setup
clear-all
create-turtles 100      ;; créer de manière pseudo-aléatoire 100 tortues
ask turtles
[ set color red         ;; changer la couleur des tortues
fd 50 ]                 ;; avance (forward) de 50 pas
ask patches
[ if pxcor > 0          ;; si la tuile est à droite de la vue
[ set pcolor green ] ]  ;; alors changer la couleur
reset-ticks
end
```

Quel est le resultat de ce code ?

Q1.1. Dans le menu `File`, créez un nouveau modèle (`New`).

Q1.2. Dans l'onglet `Interface`, créez un bouton intitulé `Configurer` qui exécute la procédure `setup`.

Q1.3. Dans l'onglet `Code`, copiez le code de la procédure `setup`.

Q1.4. Retournez dans l'onglet `Interface` et configurez votre modèle. Qu'est-ce que vous observez dans la vue ?

Q1.5. Dans le menu `File`, enregistrez votre modèle (`Save`).

Vous pouvez également utiliser la commande `ask` pour vous adresser individuellement aux tortues, aux tuiles ou aux liens pour exécuter des commandes en utilisant les primitives `turtle`, `patch`, `link` et `patch-at` comme dans l'exemple suivant :

```
to setup
  clear-all
  crt 3                          ;; crée 3 tortues
  ask turtle 0                   ;; dit à la première...
    [ fd 1 ]                     ;; ...avance
  ask turtle 1                   ;; dit à la seconde...
    [ set color green ]          ;; ...devient vert
  ask turtle 2                   ;; dit à la troisième...
    [ rt 90 ]                    ;; ...tourne à droite
  ask patch 2 -2                 ;; demande à la tuile (2,-2)...
    [ set pcolor blue ]          ;; ...devient bleu
  ask turtle 0                   ;; demande à la première tortue...
   [ ask patch-at 1 0            ;; ...de demander à la tuile à l'est
     [ set pcolor red ] ]        ;; ...de devenir rouge
  ask turtle 0                   ;; demande à la première tortue...
    [ create-link-with turtle 1 ] ;; ...de créer un lien avec la seconde
  ask link 0 1                   ;; demande à ce lien
    [ set color blue ]           ;; ...de devenir bleu
  reset-ticks
end
```

La primitive `turtle` prend en argument un entier, i.e. le numéro de la tortue. La primitive `patch` prend en argument les valeurs de coordonnées de la tuile, i.e. `pxcor` et `pycor`. La primitive `link` prend en argument les numéros des deux tortues qu'il relie. La primitive `patch-at` prend en argument les distances vis-à-vis de la tortue dans les directions x et y. Par exemple, la procédure `ask turtle 0 [ ask patch-at 1 0 [ ... ] ]` demande à la tortue numéro 0 de s'adresser à la tuile à l'est d'elle-même.

Q1.6. Dans l'onglet `Code`, copiez le code de la procédure `setup`.

Q1.7. Retournez dans l'onglet `Interface` et configurez votre modèle. Qu'est-ce que vous observez dans la vue ?

Quand vous adressez un ensemble de commandes à un ensemble d'agents, chaque agent exécute toutes les commandes, puis l'agent suivant les exécute toutes, etc. si vous écrivez :

```
ask turtles
[ fd 1
set color red ]
```

D'abord une tortue avance et devient rouge, puis une autre tortue avance et devient rouge, etc. Si vous l'écrivez de cette manière :

```
  ask turtles [ fd 1 ]
  ask turtles [ set color red ]
```

D'abord toutes les tortues avancent, ensuite elles deviennent rouge.

Exercice 6 : La primitive agentset

La primitive agentset permet de construire des ensembles d'agents qui contiennent certaines tortues, certaines tuiles ou certains liens, e.g toutes les tortues rouges, les tuiles dont la coordonnée pxcor est divisible par cinq, ou les tortues du premier quadrant qui sont sur une tuile verte ou connectées à la tortue numéro 0. Vous pouvez alors vous adresser à cet ensemble avec la commande ask. Une autre façon de procéder pour créer un ensemble de tortues consiste à utiliser les primitives :

- turtles-here avec les tortues sur une tuile ;
- turtles-at avec les tortues sur une autre tuile situées à des distances x et y ;
- turtles-on avec les tortues situées sur une autre tuile ou sur un ensemble de tuiles, voire sur la même tuile qu'une tortue ou un ensemble de tortues.

Q1. Commentez les ensembles d'agents dans le code suivant :

```
;; Toutes les autres tortues à l'exception de celle-ci
other turtles
;;
other turtles-here
;;
turtles with [color = red]
;;
turtles-here with [color = red]
;;
patches with [pxcor > 0]
;;
turtles in-radius 3
;;
patches at-points [[1 0] [0 1] [-1 0] [0 -1]]
;;
neighbors4
;;
turtles with [(xcor > 0) and (ycor > 0)
and (pcolor = green)]
;;
turtles-on neighbors4
;;
[my-links] of turtle 0
```

Q2. Pour un ensemble d'agents, vous pouvez utiliser les commandes :

- ask qui adresse des instructions aux agents ;
- any? qui retourne faux si l'ensemble est vide ;
- all? qui teste une condition sur tous les agents de l'ensemble ;
- count qui retourne le nombre d'agents dans l'ensemble.

Q2.1. Configurez votre modèle.

Q2.2. Dans le centre de commande, saisissez l'instruction `ask one-of turtles [ set color blue ]`. Quel est le résultat ?

Q2.3. Saisissez l'instruction `ask one-of patches [ sprout 1 ]`. Répétez cette instruction. Quel est le résultat ? Pour plus de détail consultez la documentation[1].

Q2.4. Saisissez l'instruction `ask max-one-of turtles [xcor] [ die ]`. Répétez cette instruction. Quel est le résultat ?

Q2.5. Saisissez l'instruction `show mean [xcor] of turtles`. Quel est le résultat ?

Les ensembles `no-turtles`, `no-patches` et `no-links` sont vides. L'égalité entre des ensembles d'agents peut être testé avec les opérateurs = et `!=`. Si un agent est membre d'un ensemble, `member?` retourne vrai.

1. https://ccl.northwestern.edu/netlogo/docs/dictionary.html#sprout

Exercice 7 : Ensembles spéciaux d'agents

Les ensembles d'agents `turtles` et `links` ont un comportement spécial, car ils contiennent toujours les ensembles de toutes les tortues et de tous les liens. Par conséquent, ces ensembles peuvent croître.

Q1. Dans l'onglet Code, déclarez la variable globale g grâce à l'instruction `globals [ g ]`. Dans le centre de commande, saisissez les commandes suivantes et notez les valeurs de retour. Comment interprétez-vous ces résultats ?

```
observer> clear-all
observer> create-turtles 5
observer> set g turtles
observer> print count g

observer> create-turtles 5
observer> print count g

observer> set g turtle-set turtles
observer> print count g

observer> create-turtles 5
observer> print count g

observer> print count turtles
```

Exercice 8 : Espèce

Vous pouvez définir différentes espèces (`breeds`) de tortues ou de liens qui se comportent différemment (e.g. des moutons et des loups, des rues et des trottoirs). Vous définissez les espèces de tortues à l'aide du mot-clé `breed`, en haut de l'onglet Code, avant toute procédure. Par exemple,

```
breed [wolves wolf]
breed [sheep a-sheep]
```

Pour faire référence à un membre de l'espèce, utilisez le nom de l'espèce au singulier, comme le fait le rapporteur primitif `turtle`. Certaines commandes et rapporteurs contiennent le nom de l'espèce au pluriel, comme `create-wolves`. D'autres ont le nom de l'espèce au singulier, comme `wolf`. L'ordre de déclaration est également l'ordre de superposition dans la vue. Par exemple, les moutons apparaîtront au-dessus des loups.

Quand vous définissez une espèce telle que sheep, un ensemble d'agents de cette espèce automatiquement créé. Les primitives `create-sheep`, `hatch-sheep`, `sprout-sheep`, `sheep-here`, `sheep-at`, `sheep-on`, et `is-a-sheep` sont disponibles. Utilisez `sheep-own` pour définir de nouvelles variables de mouton. La variable de tortue `breed` contient son ensemble d'agents de la même espèce. Par exemple, `if breed = wolves [ ... ]` permet de tester l'espèce de l'agent. Un agent peut changer d'espèce. Par exemple, la commande `ask one-of wolves [ set breed sheep ]` transforme un loup pris au hasard en mouton.

Q1. Dans le menu `File`, ouvrez la bibliothèque de modèles (`Models Library`) et sélectionnez `Breeds and Shapes Example`. Examinez le code du modèle, configurez et exécutez.

Exercice 9 : Espèce de lien

Les espèces de liens sont très similaires aux espèces de tortues, cependant, il y a quelques différences. Lorsque vous déclarez une espèce de liens, vous devez indiquer s'il s'agit d'une espèce de liens dirigés ou non dirigés en utilisant les mots-clés `directed-link-breed` et `undirected-link-breed`. Par exemple,

```
directed-link-breed [streets street]
undirected-link-breed [friendships friendship]
```

Une fois que vous avez créé un lien dirigé, vous ne pouvez plus créer de liens non dirigés et vice versa. Vous pouvez, cependant, avoir des liens dirigés et non dirigés dans le même environnement, mais pas dans la même espèce. Contrairement aux espèces de tortues, le nom singulier de l'espèce est nécessaire pour les espèces de liens, car de nombreuses commandes et rapporteurs de liens utilisent le nom singulier, comme `<link-breed>-neighbor?`.

De nouvelles primitives sont alors disponibles :

— pour une espèce de lien dirigée, e.g. `create-street-from`, `create-streets-from`, `create-street-to`, `create-streets-to`, `in-street-neighbor?`, `in-street-neighbors`, `in-street-from`, `my-in-streets` `my-out-streets`, `out-street-neighbor?`, `out-street-neighbors` et `out-street-to`.

— pour une espèce de lien non dirigé, e.g; `create-friendship-with`, `create-friendships-with`, `friendship-neighbor?`, `friendship-neighbors`, `friendship-with` et `my-friendships`.

Plusieurs espèces de liens peuvent déclarer la même variable `-own`, mais une variable ne peut pas être partagée entre une espèce de tortues et une espèce de lien. Comme avec les espèces de tortues, l'ordre de déclaration est également l'ordre de superposition dans la vue. Utilisez `<link-breeds>-own` pour déclarer séparément les variables de chaque espèce de liens. Vous pouvez changer l'espèce d'un lien avec la commande `set breed`. Cependant, vous ne pouvez pas changer un lien typé en un lien non typé.

Q1. Dans le menu `File`, ouvrez la bibliothèque de modèles (`Models Library`) et sélectionnez `Link Breeds Example`. Examinez le code du modèle, configurez et exécutez le.

Exercice 10 : Les listes

Une liste peut contenir n'importe quel type de valeur : un nombre, une chaîne de caractères, un agent, un ensemble d'agents, ou une autre liste. Le dictionnaire NetLogo contient une section qui énumère toutes les primitives relatives aux listes.

Pour créer une liste constante, mettez simplement les valeurs que vous voulez entre crochets, e.g. `set mylist [2 4 6 8]`, `set mylist [[2 4] [3 5]]` ou `set mylist []`.

Pour créer à la volée une liste, utilisez le rapporteur `list`, e.g. `set random-list list (random 10) (random 20)`, `set (list random 10)` ou `set (list random 10 random 20 random 30)`.

Le rapporteur `replace-item` retourne une liste en modifiant un élément d'une autre liste. Il prend trois arguments. Le premier indique quel élément de la liste doit être modifié. `0` signifie le premier élément, `1` signifie le deuxième élément, etc. Par exemple,

```
set mylist [2 7 5 Bob [3 0 -2]]
; [2 7 5 Bob [3 0 -2]]
set mylist replace-item 2 mylist 10
; [2 7 10 Bob [3 0 -2]]
```

Pour ajouter un élément à la fin (respectivement au début) d'une liste, utilisez le rapporteur `lput` (respectivement `fput`). Par exemple,

```
set mylist lput 42 mylist
; [2 7 10 Bob [3 0 -2] 42]
```

Pour supprimer le dernier (respectivement premier) élément d'une liste, utilisez le rapporteur `but-last` (respectivement `but-first`). Par exemple,

```
set mylist but-last mylist
; [2 7 10 Bob [3 0 -2]]
set mylist but-first mylist
; [7 10 Bob [3 0 -2]]
```

Ces rapporteurs peuvent être appelés récursivement. Par exemple,

```
set mylist (replace-item 3 mylist
(replace-item 2 (item 3 mylist) 9))
; [7 10 Bob [3 0 9]]
```

La commande `foreach` permet d'exécuter une ou plusieurs commandes sur chaque élément d'une liste. Elle prend argument une liste et une commande ou un bloc de commandes. Par exemple,

```
foreach [1 2 3] show
; 1
; 2
; 3
foreach [2 4 6]
[ n -> crt n
show (word "created " n " turtles") ]
; created 2 turtles
; created 4 turtles
; created 6 turtles
foreach [1 2 3] [ steps -> ask turtles [ fd steps ] ]
foreach [true false true true] [ should-move? -> ask turtles [ if should-move? [ fd 1 ] ] ]
```

Le rapporteur `map` fonctionne de manière similaire. Il prend argument une liste et une commande ou un bloc de commandes. Par exemple,

```
show map round [1.2 2.2 2.7]
;; affiche [1 2 3]
show map [ x -> x < 0 ] [1 -1 3 4 -2 -10]
;; affiche [false true false false true true]
show map [ x -> x * x ] [1 2 3]
;; affiche [1 4 9]
```

D'autres primitives comme `filter`, `reduce` et `sort-by` permettent de traiter des listes. Dans d'autre cas, vous devrez utiliser des boucles avec les commandes `repeat` ou `while`, voire une procédure récursive.

Pour que vos agents exécutent des instructions dans un ordre fixe, vous devez établir une liste d'agents en utilisant les primitives `sort` et `sort-by` qui prennent un ensemble d'agents en argument :

- `sort` sur un ensemble d'agents composé de tortues retourne une liste de tortues triées selon l'ordre croissant de leur numéro ;
- `sort` sur un ensemble de tuiles retourne une liste de tuiles triées de gauche à droite et de haut en bas ;
- `sort` sur un ensemble de liens retourne une liste triée par ordre croissant selon l'origine puis la destination, en cas d'égalité les liens sont triés par espèce dans l'ordre de déclaration ;
- `reverse sort` trie par ordre décroissant ;
- `sort-by` trie selon un critère, par exemple la taille avec la commande `sort-by [ [a b] -> [size] of a < [size] of b ] turtles`.

Vous pouvez alors demander à la liste d'agents d'exécuter des commandes :

```
foreach sort turtles [ the-turtle ->
  ask the-turtle [
     ...
  ]
]
```

Exercice 11 : En conclusion

Vous connaissez désormais les principes et les principaux éléments de syntaxe du langage Netlogo pour pouvoir définir vos propres modèles multi-agents.

Pour aller plus loin, vous pouvez consulter :

- le guide d'interface qui décrit chaque élément de l'interface et sa fonction ;
- le guide de programmation qui explique comment écrire des procédures ;
- le dictionnaire qui répertorie et détaille toutes les primitives du langage.