

# Programmation orientée agents #1

v 1.4

M1 S2 - Université de Montpellier II

HMIN108-Master Informatique– spécialité Imagina-Aigle

**Jacques Ferber**

**[www.lirmm.fr/~ferber](http://www.lirmm.fr/~ferber)**

**Sept 2019**

**Resp du module: J. Ferber**

# Prolégomènes

## ◆ La programmation utilise des techniques et des concepts

- On s'éloigne de plus en plus des machines:
  - ☞ De la programmation en langage machine jusqu'aux langages de scripts
- On utilise des principes issus d'autres disciplines
  - ☞ Electronique: composants
  - ☞ Logique / mathématiques: fonctions (Lisp, Scheme, etc., C, Pascal),
  - ☞ Philo: programmation par objets
    - Un objet est défini par ses attributs et les opérations qu'on peut faire sur lui.
    - On crée une classification des types d'objets (héritage)

## ◆ La programmation par agents utilise une métaphore sociale

# Des sociétés animales ...

## ◆ Construire des programmes sous la forme d'entités autonomes en interactions

- Métaphore des sociétés animales et humaines

### FOURMIS



Abeilles

Termites



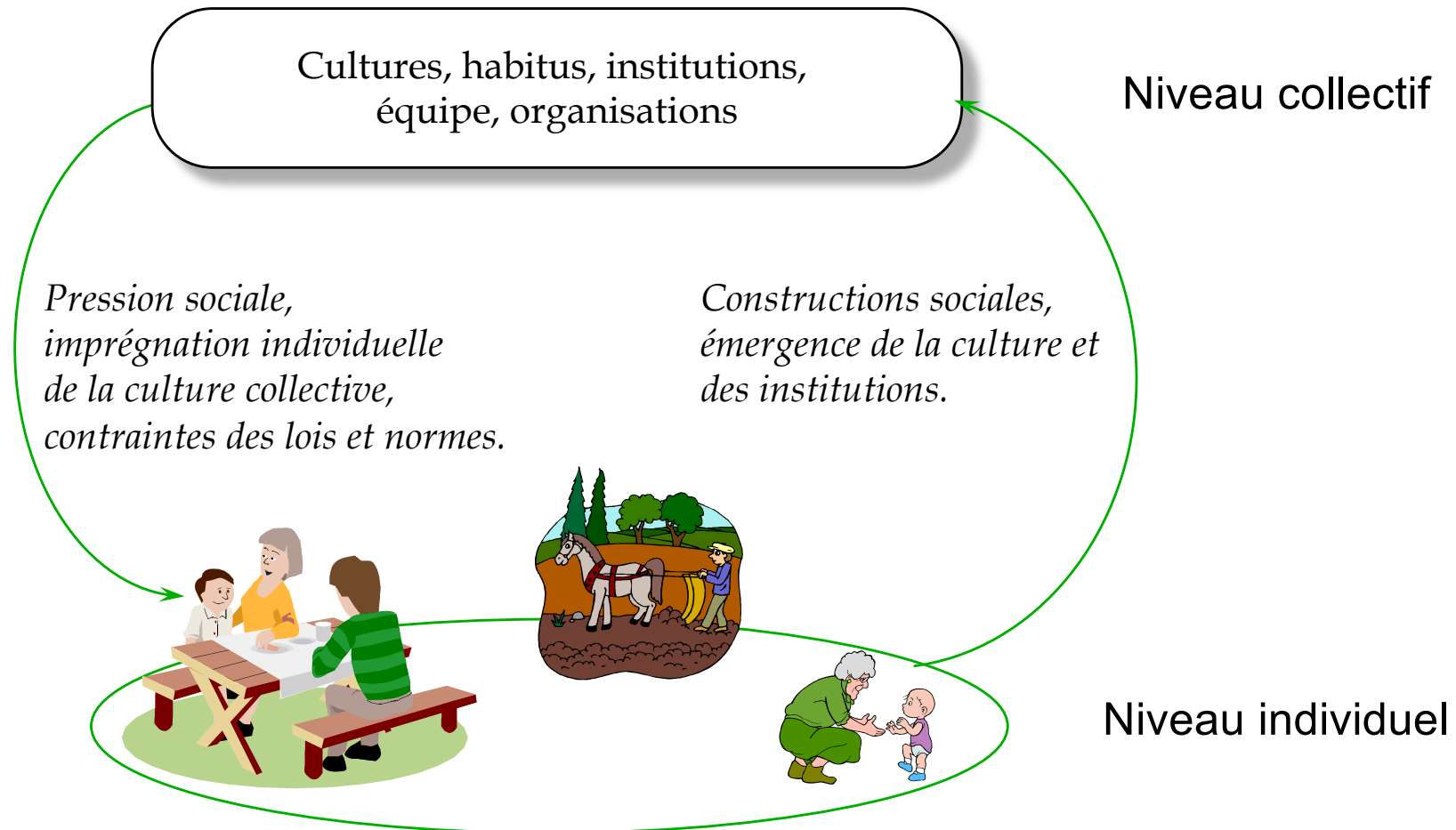
# aux sociétés humaines



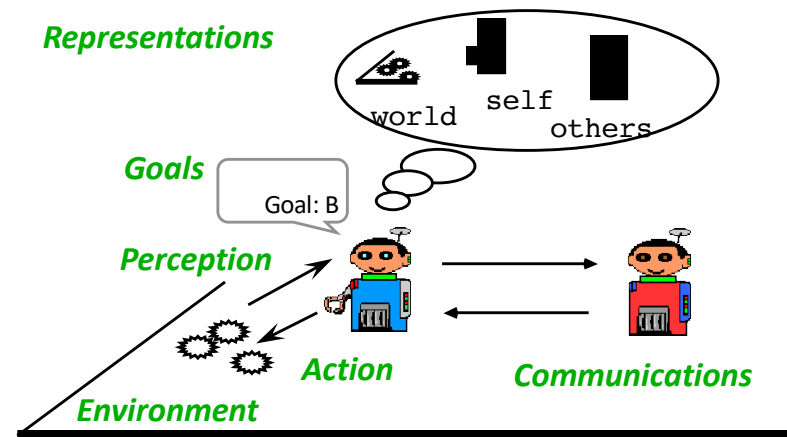
## ◆ Point commun

- Existence d'individus
- Ces individus interagissent (communication entre autre) dans un environnement
- Ils coordonnent leurs actions et/ou entrent en conflit/compétition
- Il existe une organisation sociale fondée sur la notion de rôle (métier, statut, caste, poste, etc..) et de groupes (familles, nid, clans, tribus, sociétés, associations, bandes, etc..)
- Ils produisent collectivement des structures (objets, sociétés, activités), qu'il n'auraient pas pu créer individuellement

# Les deux niveaux d'une société



# Les systèmes multi-agents



## ◆ Un SMA est défini comme:

- Un ensemble **C d'entités** plongées dans un **environnement E** (E est caractérisé par l'ensemble des états de l'environnement S)
- Un ensemble **A d'agents** avec A inclus dans C
- Un système **d'action** (opérations) permettant à des agents d'agir dans E (une opération est une fonction de  $S \rightarrow S$ )
- Un système de **communication** entre Agents (envoi de messages, diffusion de signaux,...)
- Une **organisation O** structurant l'ensemble des agents et définissant les fonctions remplies par les agents (notion de **rôle** et éventuellement de **groupes**)
- **Eventuellement:** une relation à des utilisateurs **U** qui agissent dans ce SMA via des agents interfaces **U** inclus dans A

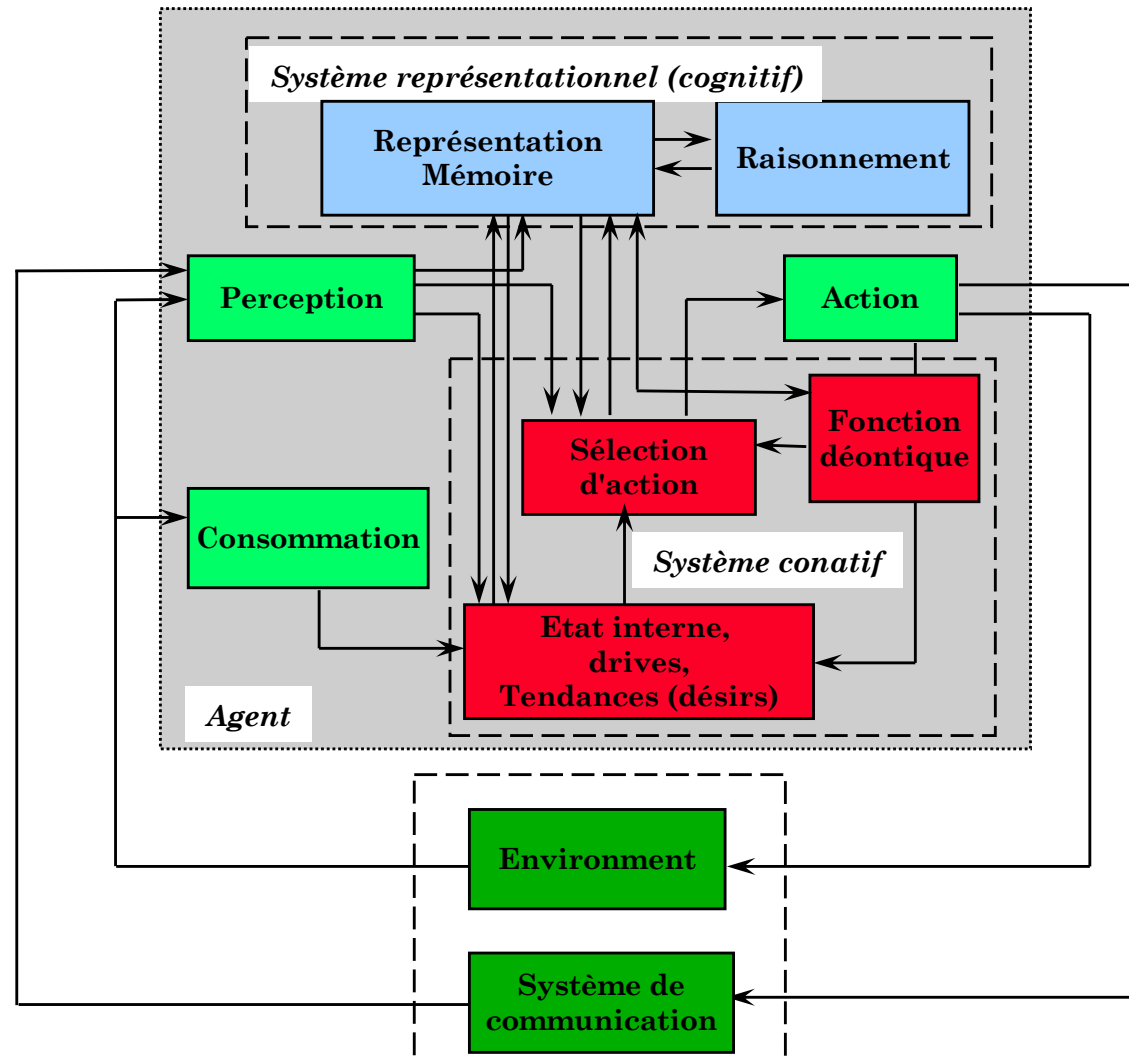
# Un agent

Un **agent** est une entité physique (hard) ou logicielle (soft) située dans un environnement (réel ou virtuel) qui est capable de:

- **agir** dans son environnement
- **percevoir** and partiellement **se représenter**\* son environnement (et les autres),
- **communiquer** avec d'autres agents,
- mû par se **tendances internes** (buts, recherche de satisfaction, "drives"),
- **se conserver**\* et **se reproduire**\*,
- **Jouer un (des) rôle** dans une organisation,

et qui présente un **comportement autonome** qui est la conséquence de ses perceptions, de ses représentations et de ses communications.

# Architecture générique d'agent



perception -> délibération -> action



# Concepts fondamentaux

## ◆ Pas de centralisation

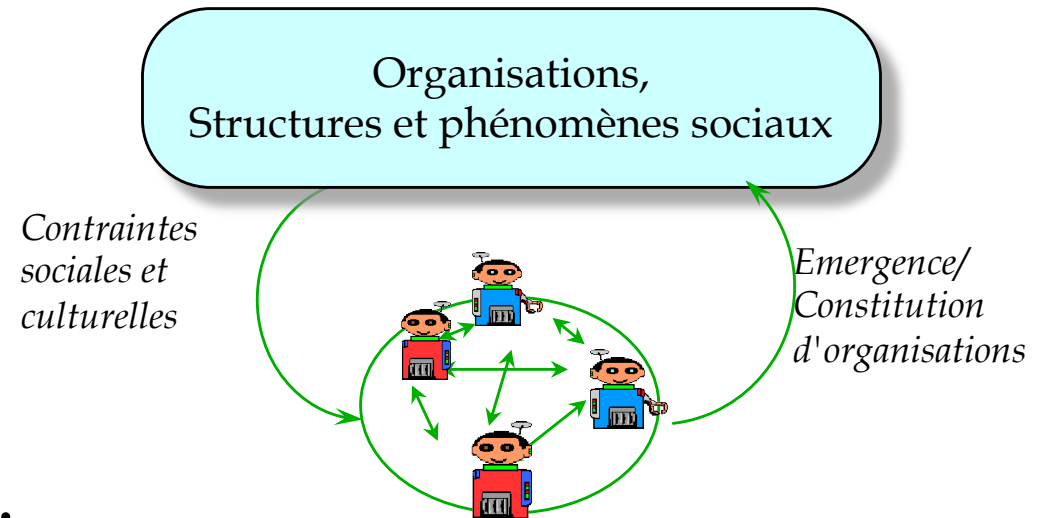
- Autonomie
- Localité
- Contrôle distribué

## ◆ Structures de coordination

- Coopération / compétition

## ◆ Emergence

- Existence d'un niveau micro (le niveau agent) et d'un niveau macro (la société dans son ensemble)



# Usage

◆ Utilisée pour la création d'effets spéciaux de films (effets de foule)

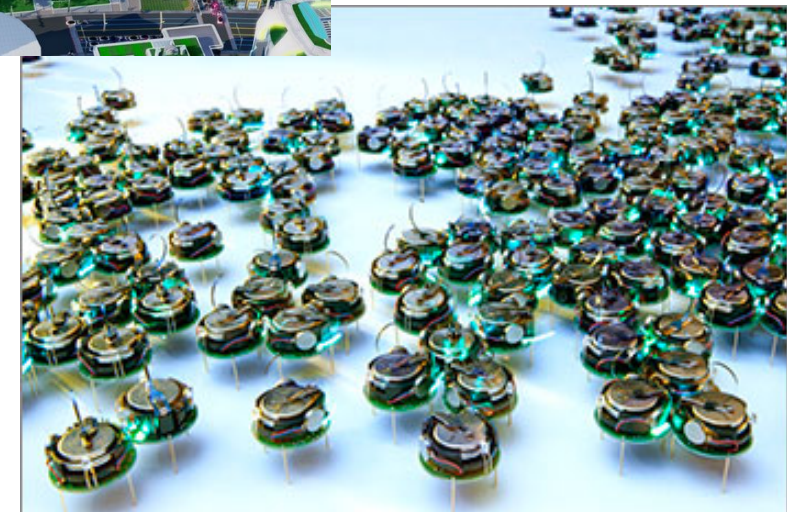
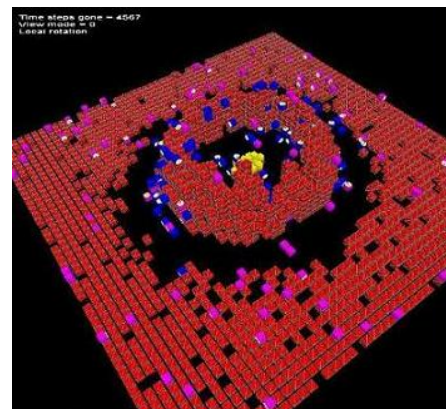


◆ La robotique collective



◆ Les jeux video (intelligence des caractères)

◆ La simulation de systèmes complexe



# Notion de niveaux de cognition

## ◆ Agents "réactifs":

- Qui ne disposent pas d'une représentation explicite de leur environnement
  - ☞ Actions situées
  - ☞ Ex: fourmis

## ◆ Agents "cognitifs":

- Qui ont une représentation de leur environnement, d'eux-mêmes et des autres agents et qui peuvent raisonner sur leurs représentations
  - ☞ Planification
  - ☞ Ex: êtres humains

# Exemple d'agent réactif

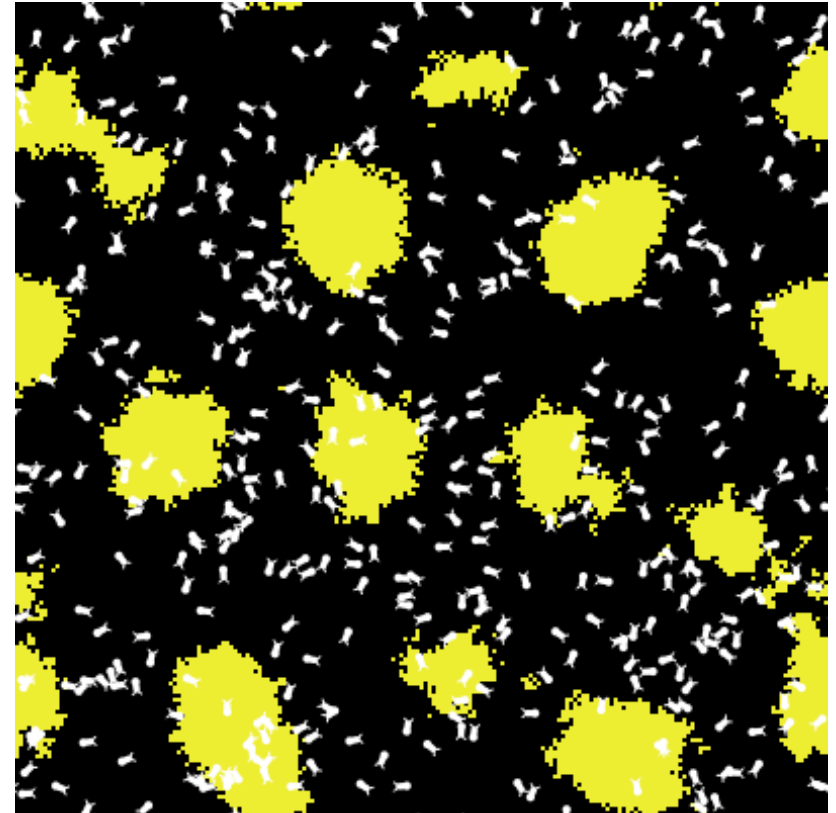
## ◆ Un garde dans un jeu

- Tant que je ne vois rien, je suis mon chemin de garde
- Si je vois un ennemi
  - ☞ S'il n'est pas menaçant et si je ne suis pas blessé, je l'attaque
  - ☞ S'il est menaçant ou si je suis blessé, je sonne l'alarme, et je m'éloigne



# Emergence: les termites

- ◆ **Construction de tas de bois**
  - Comme les termites lors de la construction de leur nid
- ◆ **Termites assemblent des morceaux de bois et les empilent. Les termites suivent un ensemble de règles simples individuelles et locales**
  - Règles
    - ☞ Si je rencontre un morceau de bois, je prend le morceau et continue mon chemin.
    - ☞ Quand je porte un morceau de bois et que je rencontre un autre morceau de bois par terre, je cherche un coin vide et je dépose mon morceau de bois.
- ◆ **Avec ces règles, finalement, les regroupements de bois, se transforment en piles..**



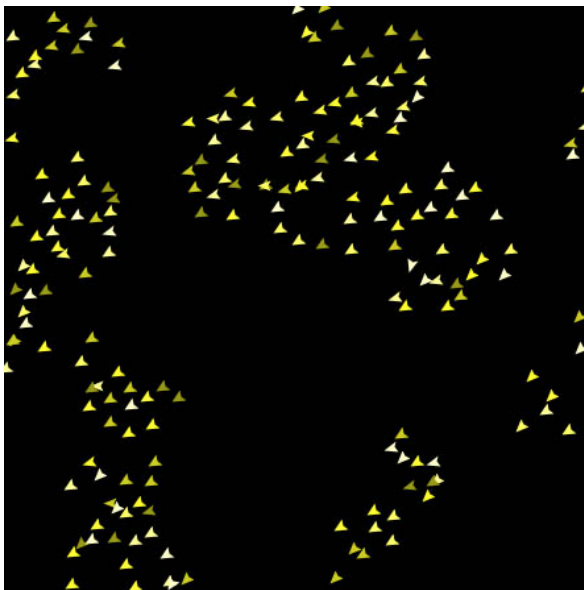
# NetLogo

- ◆ **Environnement de développement multi-agents réactifs, pour l'étude de systèmes complexes**
  - On peut gérer des centaines (voire des milliers) d'agents qui opère en même temps dans un environnement
  - Ecrit en Java
  
- ◆ **Très facile à utiliser**
  - Interface conviviale..
  - Tourne sur toutes les machines (Windows, Mac OS, Linux)

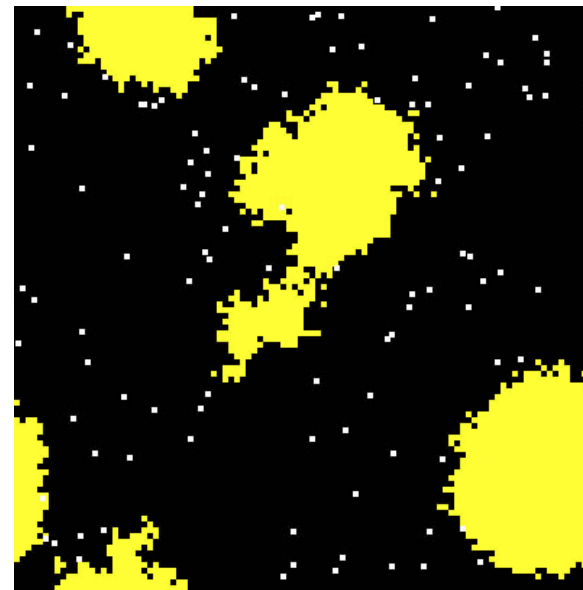
# Structure de NetLogo

## ◆ NetLogo est un monde 2D constitué de

- Patches: constitue des "zones", des portions de l'environnement
- Tortues : créatures qui peuvent se déplacer et agir dans cet environnement
  - ☞ Attention: NetLogo appelle "agent" à la fois les patches et les tortues, mais cela ne correspond pas à la vision multi-agents



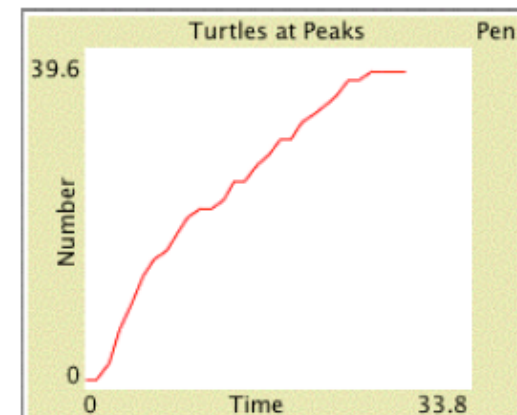
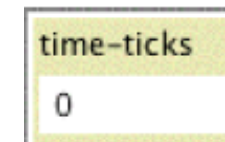
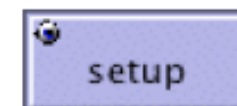
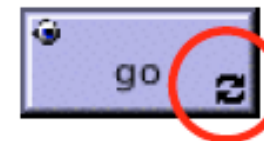
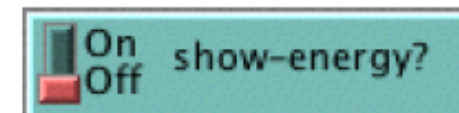
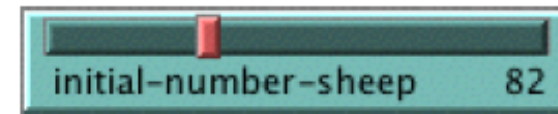
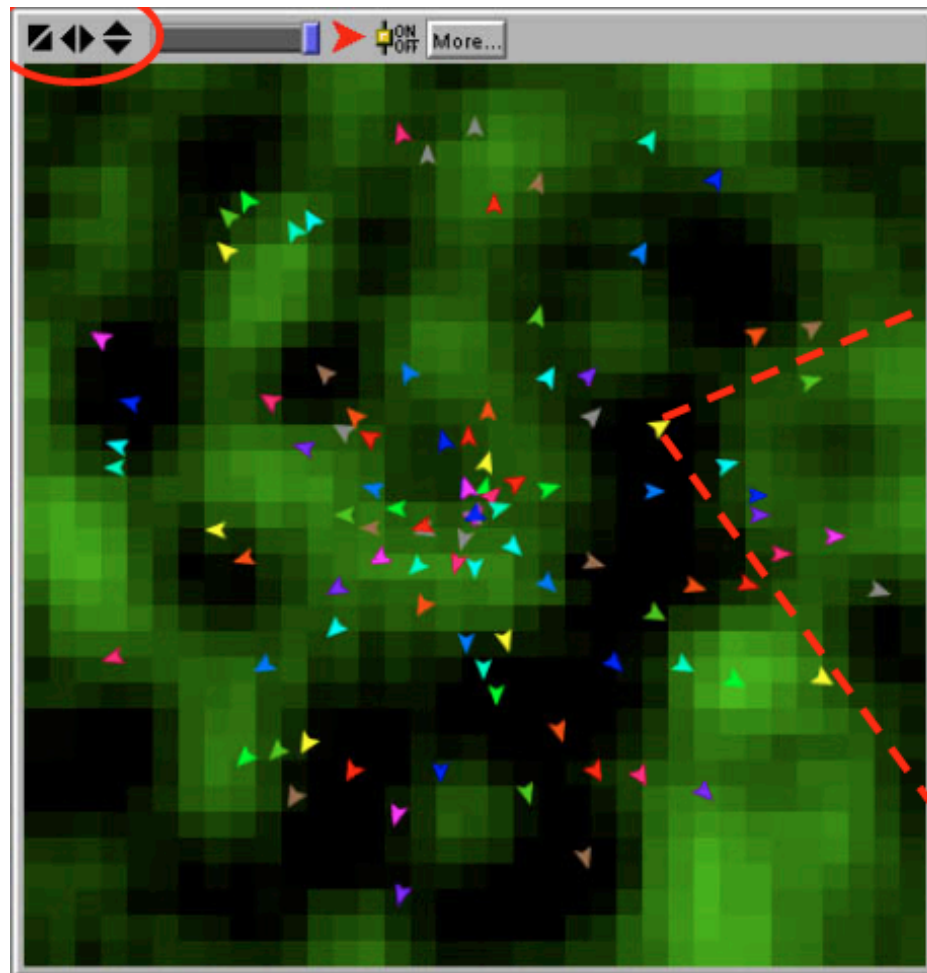
Vol en formation



Vol en formation

# Interface graphique

- ◆ Très facile de rajouter des contrôle et des moniteurs et de les relier à des paramètres du modèle





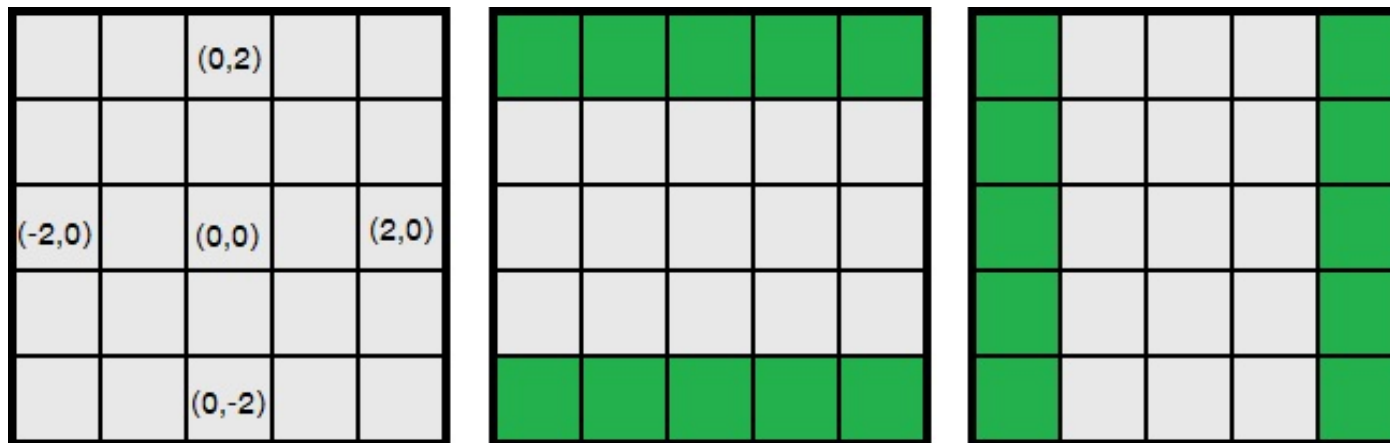
# Deux types d'entités

## ◆ Les tortues

- Elles bougent et elles sont utilisés pour implémenter les agents d'un SMA

## ◆ Les patches

- Se sont des parcelles de terrain. Elles ne bougent pas mais peuvent avoir un comportement...



# Les procédures

## ◆ Syntaxe logo

```
to draw-carre[taille]
  pen-down
  repeat 4 [fd taille rt 90]
end
```

## ◆ Fonctions (retournent une valeur):

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

# Variables

## ◆ Global

- Définition de variables globales

```
globals [ max-energy ]
```

## ◆ Définition de variables locales: **let <var> <val>**

```
Let r 56 ;; la variable r contient maintenant 56
```

## ◆ Paramètres: [**<var>**]

```
to draw-carre [taille]
```

## ◆ Set : affectation de variables

```
set energy 35
```

```
set color-of turtle 5 red
```

# Les attributs des tortues

- ◆ **Fonctionnent comme des attributs dans les langages objets**

- ◆ **Doivent être déclarés**

```
turtles-own [energy speed]
```

- ◆ **Peuvent être lus directement depuis l'intérieur**

```
set energy energy - 1
```

- ◆ **Lus depuis l'extérieur de l'agent**

```
[energy] of latortue
```

# Les patches ont des attributs

◆ **Les patches ne bougent pas mais peuvent évoluer**

◆ **Déclaration**

```
patches-own [taille quantite]
```

◆ **Attributs**

- pcolor : la couleur du patch

◆ **Particularité:**

- Une tortue a un accès direct à l'attribut du patch sur lequel elle se situe:

```
to-do-something  
  set color red ;; la tortue devient rouge  
  set pcolor blue ;; le patch devient bleu
```

# Quelques primitives #2

## ◆ Ask

- Demande à un ensemble d'entités (tortues et patches) de faire quelque chose

```
ask turtles [  
    set color white  
    setxy random-xcor random-ycor ]
```

```
ask patch 2 3 [ set pcolor green ]
```

## ◆ Create-turtle

- Crée un ensemble n de tortues

```
create-turtles n [  
    set color white  
    set size 1.5 ;; easier to see  
    set energy random (2 * max-energy)  
    setxy random-xcor random-ycor  
]
```

# NetLogo: structures de contrôle

## ◆ if

- Deux formes: if et ifelse

- ☞ if <condition> [<instructions>]

- ☞ ifelse <condition>

- [<instructions-then>][<instructions-else>]

## ◆ repeat

- Pour répéter une instruction

- ☞ repeat <nombre> [<instructions>]

# NetLogo: géométrie tortue

## ◆ On peut dessiner des figures à partir du comportement des tortues

- Pour avancer: `fd <n>`

- Pour se diriger vers la droite (gauche):

  - ☞ `rt <n>` (ou `lt <n>`) ;; tourne d'un angle de n (en degrés) vers la droite (ou la gauche)

## ◆ Figures standard:

```
To carre [n]
  repeat 4 [fd n rt 90]
end
```

```
To poly [c n]
  repeat c [fd n rt 360 / c ]
end
```

```
To cercle-carres [n]
  repeat 9 [carre n rt 30]
end
```

```
To spirale [n k a]
  repeat k [
    fd n rt 360 / a
    set n n + 1 ]
end
```



# Pour aller dans une direction

```
set heading towards <entité>
```

```
face <entité>
```

## ◆ exemples

```
set heading towards patch 0 0
```

```
;; va vers le patch qui se situe au centre du terrain
```

```
face patch 0 0
```

```
;; va vers le patch qui se situe au centre du terrain
```

```
let v one-of other turtles in-radius 3
```

```
If (v != nobody)
```

```
  [face v]
```

```
  ;; faire autre chose...
```

# AgentSets

## ◆ Un sous ensemble d'entités (patches ou tortues)

```
turtles with [color = red ]
```

```
patches with [pxcor > 0]
```

```
turtles in-radius 3
```

```
turtles in-radius 3 with [color = red ]
```

**aux éléments duquel on peut demander quelque chose:**

```
ask turtles with [color = red] [bouge 30]
```

# Quelques tests sur les agentsets

## ◆ Tester si un agentset est vide

### ● Any?

```
any? turtles in-radius 3
```

## ◆ Tester si un élément n'existe pas (= nobody)

```
let v one-of turtles in-radius 3
if v != nobody
  [;; si c'est différent de nobody ]
```

## ◆ Compter le nombre d'élément dans un agentset

```
count turtles with [color = blue]
;; retourne le nombre de tortures bleues
```

```
Count other turtles in-radius 5
;; compte le nombre de tortues autour de la tortue
;; courante et différente de cette dernière
```

# Espèces (breed)

- ◆ On peut créer des espèces (breed) ou "races" de créatures

```
breed[ wolves wolf ]
```

- ◆ Crée automatiquement les procédures associées

```
create-<breed>  
<breed>-own  
<breed>-here  
<breed>-at  
...
```

# Les listes

- ◆ **Comprend un ensemble de primitives permettant de manipuler les listes à la mode « lisp » ou « scheme ».**
- ◆ **first, but-first, last, item**
- ◆ **fput, lput,**
- ◆ **length, empty?, member,**
- ◆ **remove, remove-item, replace-item**
- ◆ **list, sentence, sublist**
- ◆ **sort**

# Exemple: le tri par les termites

```
to setup
  clear-all
  set-default-shape turtles "bug"
  ;; randomly distribute wood chips
  ask patches
    [ if random-float 100 < density
      [ set pcolor yellow ] ]

  ;; randomly distribute termites
  create-turtles number [
    set color white
    setxy random-xcor random-ycor
    set size 5 ;; easier to see
  ]
```

# Les termites

```
to go
  search-for-chip
  find-new-pile
  put-down-chip
end
```

```
to search-for-chip
  ifelse pcolor = yellow
  [ set pcolor black
    set color orange
    fd 20 ]
  [ wiggle
    search-for-chip ]
end
```

```
to find-new-pile
  if pcolor != yellow
  [ wiggle
    find-new-pile ]
end
```

```
to put-down-chip
  if pcolor = black
  [ st pcolor yellow
    set color white
    get-away ]
  [ rt random 360
    fd 1
    put-down-chip ]
end
```