PROGRAMMATION ORIENTÉE AGENT : NETLOGO

EN CONSTRUCTION !

28/01/13

Modélisation/simulation de phénomènes collectifs

http://ccl.northwestern.edu/netlogo/

NetLogo

- Un environnement de programmation pour la modélisation/ simulation de phénomènes collectifs
- Adapté à la modélisation de systèmes complexes composés de nombreux agents inter-agissant en parallèle
- Simulations disponibles en sociologie, biologie, médecine, physique, chimie, mathématiques, informatique, économie, psychologie sociale, …
- Possibilité de créer ses propres modèles et de les simuler

NetLogo : un monde d'agents

- Constitué d'agents qui peuvent agir
- Les activités des différents agents s'exécutent simultanément
- 3 types d'agents : patch - turtle - link
- + un unique observateur du monde !

Le Monde = grille 2D

- Grille constituée de patches (carrés)
- Un patch est repéré par ses coordonnées
- Origine du monde (0,0) au centre de la grille
- Dimension de la grille
  - min-pxcor <= … <= max-pxcor
  - min-pycor <= … <= may-pxcor
  - Par défaut min=-16 et max=+16 ; 33*33=1089 patches
- Le monde des patches est, par défaut, une grille circulaire
- On peut changer le nombre de patches et la nature de la grille avec le bouton « setting »

3 types d'agents

1) Patch (contenir)
   - Position fixe (int pxcor, int pycor)
   - Peut créer une turtle
   - Peut contenir plusieurs turtles
2) Turtle (bouger)
   - Agent réactif qui se déplace dans le monde
   - Positionner sur un patch (float xcor, float ycor)
   - Peut changer de patch
3) Link (relier)
   - Connecte deux turtles
     `(turtle end1, turtle end2)`
4) Et un Observer unique (observer)
   Unique
   Ne fait pas partie du monde
   Peut créer une turtle

Agents réactifs = turtles

- Création par :
  - l'observateur
  - un patch

- Coordonnées dans le monde :
  - `xcor`, `ycor` de type `float`
  - une turtle est situé sur un patch de coordonnées de type `int`
  - Un patch peut contenir plusieurs turtles

Commandes

- Une commande définit les actions à exécuter par les agents
  - pré-défini : primitive Netlogo
  - à définir par le programmeur : procédure

- Une procédure est définie par : `to unNom … end`

```
to setup
    ca     ;; clear all
    crt10 ;; create 10 turtles
end
```

Primitives Netlogo

- Commandes prédéfinies dans NetLogo
- Exemples :
  - `ca` (clear all)
  - `crt` (create turtles)
  - `lt` (left turn), `rt` (right turn), `fd` (forward)
  - `set` (affectation de variable)
  - `ask`, ...
- peuvent avoir des valeurs en entrée :
  - `crt 100`
  - `rt 50` ...

NetLogo avec le « Commande Center »

- Choisir « New » dans le fichier menu
- Créer des turtles :
  - `observer> crt 100`
- Les turtles créées sont regroupées au centre, pour les voir il faut augmenter le rayon du cercle :
  - `turtles> fd 10`

  Ou
  - `turtles> fd (random 10)`
- Pour passer de la commande `observer>` à `turles>` il faut cliquer sur `observer` et choisir à partir du menu l'option turtles

Écrire une procédure NetLogo

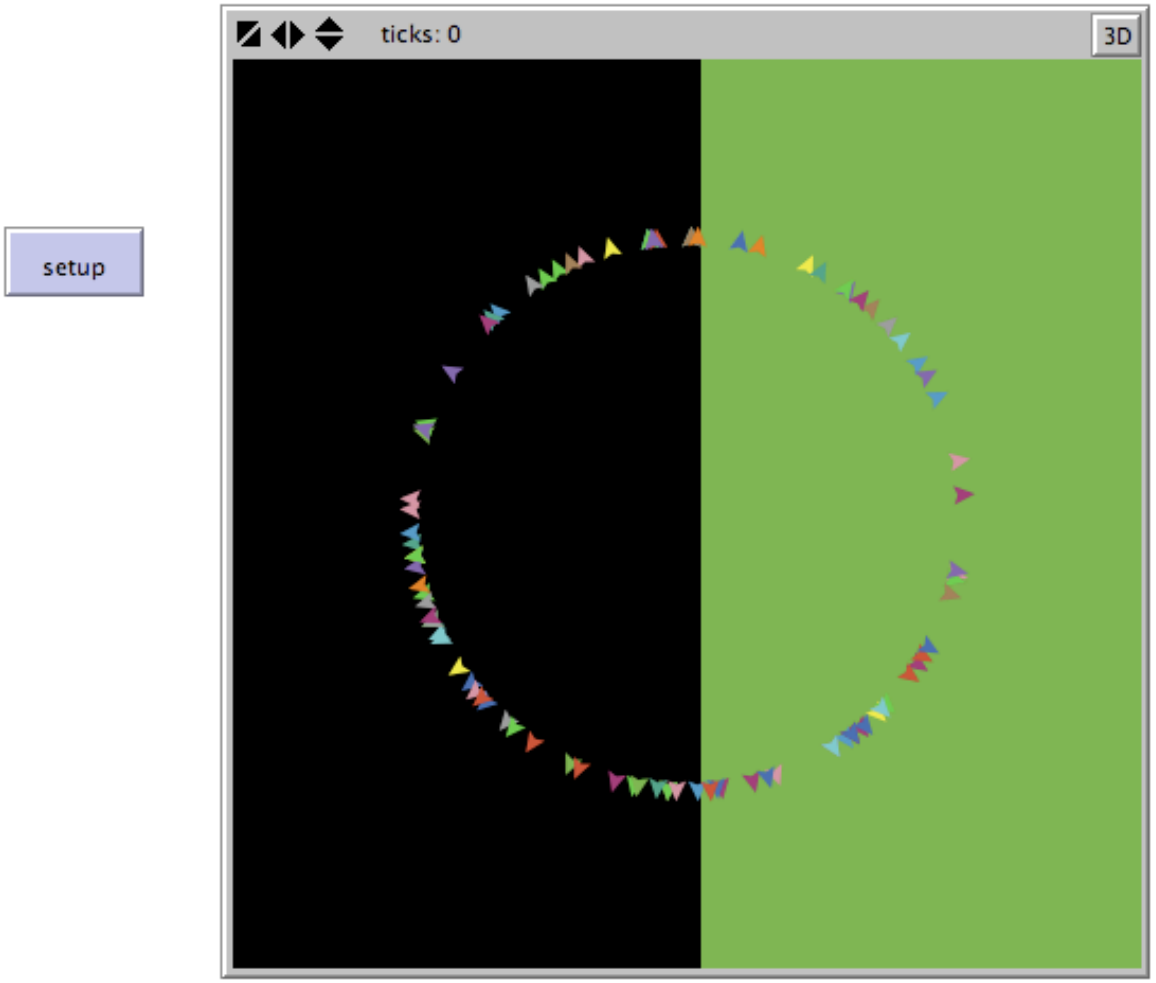
- Créer un bouton « setup » :
  - cliquer sur l'icône bouton dans la barre d'outil
  - Puis cliquer dans un endroit de la fenêtre graphique où vous voulez placer votre bouton
  - Une fenêtre de dialogue s'ouvrira, écrire « setup » dans le boxe « code », et cliquer sur le bouton « ok »
  - Le bouton s'affichera sur l'interface graphique

Exemple 1: procédure « setup »

- Avant de pouvoir actionner le bouton « setup »
  - Cliquer sur l'option « Procedures »
  - Saisir le code suivant :

```
to setup
  ca
  crt 100
  ask patches [
    if (pxcor > 0) [set pcolor green]
  ]
  ask turtles [
    fd 10
  ]
end
```

Exécution : actionner le bouton



Exemple 2 : Procédure « setup »

- Cliquer sur l'option « Procédures »
- Remplacer le code par :

```
to setup
  ca
  crt 100
  ask turtles [
   setxy (random 100) (random 100)
     set shape "circle"
  ]
end
```

Analyser la procédure « setup »

- **ca**: clear-all, effacer l'écran et initialise toutes les variables à 0
- **crt 100**: créer 100 tortues qui se localisent à la position (0, 0)
- **ask turtles**[ ]: oblige les tortues à exécuter de manière indépendante, les instructions contenues entre les [ ]
- **set shape « circle »:** les turtles seront sous forme d'un cercle. On peut changer cette forme en consultant outil dans le menu bar, option « turtles shapes »

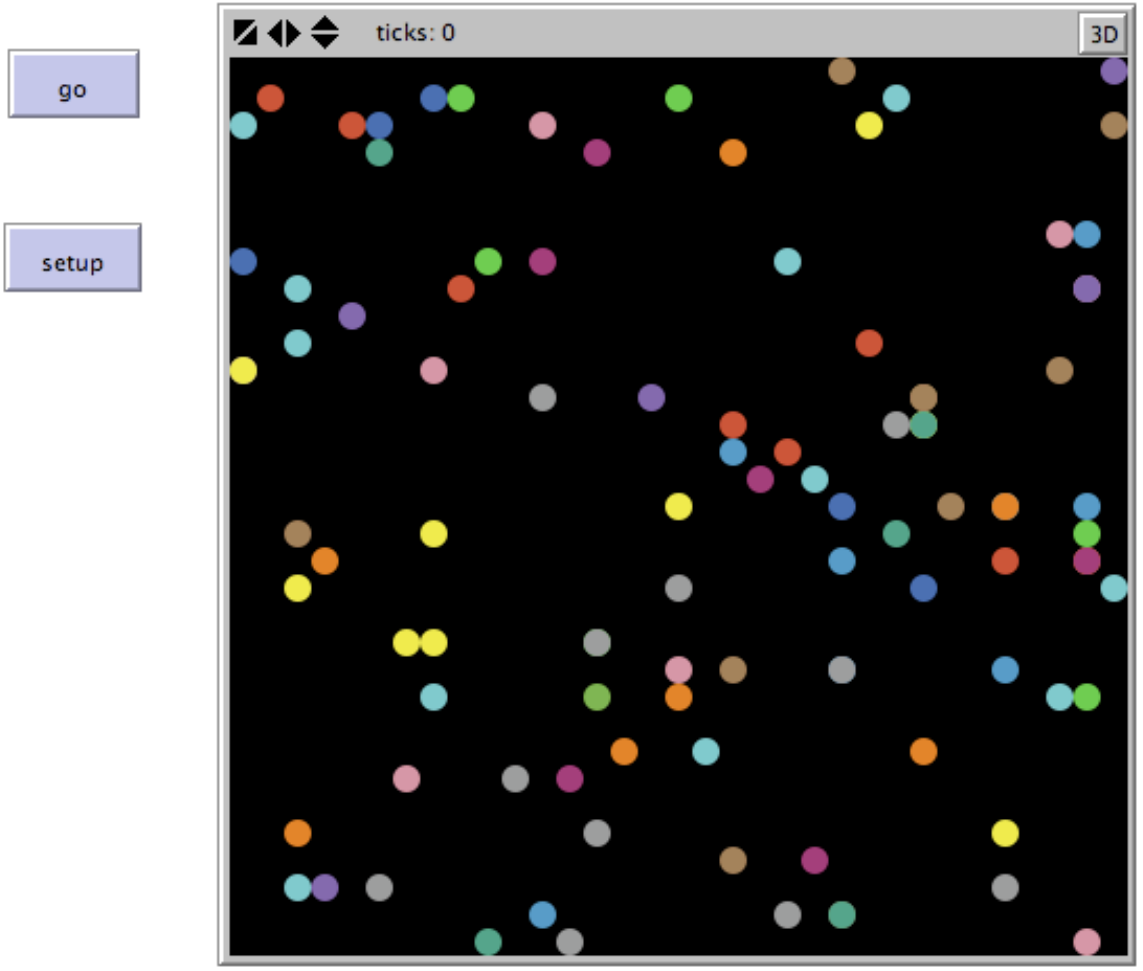
Procédures « go » et « bouge »

- Créer un bouton « go »
- Dans procédure écrire le code :

```
to go
  bouge
end


to bouge
    ask turtles[
        set heading random 360
       fd 1
     ]
end
```

Exemple d'exécution



procédure vs. reporter

- Un reporter est une commande qui calcule et retourne un résultat
- Utiliser `to-report` à la place de `to` pour commencer
- Dans le corps d'un reporter, utiliser `report` pour indiquer la valeur à retourner

```
to-report absolute-value [number]
  ifelse number >= 0
    [ report number ]
    [ report (- number) ]
end
```

Variables Netlogo

- Variable global
  - unique exemplaire
  - Accessible par chaque agent
- Variable turtle
  - Une valeur pour chaque turtle
- Variable patch
  - Une valeur pour chaque patch
- Variable link
  - Une valeur pour chaque link

- Variables pré-définies
  - Variable turtle : `color`
  - Variable patch : `pcolor`
  - Variable link : `end1`

Variables turtle pré-définies

- `breed`
- `color`
- `heading`
- `hidden?`
- `Label`
- `label-color`

`pen-mode`
`pen-size`
`shape`
`size`
`who`
`xcor`
`ycor`

Variable turtle pré-définie : heading

- Indique la direction pour chaque turtle
  - Compris entre 0 et 360
  - 0 est le Nord, 90 est l'Est, …
- On change cette valeur pour faire « tourner » une turtle sur elle-même

```
set heading 45       ;; turtle is now facing northeast
set heading heading + 10 ;; same effect as "rt 10"
```

Variables patch pré-définies

- `pcolor`
- `plabel`
- `plabel-color`
- `pxcor`
- `pycor`

Variables link pré-définies

- `breed`
- `color`
- `end1 end2`
- `hidden?`
- `label`
- `label-color`
- `shape`
- `thickness`
- `tie-mode`

Définir ses propres variables

- Variable globale :
  - Ajouter un switch ou un slider
  - Utiliser le mot-reservé `globals` au début du code :

```
globals [score ]
```

- Variable turtle, patch ou link
  - Utiliser turtles-own , patches-own and links-own :

```
turtles-own [energy speed ]
patches-own [friction ]
links-own [strength ]
```

- Utiliser la commande set pour l'affectation
- Par défaut la valeur est zéro

Conditions d'accès!

- Une variable globale peut être lue et modifiée à tout moment par chaque agent
- Une turtle peut lire et modifier une variable patch si elle est positionnée sur ce patch. Par exemple, le code :

```
ask turtles [set pcolor red ]
```

  permet à chaque turtle de « colorier » son patch en rouge

- Les variables patch sont donc partagées par les turtles, on ne peut donc pas avoir une variable turtle et une variable patch de même nom

Utiliser `of` pour limiter les accès


```
;; prints current color of turtle with who number 5

show [color] of turtle 5
```

```
;; prints the sum of the x and y coordinates of
;; turtle with who number 5

show [xcor + ycor] of turtle 5
```

Variable locale

- Définir et utiliser uniquement dans le contexte d'une procédure particulière (ou d'une partie de cette procédure)
- Création :
  - Utiliser (n'importe où dans la procédure) la commande **`let`**
  - Si début d'une procédure : existe dans toute la procédure
  - Si entre crochets : existe uniquement à l'intérieur des crochets

```
to swap-colors [turtle1 turtle2]
  let temp [color] of turtle1
  ask turtle1 [ set color [color] of turtle2 ]
  ask turtle2 [ set color temp ]
end
```

Commande `ask`

- Pour donner des « ordres » aux turtles, patches, et links
- Tout code exécuté par les turtles doit être localisé dans un « turtle context »
- Comment établir un « turtle context » :
  - dans un button, en choisissant "Turtles" dans le menu popup
  - dans le Command Center, en choisissant "Turtles" dans le menu popup
  - en utilisant ask turtles

Commande **ask**

- Idem pour les patches, les links, et l'observer
- On ne peut pas utiliser `ask` pour l'observer
- Tout code qui n'est pas dans un `ask` est par défaut un code observer

```
to setup
  clear-all
  crt 100                ;; create 100 turtles with random headings
  ask turtles
    [ set color red      ;; turn them red
      fd 50 ]            ;; spread them around
  ask patches
    [ if pxcor > 0           ;; patches on the right side
        [ set pcolor green ] ]  ;; of the view turn green
end
```

Les reporters turtle, patch, link, patch-at

- Le reporter `turtle` prend un entier en entrée, et retourne la turtle avec ce numéro (who number)
- Le reporter `patch` prend les valeurs pour pxcor et pycor, et retourne le patch avec ces coordonnées
- Le reporter `link` prend deux entrées, le « who numbers » des deux turtles qu'il connecte
- Le reporter `patch-at` prend les distances relatives (en x et y) en entrée, et retourne le patch désigné

```
ask turtle 0                    ;; ask the first turtle
    [ ask patch-at 1 0          ;; ...to ask patch to the east
      [ set pcolor red ] ]      ;; ...to become red
```

ask vs. ask … ask

first one turtle moves and turns red, then …

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

- … another turtle moves and turns red, and so on

first all of the turtles move. After …

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

- … they have all moved, they all turn red

ask vs. ask-concurrent


```
ask turtles [ fd 5 ]
ask-concurrent turtles [ fd 5 ]
```

- avec `ask`, la première turtle avance de 5 pas, puis la deuxième avance de 5 pas, etc

- avec `ask-concurrent`, toutes les turtles avancent de un pas, puis elles avancent toutes d'un deuxième pas, etc

ask-concurrent est équivalent à :

```
repeat 5 [ ask turtles [ fd 1 ] ]
```

Agentsets

- Ensemble non ordonné d'agents qui peut contenir turtles, patches ou links (sans mélange)
- = ou != pour comparer deux agentsets
- member? Pour savoir si un agent appartient à un agentset
- Agentsets totaux:
  - Le reporter `turtles` renvoie toutes les turtles
  - Le reporter `patches` renvoie tous les patches
  - Le reporter `links` tous les links

Comment définir ses propres agentsets ?


```
other turtles ;; all other turtles

other turtles-here ;; all other turtles on this patch

turtles with [color = red] ;; all red turtles

turtles-here with [color = red] ;; all red turtles on my patch

patches with [pxcor > 0] ;; patches on right side of view

turtles in-radius 3 ;; all turtles less than 3 patches away

patches at-points [[1 0] [0 1] [-1 0] [0 -1]]
;; the four patches to the east, north, west, and south

turtles with [(xcor > 0) and (ycor > 0) and (pcolor = green)]
;; turtles in the first quadrant that are on a green patch

turtles-on neighbors4 ;; turtles standing on my neighboring four patches
[my-links] of turtle 0 ;; all the links connected to turtle 0
```

Utiliser un agentset

- `ask`
  - Pour donner des ordres à tous les agents dans un agentset
- `any?`
  - Pour savoir si un agentset est vide
- `all?`
  - Pour savoir si chaque élément d'un agentset vérifie une condition
- `count` : retourne le cardinal d'un agentset

Utiliser un agentset

- `one-of` : **choisir un agent au hasard dans un agentset**

```
ask one-of turtles [ set color green ]
```

- `max-one-of` **ou** `min-one-of` : **trouver l'agent le meilleur ou le pire selon un critère**

```
ask max-one-of turtles [age] [die ]
```