HMIN108 - Programmation orientée agents
Partie 1: Agents réactifs
3 - Architectures réactives

Suro François
(adaptation des cours de Jacques Ferber)

Université de Montpellier
Laboratoire d'informatique, de robotique
et de microélectronique de Montpellier

Septembre 2020

Question: comment programmer un agent?

1. Programmer en faisant au mieux
   - Ad hoc: Pas d'abstraction, sur mesure
   - Devient vite confus

2. Utiliser des méthodes et patterns
   - Approche développement logiciel
   - Identifier des caractéristiques
   - Maintenance et évolution

Architectures pour agents réactifs

Architectures d'agent qui ne prennent pas en compte la notion de représentation

- (Ou de manière très simplifiée)

Deux approches de base

- Partir d'une notion d'état interne: Finite State Machine
- Partir d'une relation aux perceptions: Subsomption, Architecture neuronale

Autre possibilités

- Adaptation, Tâches en compétition
- Prise en compte de la satisfaction de l'agent
- ...

Architectures basées sur les perceptions

Rodney Brooks: critique de l'IA symbolique

- Pas de représentations symboliques, pas de raisonnement abstrait
- l'intelligence est une propriété *émergente*

Considérer que pratiquement toute l'information se trouve dans l'environnement

- Actions situées
  - Dirigée par les événements (percepts)
  - Donner une grande importance à l'environnement
- Communication par dépôt de marques (indices et chemins)
- Buts et obstacles sont dans l'environnement

Actions situées

Comportement lié aux percepts

- Pas de mémorisation de l'environnement
  - Pur: perception immédiate (pas de mémoire)
  - Impur (avec apprentissage): mémorisation uniquement des états internes passés..
- Règles d'action
  - Si *état interne* et/ou *état perçu* alors *action*

R1
Si j'ai soif
et je vois du café sur la table
et je suis loin de la table
alors je m'approche de la table

R2
Si j'ai soif
et je vois du café sur la table
et je suis près de la table
alors je prend le café

Codage des actions situées

Perceptions

- Perceptions externes (vision, ouïe, radars, etc..)
  - Eventuellement issues d'une phase de reconnaissance.
    Ex: si je perçois un ennemi...
- Capteurs sur des données corporelles du robot..
  - Ex: si j'ai faim, si mes points de vie sont inférieurs à une valeur, etc..

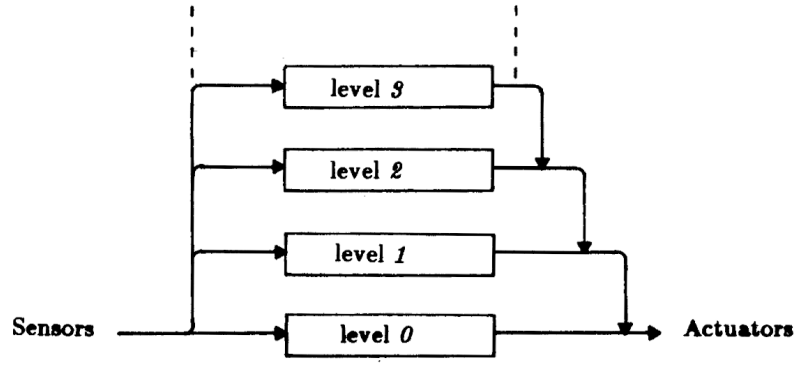
Actions (ou tâches)

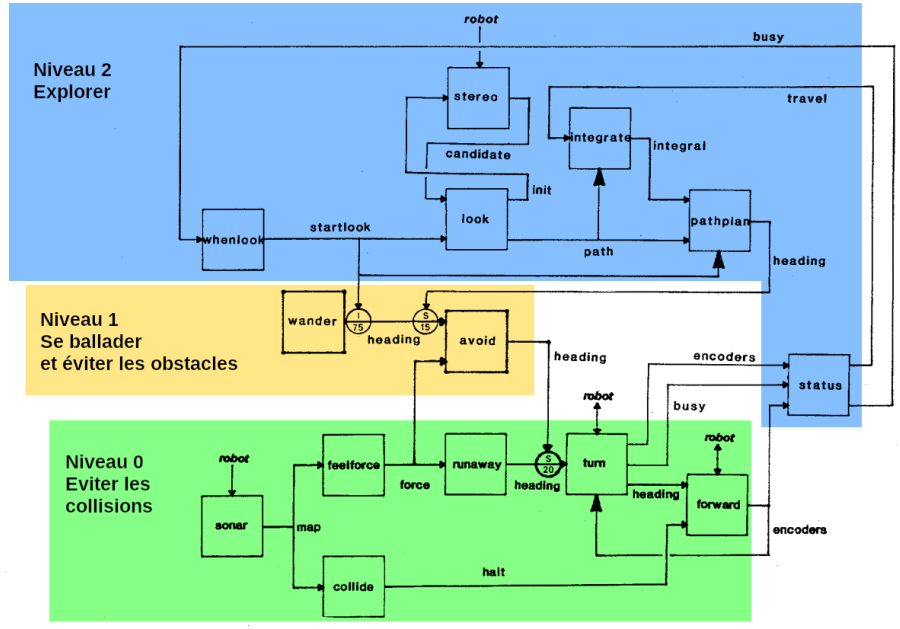
- Actions élémentaires
  - Avancer, reculer, fuir, aller vers la perception, etc..
- Actions composites
  - Rapporter nourriture

Subsomption

Application à un robot

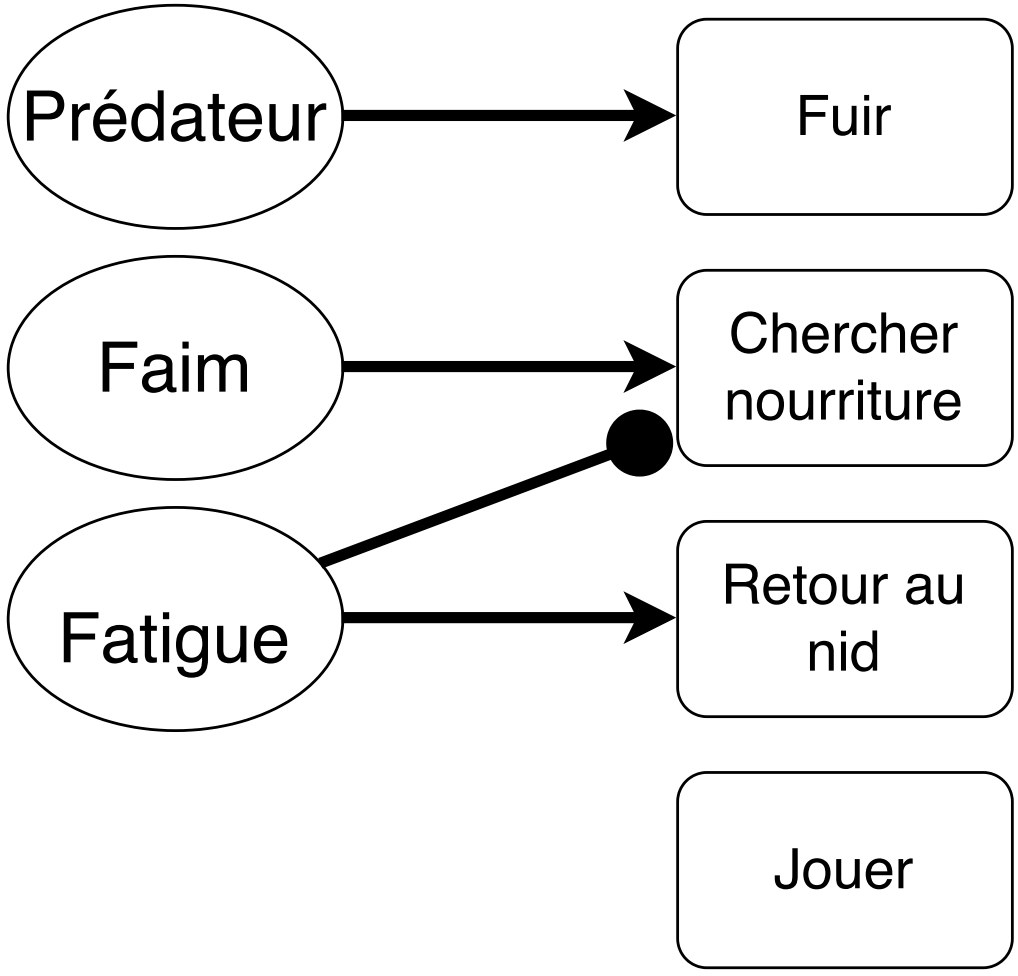
- Chaque tâche accède aux perceptions (capteurs)
- Chaque tâche calcule une action (moteurs)
- Si une tâche de bas niveau n'est pas activée, elle cède le contrôle à une tâche de plus haut niveau





A Robust Layered Control System For A Mobile Robot [Brooks, 1986]

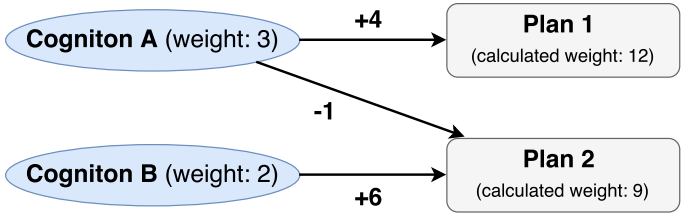
Subsomption



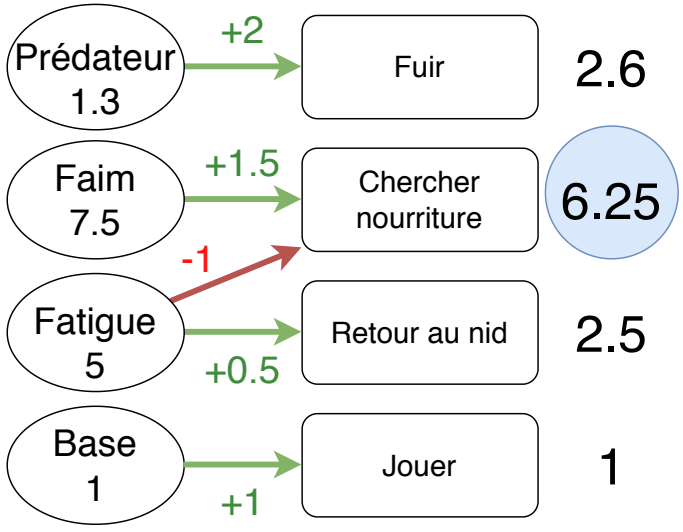
Cognitons

Système de décision pour la modélisation de sociétés

- Chaque plan est complètement autonome
- Les cognitons représentent des perception, des états internes et autres facteurs entrant en compte dans la décision.
- Chaque cogniton a un poids qui représente son importance.
- Pour décider du plan a choisir, le poids de chaque cogniton est transmis à travers un lien d'influence.



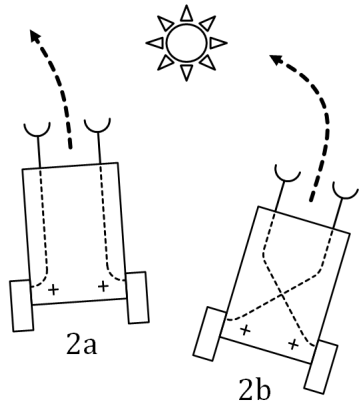
MetaCiv [Ferber, 2014]



Signal, connexionnisme

Signal

- Les capteurs fournissent un signal continu
  ex: quantité de lumière perçue
- Les divers signaux sont combinés, modifiés
  ex: seuil, fonctions de transfert, inversion ...
- Un signal est transmis aux actionneurs
  ex: voltage, position angulaire ...

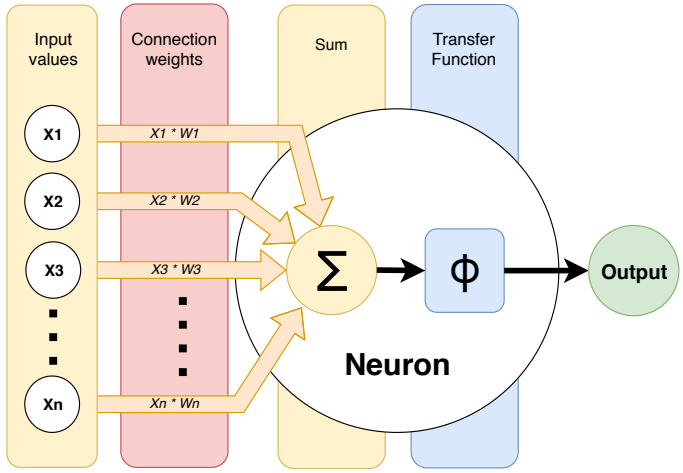


Vehicles: Experiments in Synthetic Psychology [Braitenberg, 1984]

Réseaux neuronaux

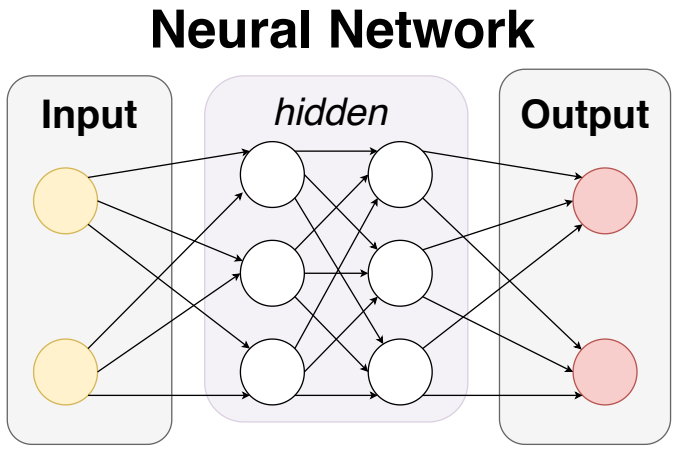
Perceptron

- Les valeurs d'entrée sont multipliés par le poids de leurs connexions (des "synapses").
- Le résultat est sommé dans le neurone, une fonction de transfert est appliqué (sigmoid, Relu ...).



Réseau de neurones: perceptron multi couches

- Plusieurs couches connectés en succession.
- Couche: ensemble de neurones non connectés entre eux
- Signal propagé de l'entrée vers la sortie

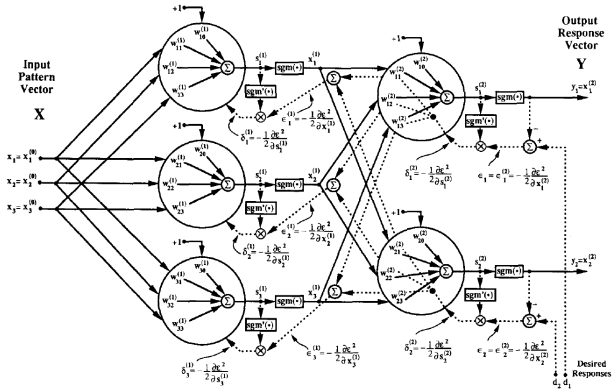


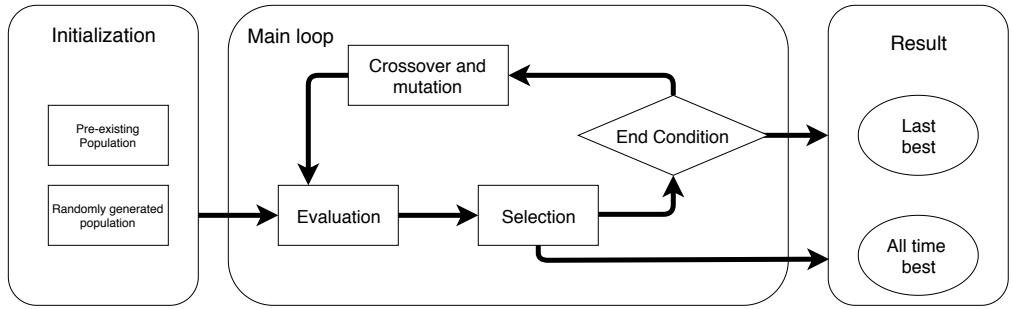
Réseaux neuronaux: apprentissage

Le comportement est décrit par les poids des connexions

Impossible à la main, besoin d'apprentissage:

- Backpropagation: corriger progressivement l'erreur sur un ensemble d'exemples.
- Algorithmes génétique: tester plusieurs configurations, sélectionner les meilleures pour en générer de nouvelles.

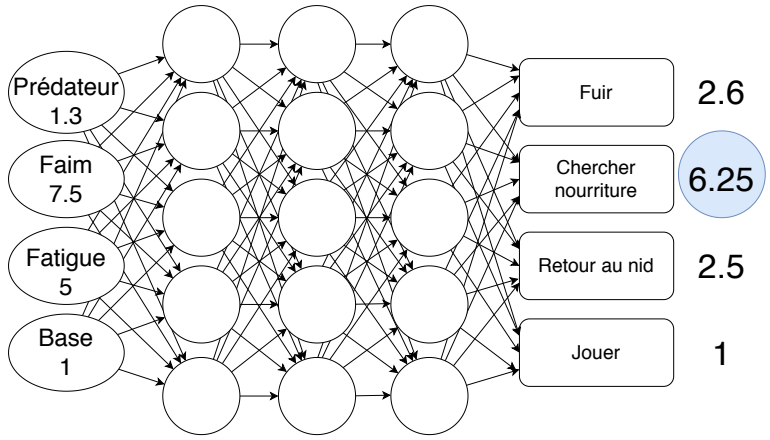




Réseaux neuronaux: applications

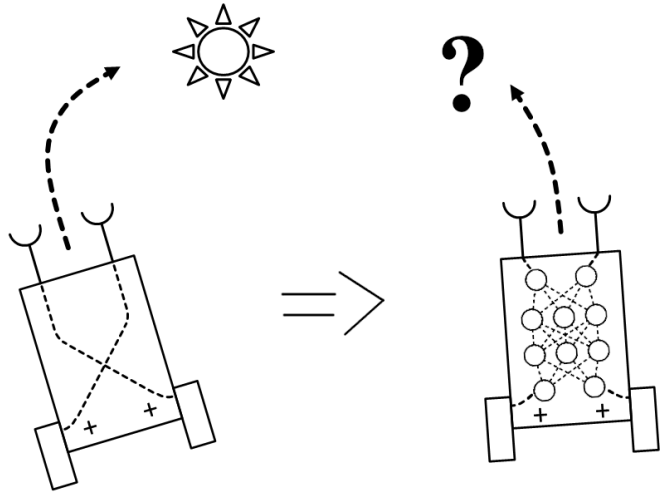
Classification

- Les percepts sont fournis en entrée du réseau qui calcules un "score" pour chaque action possible (comme la "probabilité" d'un classificateur).
- L'action avec le score le plus haut est choisie.



Signal, intensité

- Le signal des capteurs est modifié par le réseau dont la sortie est fournie comme signal aux actionneurs
- Replace le câblage d'un véhicule de Braitenberg



Architecture basée sur un état

Actions situés

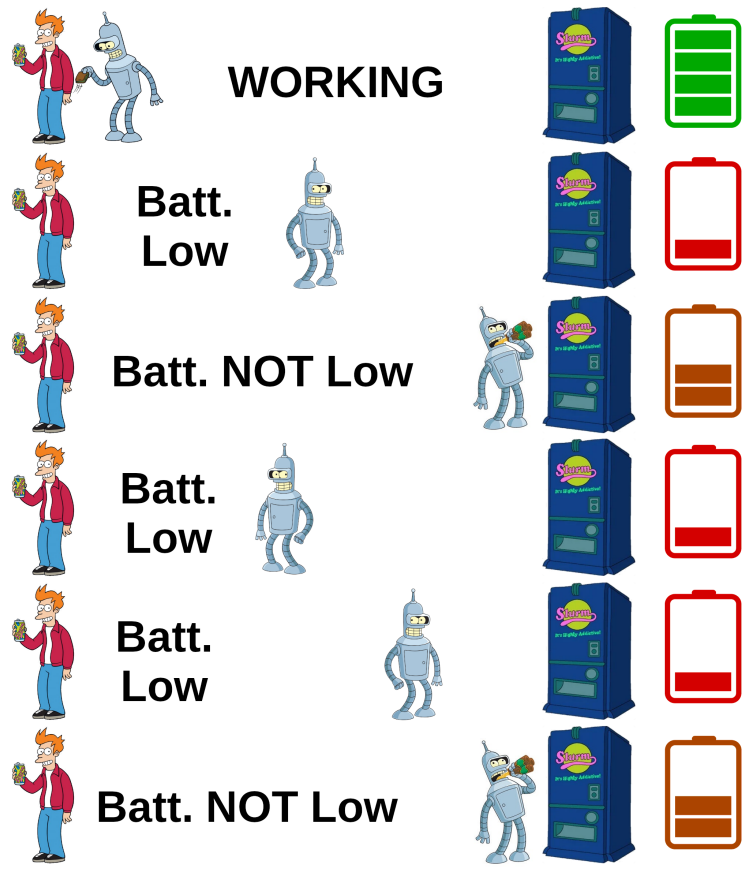
- Percept actif=action active
- Percept inactif=action inactive

Problème

- Oscillation entre 2 comportements (sans accomplir aucun des deux).
- Besoin de "s'engager dans une tâche"

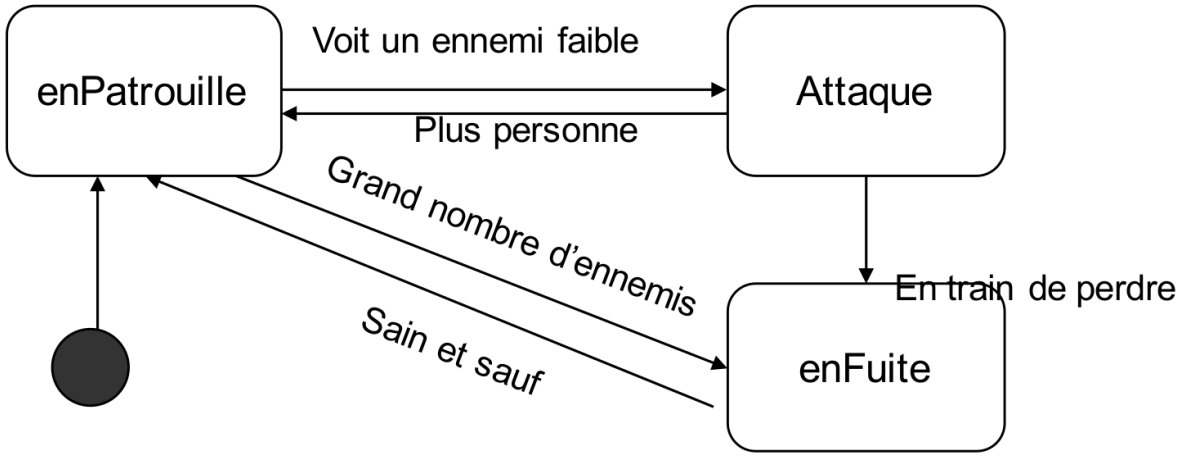
Solution basée sur un **état** qui persiste indépendamment des perceptions:

Machines a états finis



Machines à états finis (FSM, automates)

- **État de l'automate**: une activité de l'agent.
- **Événement**: quelque chose qui se passe dans le monde extérieur (ou intérieur à l'agent) qui peut être perçu. Sert de déclencheur (trigger) d'une activité.
- **Action**: quelque chose que l'agent fait et aura pour conséquence de modifier la situation du monde et de produire d'autres événements.
  - L'action est liée directement à l'activité



Implémentation FSM : un simple switch

```
1  void run(TypeEtat etat){
2  switch(etat){
3    case etat_enFuite:
4      echapperEnnemis();
5      if (sauf())
6      changerEtat(etat_Patrouille);
7      break;
8    case etat_Patrouille :
9      patrouiller();
10     if (menace()) {
11     if (plusFortQueEnnemis())
12       changerEtat(etat_attaque);
13     else
14       changerEtat(etat_enFuite);}
15       break;
16   case etat_attaque :
17     seBattre();
18     if (ennemisVaincus())
19       changerEtat(etat_enPatrouille);
20     else if (plusFaibleQueEnnemis())
21       changerEtat(etat_enFuite);
22     break;
23 }//end switch
```

Implémentation FSM : table

- Construit une table correspondant au système des états
  - Un interprète va sélectionner l'état courant dans la table et déclencher la chose à faire ensuite
  - Si pas de condition vérifiée on demeure dans le même état

Etat courant	**Action**	**Condition**	**Etat suivant**
enFuite	fuirEnnemis	enSécurité	Patrouille
Patrouille	Patrouiller	Menace ET ennemisPlusfort	enFuite
–		Menace ET ennemisMoinsfort	enAttaque
enAttaque	seBattre	ennemisVaincus	Patrouille
–		ennemisPlusFort	enFuite

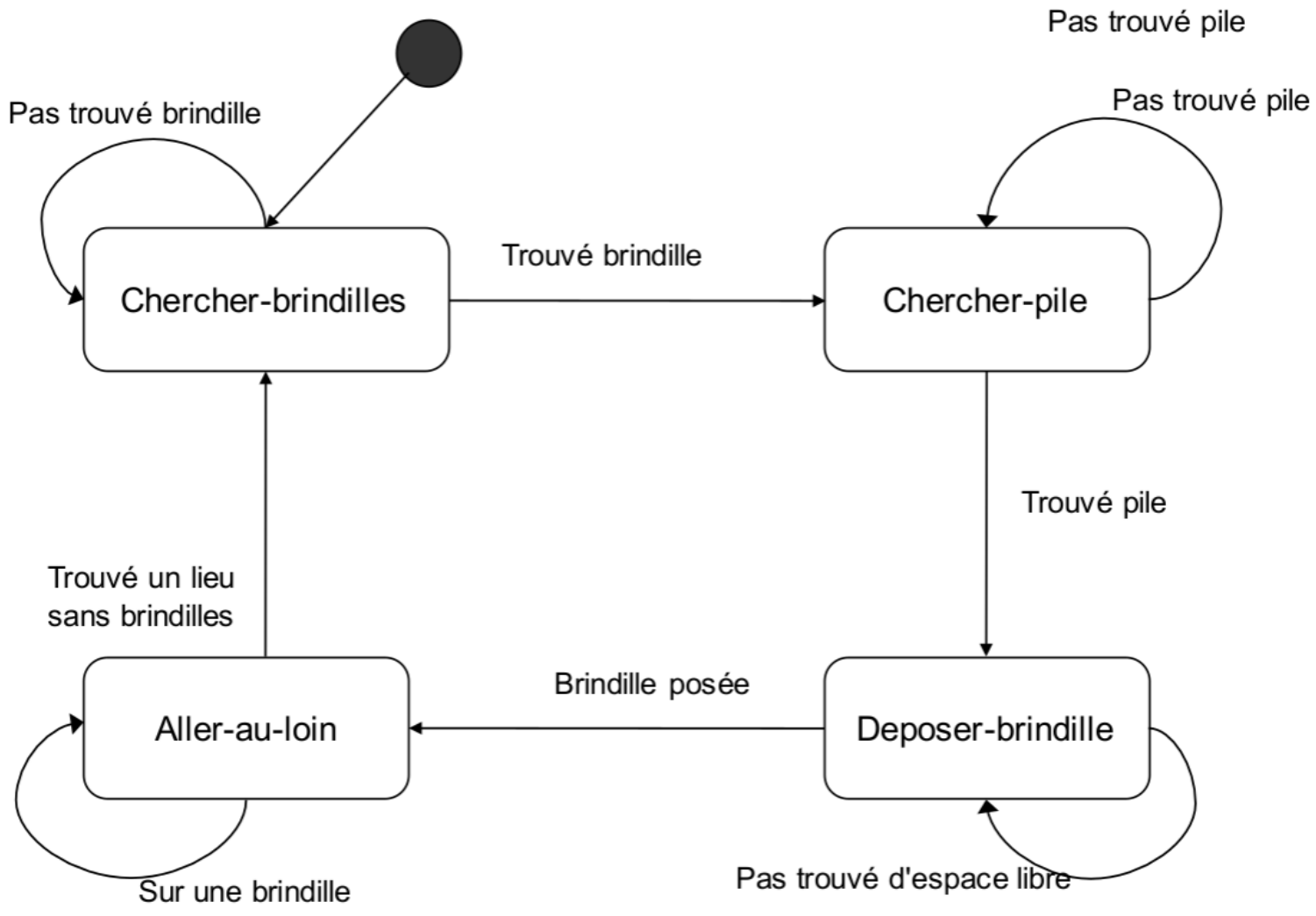
Implémentation FSM : objets

```
1  class Etat{
2    abstract void exec(Agent ag);
3  }
4
5  class EnFuite extends Etat{
6    void exec(Agent ag){
7      ag.echapperEnnemis();
8      if (ag.enSecurite())
9        ag.changerEtat(new
             Patrouiller());
10   }
11 }
12
13 class Agent {
14   Etat ctask;
15     void do(){
16     ctask.exec(this);
17   }
18   void changerEtat(Etat etat){
19     ctask = etat;
20   }
21   boolean sauf(){... }
22   void echapperEnnemis() { ... }
23 }
```

Classe Etat

- Classe abstraite Etat
- Chaque état de la FSM est implémenté sous la forme d'une classe hérité
- Création, recherche dans un dictionnaire(singleton), statique ...

Les termites en FSM: schéma



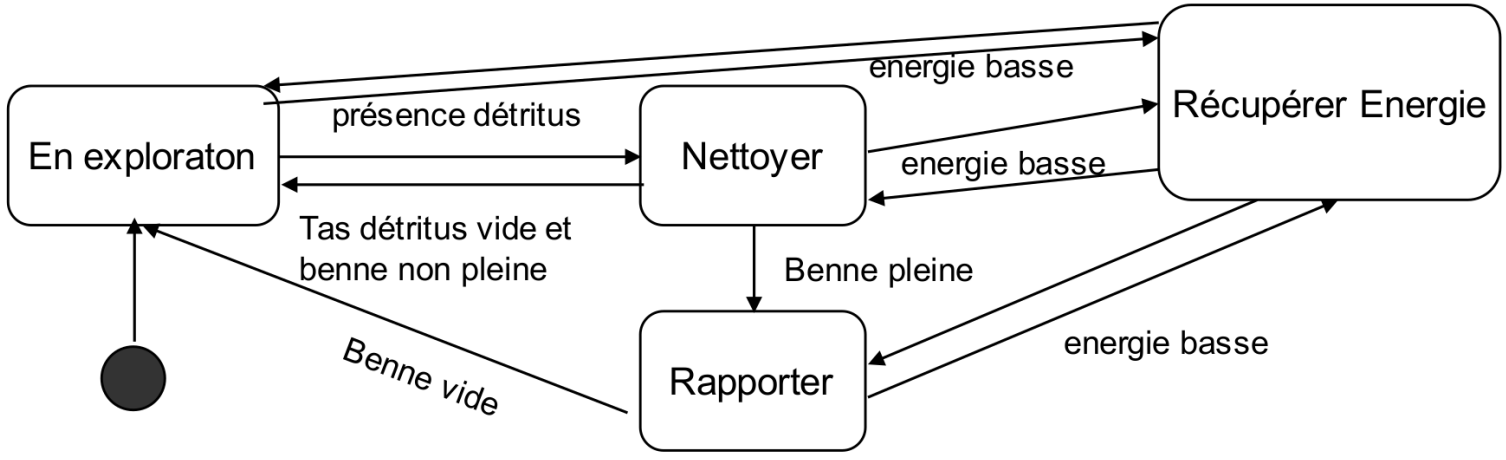
Les termites en FSM: implémentation (attribut *ctask*)

```
turles own [ctask]

to go
   ask turtles
      [run ctask]
end

to chercher-brindilles
   ifelse pcolor = yellow
   [ set pcolor black
     set color orange
     fd 20
     set ctask "chercher-pile"]
   [wiggle fd 1]
end

to get-away
   ifelse pcolor = black
   [ set task "chercher-brindilles"
       ]
   [wiggle fd 1]
end
```

```
to chercher-pile
   ifelse pcolor = yellow
   [set ctask "deposer-brindille"]
   [wiggle fd 1]
end

to deposer-brindille
   ifelse pcolor = black
   [ set pcolor yellow
     set color white
     fd 20
     set ctask "aller-au-loin" ]
   [wiggle fd 1 ]
end
```

Machines à états finis hiérarchique

Si l'on a besoin de caractériser des "modes" comportementaux différents, exprimés de manière hiérarchique

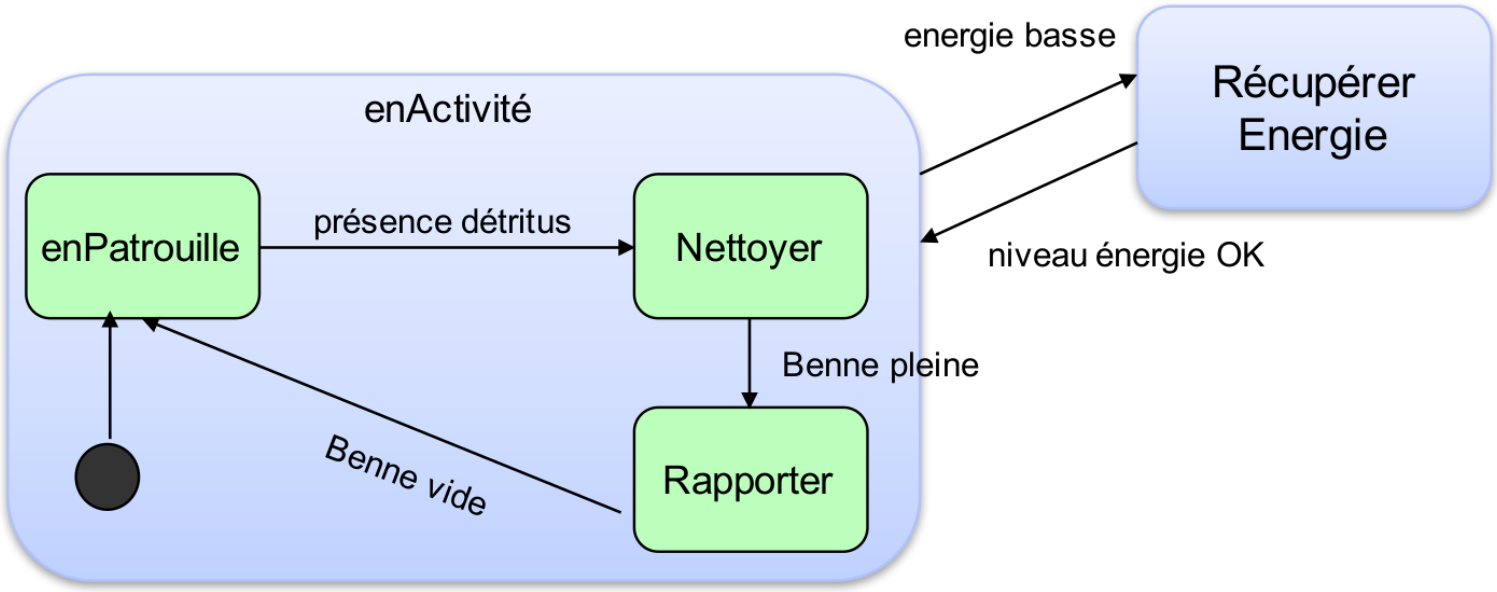


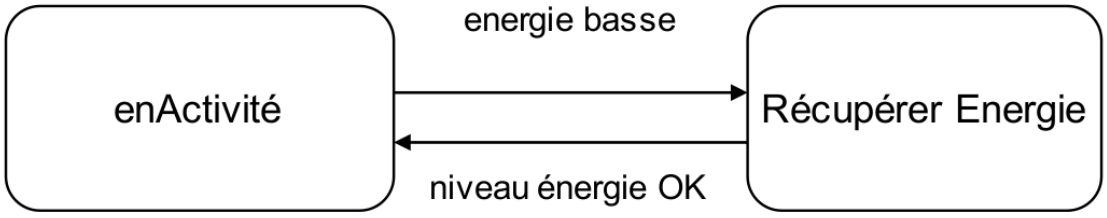
Ce robot doit récupérer de l'énergie quoi qu'il fasse par ailleurs

FSM hiérarchique

Construction d'activités correspondant à un automate

Nécessite d'implémenter une pile (fonctionnement des procédures)



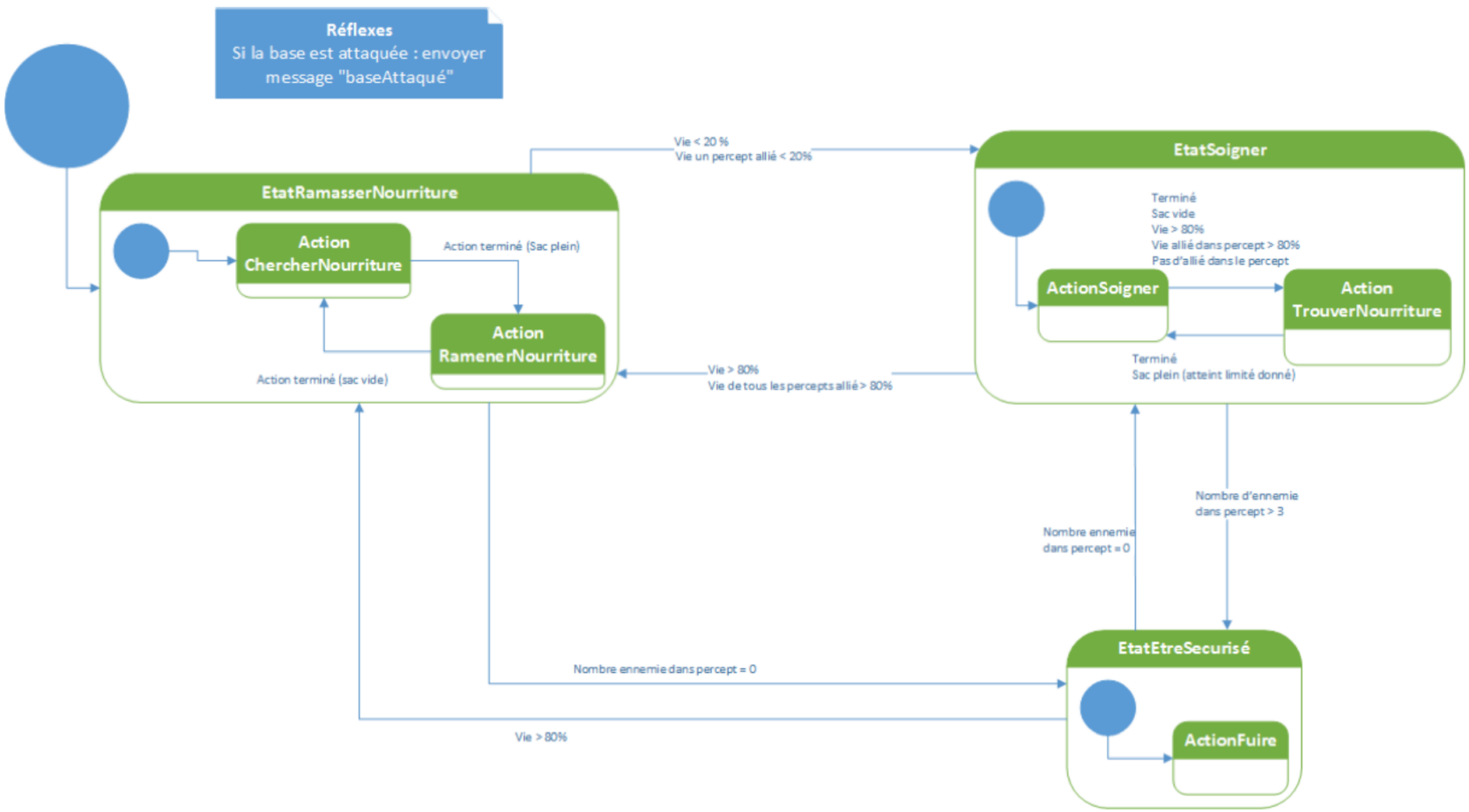


Mode 'enActivité'

Mode 'Récupérer énergie'

FSM hiérarchique

Les différents mode/niveaux de la hiérarchie peuvent être des FSM, organisés en FSM.



FSM hiérarchique

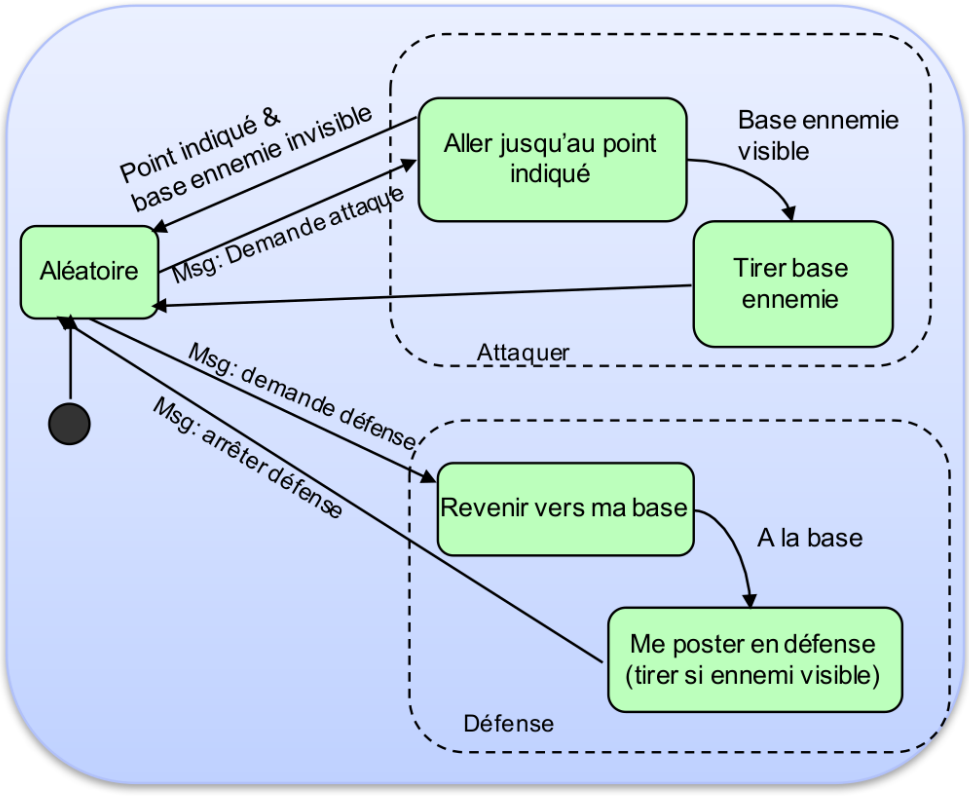
Nécessite d'implémenter une pile d'états

- Pour conserver l'état d'où l'on est parti.
- Fonctionnement standard des appels de procédures, on lit le dessus de la pile.
- Pour sortir du mode, on dépile.

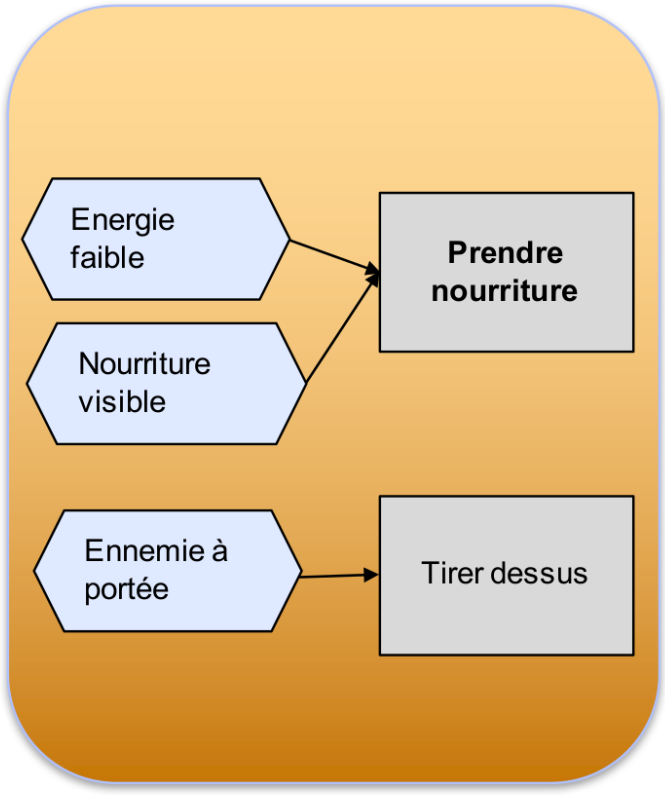
Nécessite d'implémenter des mécanismes d'exception

- Quel que soit l'état situé dans le FSM de "enActivité", il faut dépiler le FMS courant.

FSM + réflexes



FSM



Réflexes

Différence actions situées-FMS

Actions situées

l'important ce sont les perceptions.

- Avantages: permet de décrire des comportements fluides liés aux perceptions, de prendre en compte l'état immédiat.
- Inconvénient: difficulté à caractériser des suites d'action. Pas d'engagement: Les agents perdent ce qu'ils sont en train de faire.

Finite state machines

l'important c'est l'état (l'activité en cours). Les perceptions contribuent à modifier l'activité par des évènements (quelque chose a changé) qui déclenchent des transitions.

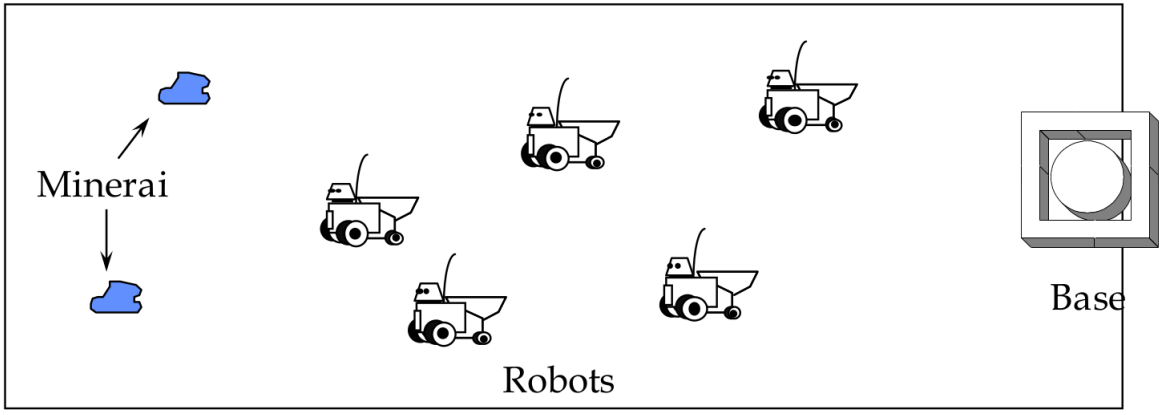
- Avantages: permet de caractériser un comportement procédural qui est déclenché en fonction d'évènements.
- Inconvénient: difficultés à prendre en compte ce qui n'est pas décrit pas l'automate (apprentissage). Trop d'engagement: Besoin d'un système de gestion de réflexes.

Cas d'étude: robots fourrageurs

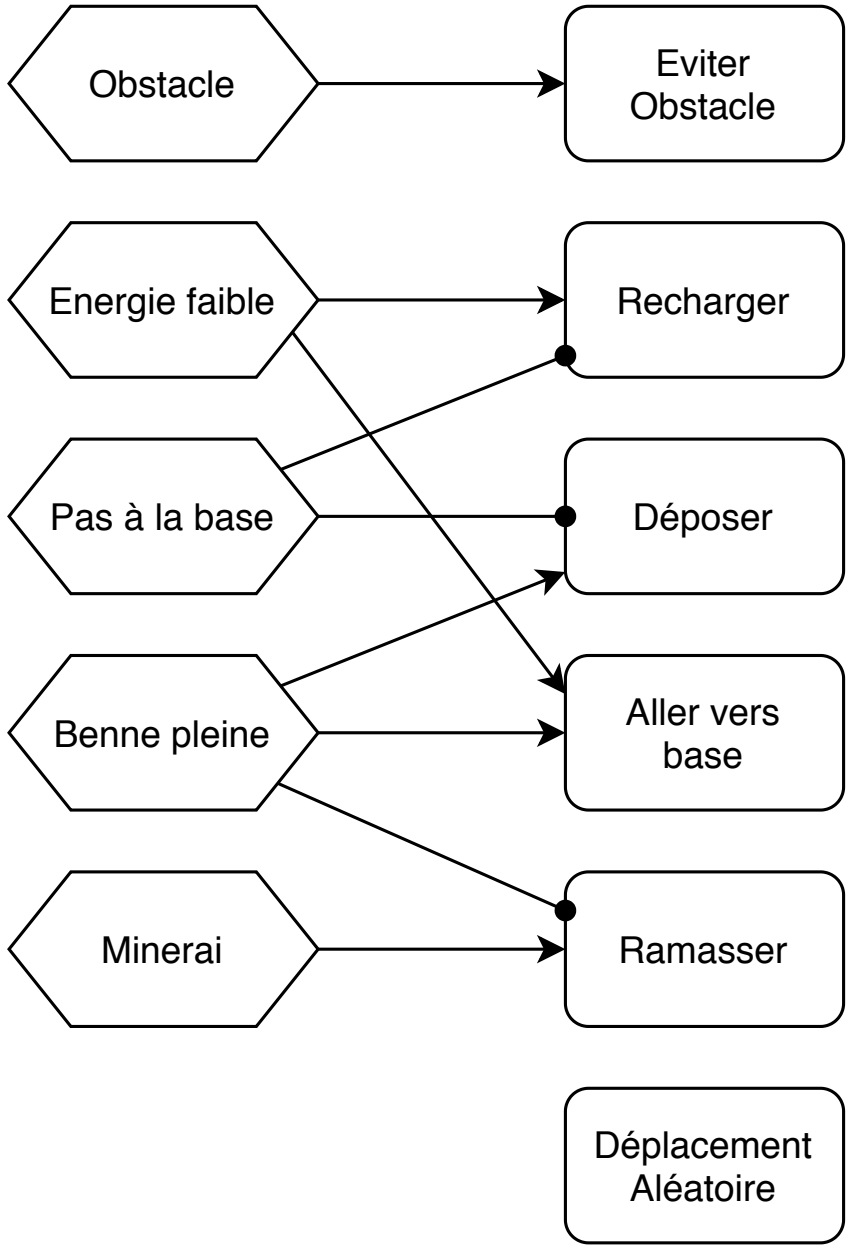
- Un ensemble de robots doivent collecter et ramener des échantillons de minerai à la bases et gérer leur énergie (similaire au fourragement des fourmis)
- Problème: comment décrire leur comportement afin qu'ils remplissent leur mission.

Actions disponibles:

- Ramasser/Déposer/Recharger
- Aller vers la base
- Déplacement aléatoire/éviter obstacles



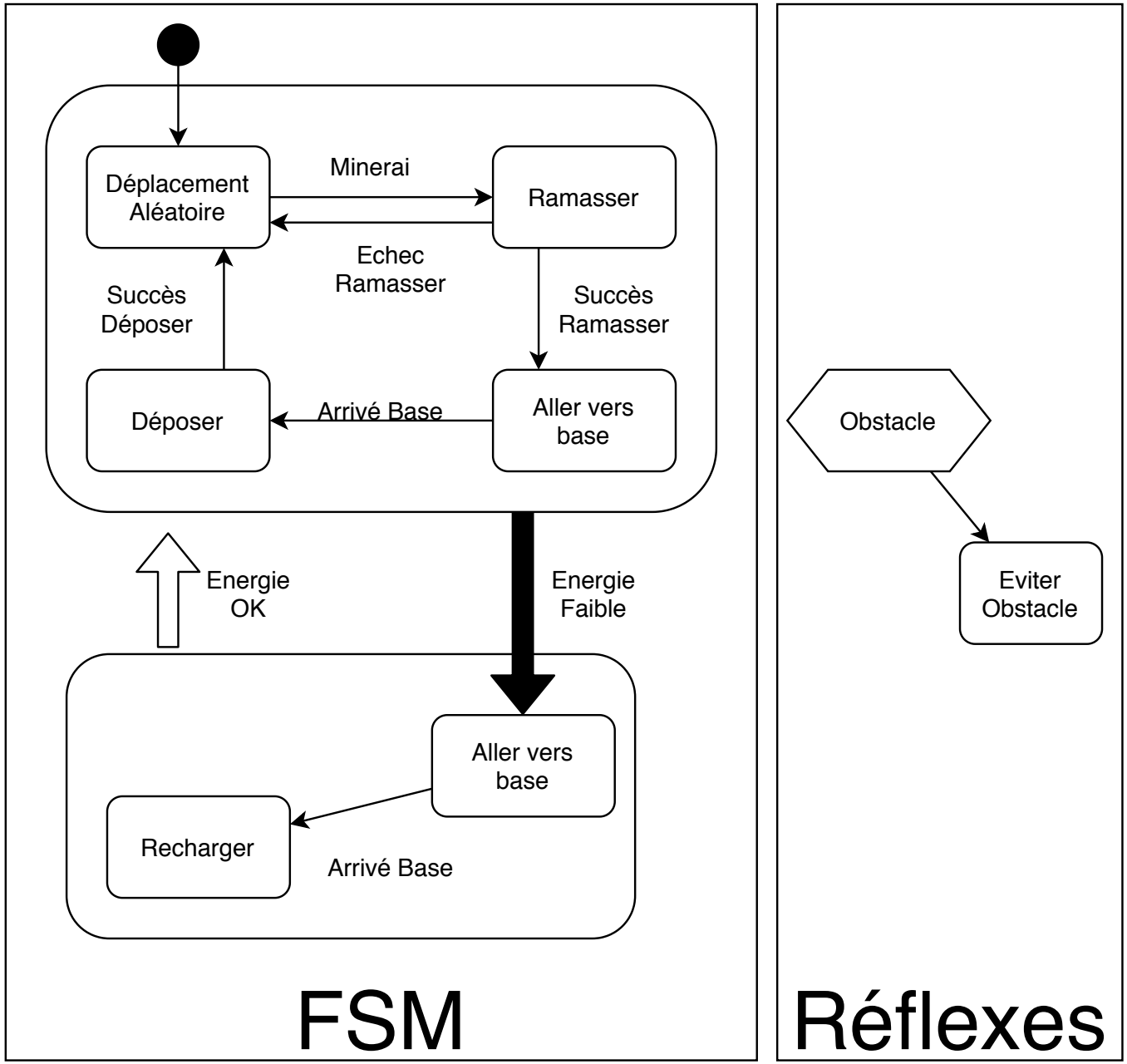
Fourragement: solution situé



Fourragement: solution situé

```
1  to go-robot
2     ifelse percept-obstacle
3     [eviter-obstacle]
4     [
5     ifelse energie < seuil-faible
6     [
7        ifelse [base?] of patch-here
8        [recharger]
9        [aller-vers-base]]
10    [
11    ifelse contenuBenne > 0
12    [
13       ifelse [base?] of patch-here
14       [deposer]
15       [aller-vers-base]]
16    [
17    ifelse [minerai?] of patch-here and contenuBenne = 0
18    [ramasser]
19    [aleatoire]
20    ]]]
21 end
```

Fourragement: solution FSM hiérarchique + réflexes



Fourragement: solution FSM hiérarchique + réflexes

Noyau de la FSM

```
1  to go-robot
2     ifelse percept-obstacle          ;;gestion des reflexes
3     [eviter-obstacle]
4     [                                ;;FSM
5        if energie < seuil-faible and savedTask = ""
6        [                             ;; passer en mode energie
7           set savedTask ctask        ;; sauver l'etat
8           set ctask "fsm-energie-aller-base"
9        ]
10       run ctask
11    ]
12 end
```

le changement de mode survient quand l'énergie franchit le seuil faible.
la restauration de l'état aura lieu à partir du mode énergie

Fourragement: solution FSM hiérarchique + réflexes

```
1  to fsm-norm-aleatoire
2     aleatoire
3     ifelse [minerai?] of patch-here
4     [set ctask "fsm-norm-ramasser"]
5  end
6
7  to fsm-norm-ramasser
8     if ramasser
9     [set ctask "fsm-norm-aller-base"]
10    [set ctask "fsm-norm-aleatoire"]
11 end
12
13 to fsm-norm-aller-base
14    aller-vers-base
15    if [base?] of patch-here
16    [set ctask "fsm-norm-deposer"]
17 end
18
19 to fsm-norm-deposer
20    deposer
21    set ctask "fsm-norm-aleatoire"
22 end
```

Etats des FSM

à gauche: mode normal

en bas: mode énergie

```
1  to fsm-energie-aller-base
2     aller-vers-base
3     if [base?] of patch-here
4     [set ctask
           "fsm-energie-recharger"]
5  end
6
7  to fsm-energie-recharger
8     recharger
9     if energie = energie-max
10    [  set ctask savedTask
11       set savedTask ""]
12 end
```