La programmation orienté agent utilise une métaphore social, on construit des programmes sous la forme d'entités autonomes en interactions.

Les systèmes multi-agenst (SMA) :

- un ensemble c d'entités
- plongées dans un environnement E
- un ensemble A d'entités inclus dans C
- un système d'action S permettant à des agents d'agir dans E
- un système de communication entre agents (envoie de messages, diffusion de signaux)
- une organisation O structurent l'ensemble des agents (notion de rôle/groupes)

Les agents physique ou logicielle :

- agissent dans un environnement
- percevent et se representent partiellement leur environnement et les autres entités
- communiquent entre eux
- sont motivé par des tendances internes (buts, recherche de satisfaction)
- se conservent et se reproduisent
- jouent un role dans une organisation

Agents réactifs : ne disposent pas d'une représentation explicite de leurs environnement (fourmis)

Agents cognitifs : ont une représentation de leurs environnement d'eux mêmes et des autres agents et peuvent raisonner sur leurs représentations (humains)

ce qui est interressant avec la programmation agents c'est l'émergences de situation imprévus

NetLogo

Un monde en 2D constitué de "patches" (portions de l'environnement) et "tortues" créatures qui peuple ce monde

Syntaxe

Procedures

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

Fonctions

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

Définitions variables globales

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

Définitions attributs (tortues/patches)

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

Définitions variables locales

```
let r one-of turtles in-radius 5
```

Affectation de variables

```
set energy 35
```

Donner des ordres

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

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

Créer des 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 ]
```

Déplacement

```
fd "distance", rt "angle", lt "angle"
```

S'orienté

```
set heading towards "patch"
```

```
face "patch"
```

Sélectionner un sous-ensemble de d'entités

```
turtles with [ color = red ] patches with [ pxcor > 0 ]
```

Tester si agents vide

any? turtles in-radius 3

if v != nobody [ ]

Compter le nombre d'agent dans un set

count turtles with [color = red]

Création de espèces

breed [wolves wolf]

Créé automatiqueme les procédures associées

create- -own -here -at

Les listes

first, but-furst, last, item

fput, lput

length, empty?, member

remove, remove-item, replace-item

list, sentence, sublist

sort

L'importance de l'environnement

objet avec l'attribut le plus petit

min-one-of reines in-radius 5 [distance myself]

objet avec l'attribut le plus grand

max-one-of patches in-radius 8 [hauteur-herbe]

diffuser un attribut au patches

diffuse

uphill (avance la tortue dans le patch dont la valeur de l'attribut est la plus grande)

Architecture réactivees

pour programmer un agents il est plus simple d'utiliser des architectures speciales.

On part d'une notion d'états interne

FSM : finite state machine

Architecture de subsumption Architecture de neuronale

Machines a états finis

Etats de l'automate : une activié de l'agent

Evenement : quelque chose qui se passe dans le monde extérieur (sert de déclancheur à lactivité)

Action : quelque chose que l'agent fait et aura pour conséquence de modifier la situation du monde et de produire d'autres évenements

implementation en utilisant un variable ctask

turtles-own [ctask]

to go ask turtles [run ctask] end

Exemple voir implementation FSM termites

Implementation à l'aide de tables

Un interprete va sélectionner l'état courant dans la table et déclencher la chose à faire ensuite Si pas de condition vérifié on demeure dans le meme état.

Etat courant | Action | Condition | Etat suivant en-fuite | fuir-enemis | sauf | patrouille patrouille | pattrouiller | menace ET ennemis-plus-fort | en-fuite -- | | menace Et ennemis-moins-fort | en-attaque en-attaque | se-battre | ennemis-vaincus | patrouille -- | | ennemis-plus-fort | en-fuite

FSM hierarchique

Necessite d'implementer une pile

declaration de mode en fonction de condition ex si energie basse mode recuperation d'energie sinon mode en activité

turtles-own [stack ctask]

;; penser à initialiser l'agent dans le setup

set stack [] set cstack "proc-initiale"

;; en fait la proc initiale du mode to go-mode [mode] if ctask != mode [set stack fput ctask stack ;; push l'état ancien set ctask mode] end

to return-mode ifelse empty? stack [;; s'arreter] [set cstak first stack set stack bf stack] end

to go ask turtles [run ctask] end

FSM hiérarchique à reflexe

turtles-own [stack ctask]

;; penser à initialiser l'agent dans le setup

```
set stack [] set cstack "proc-initiale"


;; en fait la proc initiale du mode to go-mode [mode] if ctask != mode [ set
stack fput ctask stack ;; push l'état ancien set ctask mode ] end


to return-mode ifelse empty? stack [;; s'arreter] [ set cstak first stack set
stack bf
```