Warbot: la guerre des robots

Description et programmation

Architecture réactives #1

Jacques Ferber
ferber@lirmm.fr

LIRMM - Université de Montpellier II

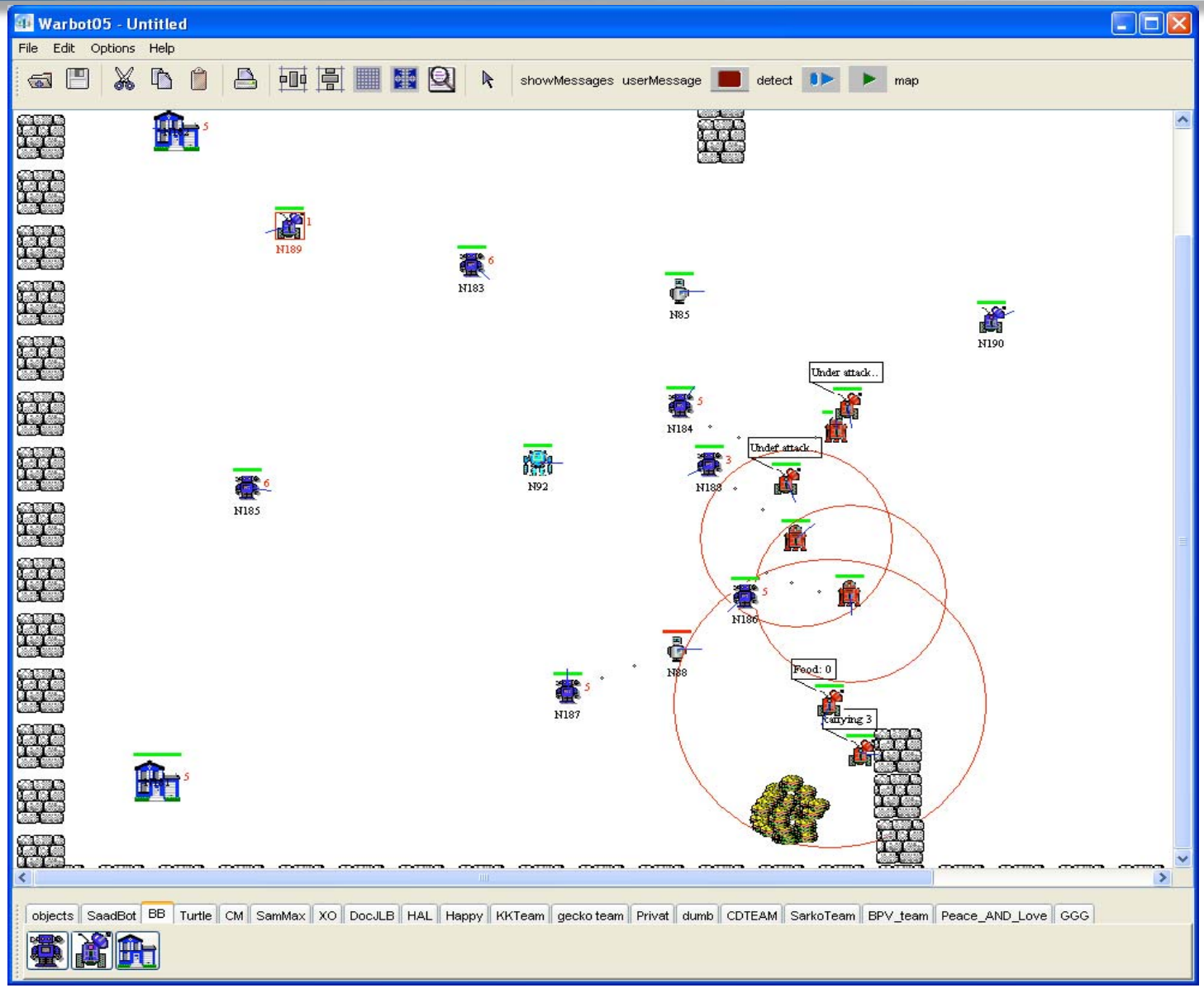
Plan

- Introduction
- Présentation de Warbot
- Architectures et programmation en Warbot
- Coordination d'actions collectives

Principes

- **Disposer d'une plate-forme pour tester des activités de coopération entre agents**
- **Les joueurs ne modifient pas le corps des agents mais simplement leur « esprit », leur mode de raisonnement et de décision**
  - Des « brains » dans la terminologie Warbot
- **Mode de fonctionnement:**
  - On définit des types de brain
  - On positionne des agents sur le terrain
  - On lance la simulation et que le meilleur gagne!
    - ☞ Pas d'interaction avec le joueur..

Une situation



Les types de corps

- **Missile launcher**
  - Lent: vitesse 5 tick
  - Résistant: energie 4000
  - Détection faible: 80
  - Peut tirer des missiles

- **Explorer**
  - Rapide: vitesse 2 ticks
  - Moins résistant: énergie 1000
  - Détection plus forte: 130

- **Base**
  - Immobile
  - Très résistant: énergie 12000
  - Détection grande: 200

Les autres entités

- **Obstacles**
  - Empêchent d'avancer
  - Peuvent être détruits par des missiles

- **Nourriture**
  - Peut être consommée et transformée en points de vie ou en nouveaux agents par la base
  - Peut être transportée

- **Missiles**
  - Envoyés par les lanceurs de missile
  - Ne sont pas téléguidés: continuent d'avancer dans la direction de leur lancement
  - S'auto-détruisent au bout d'un certain temps
  - Frappent en aveugle amis comme ennemis

La représentation spatiale

- **Pas de coordonnées globales:**
  - Les brains n'ont qu'une vision locale de leur environnement (pas de GPS, désolé ☺)
  - Fait partie du « game design »
- **Déplacements simples**
  - Les déplacements se font de manière vectorielle:
    - ☞ vitesse et direction
    - ☞ Pas de steering.. Pas d'accélération, de masse ni de physique du monde..
  - Les positionnements se font au pixel près
- **Formes ultra-simples**
  - Les objets (corps et autres objets) ont une taille (radius) et ils prennent la forme d'un cercle dans l'espace.
  - Collisions détectées très simplement par proximité

Les actions possibles #1

- **Déplacement**
  - **void setHeading(double dir)** : définit la direction à prendre (en degrés).
  - **void move()** : avance dans la direction courante. Ce n'est qu'une demande d'action: peut être bloqué pour avancer..
  - **boolean isMoving()**: indique si l'agent s'est déplacé lors de l'action précédente. Permet de savoir si un agent est bloqué par un obstacle.
  - **double towards(double a, double b)** : retourne la direction qu'il faut prendre pour aller dans la direction où se trouve le point de coordonnées a et b. Les valeurs de ces coordonnées sont relatives au robot.
  - **double distanceTo(Percept p)** : retourne la distance,
- **Manipulation d'objets**
  - **boolean eat(Food e)** : tente de manger l'entité e (si elle est comestible, sinon il ne se passe rien).
  - **boolean take(Percept e)** : tente de prendre une entité courante. Retourne le succès de l'action. Pour l'instant, on ne peut prendre que de la nourriture (percept Food qui représente un Hamburger). Quand un robot prend quelque chose, cela disparait de l'environnement.
  - **boolean drop(int i)** : dépose la ième entité disponible dans le sac du robot. Lorsqu'un robot dépose quelque chose, cela réapparait dans l'environnement à l'endroit où se trouve le robot.
- **Tirs (uniquement les lanceurs de missiles)**
  - **void launchRocket(double dir)**: lance un missile dans la direction dir

Les actions possibles #2

- **Communications**
  - Utilisent le modèle AGR (Agent/Groupe/Rôle) (voir plus loin)
  - Point à point ou broadcast (groupes et rôles)
    - **void send(AgentAdress agent, String act, String[] cont)**: envoie un message de performatif act et et de tableau d'argument cont à agent (un brain)
    - **void broadcast(String group, String role, String act, String[] cont):** envoie un message de performatif act et de tableau d'argument cont à tous les agents jouant le rôle role dans le groupe group.

Les percepts #1

- **Les percepts sont des représentations des choses perçues dans l'environnement**
  - Ce ne sont pas des pointeurs directement sur les chose!!
    - A chaque classe d'objets correspond un percept
    - Ex: classe warbot.kernel.Explorer → Explorer
  - Les percepts sont accessibles à tout moment. Ils sont rafraichis tous les ticks..
- **Méthode de base:**

  Percept[] getPercepts() :
    - retourne l'ensemble des percepts issus des entités qui se trouvent dans la portée de détection du robot sous la forme d'un tableau de percepts..

Les percepts #2

- **Les percepts sont des « objets »**
  - Methodes
    - String **getPerceptType()** : retourne le type de l'objet perçu sous la forme d'une chaîne de caractère (voir la table des correspondances entre type de percept et classe java).
    - double **getDistance()** : retourne la distance qui sépare le robot percevant de l'objet perçu
    - int **getRadius()** : retourne le rayon, c'est à dire la taille, de l'objet perçu
    - int **getEnergy()** : retourne l'énergie de l'entité perçue.
    - String **getTeam()**: retourne l'équipe de l'entité perçue (robots)
    - double **getDir()** : retourne la direction vers laquelle avance l'entité perçue

Utilisation des percepts

- **La perception est au centre de l'activité des robots**

```
Percept[] percepts = getPercets();
    for (int i=0;i<percepts.size();i++){
        Percept p = percepts[i];
        if (p.getPerceptType().equals("Food"))
            // si c'est de la nourriture, faire quelque chose
        else if (p.getPerceptType().equals("Explorer") && !p.getTeam().equals("MyTeam"))
            // sinon si c'est un Explorer de l'autre équipe, faire autre chose
    }
```

Les formalismes

- **Formalisme (fichier .fml de type xml)**
  - Fichier de l'outil Sedit qui spécifie la relation entre des choses (classes java, les « modèles » en MVC), les vues (classes Java de rendu de la chose) et les actions que l'on peut faire sur ces choses (les contrôles)
  - Il suffit que la classe soit dans le classpath du système
    - ☞ Sous MadKit / Warbot: il suffit que le jar des brains soit dans le directory <madkit>/lib

Un exemple d'éléments de formalisme

```
<node-desc name="myTeamMissileLauncher" class="warbot.kernel.RocketLauncher"
category="myTeam">
    <icon url="images/warbot/cyanrocketlauncher.gif"/>
    <property name="radius">12</property>
      <property name="team">myTeam</property>
    <property name="detectingrange">80</property>
      <property name="energy">4000</property>
      <property name="brainClass">warbot.myTeam.HomeKiller</property>
      <graphic-element class="warbot.kernel.GBasicBody">
          <property name="imageaddress">images/warbot/cyanrocketlauncher.gif</property>
          <property name="displaylabel">false</property>
    </graphic-element>
    <action description="toggle show detection range">
          <java-method name="toggleShowDetect"/>
    </action>
    <action description="toggle show user message">
          <java-method name="toggleShowUserMessage"/>
    </action>
    <action description="toggle show energy level">
          <java-method name="toggleShowEnergyLevel"/>
    </action>
  </node-desc>
```

Programmer en Warbot

- **Les types d'architecture et de mode de programmation**
- **Dirigé par les perceptions**
  - Action située
  - Subsumption
- **Automates à états finis**
  - Standard
  - Hiérarchique
  - avec réflexes
- **Tâches en compétition (EMF)**
- **BDI**

Architecture dirigée par les perceptions

- **Architecture dirigée par les événements**
- **Boucle globale de test des événements**
- **Action située: perception → action**

Actions situées

- **Comportement est lié aux événements (percepts)**
  - Pas de mémorisation de l'environnement
    - Pur: perception immédiate (pas de mémoire)
    - Impur (avec apprentissage): mémorisation uniquement des états internes passés..
  - Donne une grande importance à l'environnement
    - Buts et obstacles sont dans l'environnement
    - Communication par dépôt de marques (indices et chemins)
- **Règles d'action**
  - Si <état interne> and <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 (tente de) 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 (tente de) prend le café

Récupérer de l'énergie par des explorateurs

- **Principe:**
  - Un explorer doit récupérer de l'énergie sous forme de nourriture ('food')
  - Il doit rapporter la nourriture à la base

Règle d'exploration
si je ne porte rien
et je ne perçois pas de minerai
alors marche aléatoire

Règle trouver
Si je ne porte rien
et je perçois de la nourriture
alors aller vers la nourriture

Règle prendre
Si je ne porte rien
et je perçois de la nourriture
et la distance avec la nourriture < dist-prendre
alors prendre la nourriture

Règle rapporter
Si je porte du minerai
et je ne suis pas à la base
alors retourner à la base

Règle déposer
Si je suis à la base
et j'ai de la nourriture
alors déposer de la nourriture

Implémentation

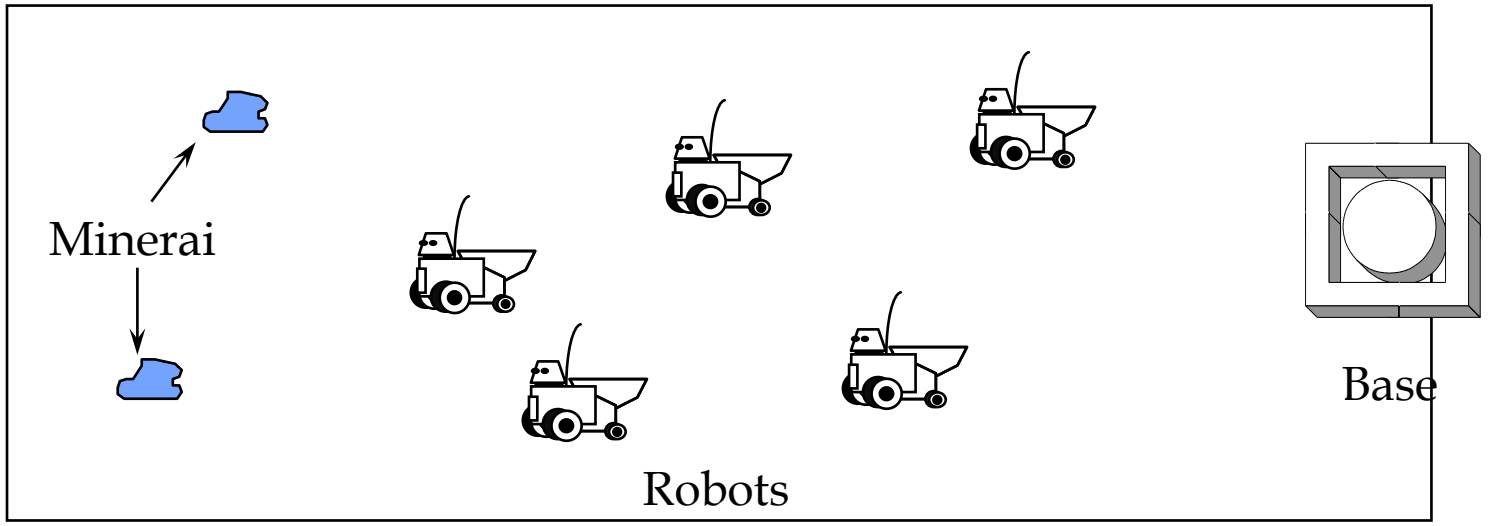
```
percepts = self.getPercepts()
        if len(percepts)>0:
                food = 0
                ennemyHome = 0
                for p in percepts: #find the closest Hamburger... and get it
                        ....
                        if (p.getPerceptType()=='Home' and p.getTeam() == self.getTeam()):
                                if (self.distanceTo(p)<=3) and  self.isBagFull()) :
                                        for i in range(len(bag)):
                                                self.drop(i)
                                                return
                        if (not food and not atHome and not self.isBagFull() and p.getPerceptType()=='Food'):
                                pmin=p
                                bag = self.getBagPercepts()
                                if self.distanceTo(pmin)<=3:       #if close enough
                                        self.take(pmin)                         #take it
                                        return
                                else:                                           #else go towards it
                                        self.setHeading(self.towards(pmin.getX(),pmin.getY()))
                                        self.move()
                                        return
                        if (self.isBagFull()):   #retour à la base
                                 self.setHeading(self.towards(homeBaseX,homeBaseY))
                                self.move()
                                return
self.move()       #default move
```

Critique

- **Devient rapidement très complexe**
  - Nombre de percepts et d'états à prendre en compte
- **Nécessite des priorités**
  - Il y a des actions préférables à d'autres
  - Ex: éviter obstacles (ou missiles) par rapport à rentrer droit à la base

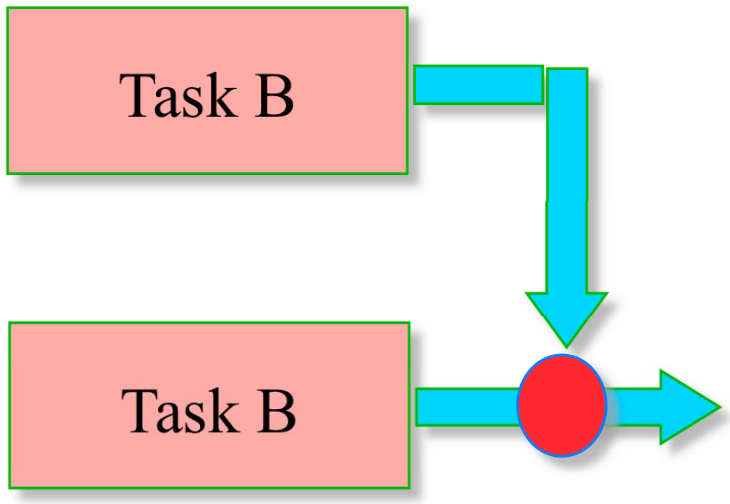
Récupérer de l'énergie par des exploreurs

- **Un explorer doit récupérer de l'énergie sous forme de 'food'**
- **Problème:** comment décrire leur comportement et leur technique de coordination afin qu'ils remplissent leur mission.
- **Hypothèses**:
  - Ils ne peuvent pas s'envoyer des messages par radio.
  - Les robots sont tous similaires (totipotence)



Architectures d'agents réactifs #2

- **Subsumption architecture (R. Brooks)**



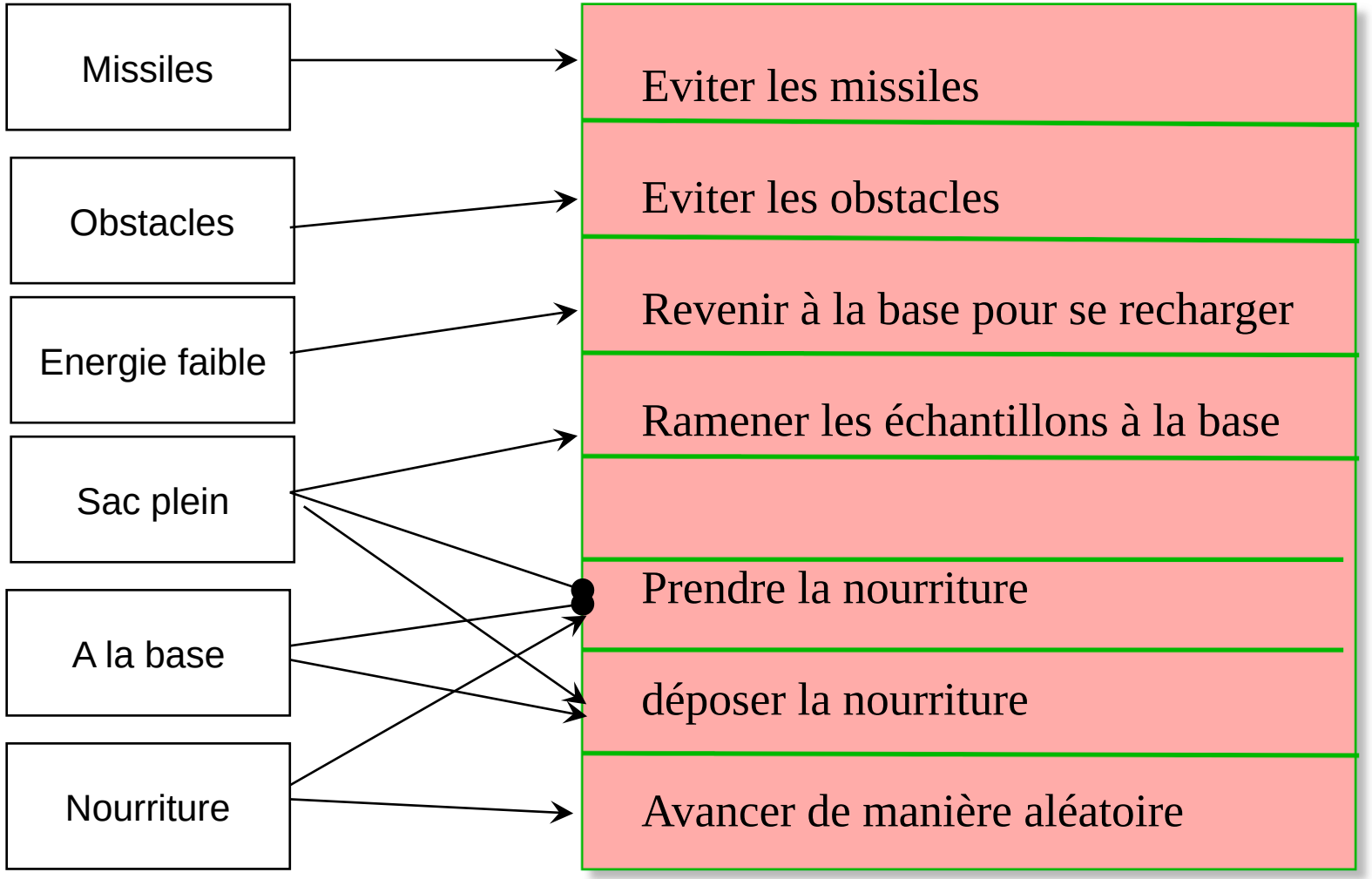
Eviter les obstacles

Maintenir un niveau d'énergie

Accomplir une tâche productive

Avancer de manière aléatoire (comportement par défaut)

Exemple récupération de nourriture



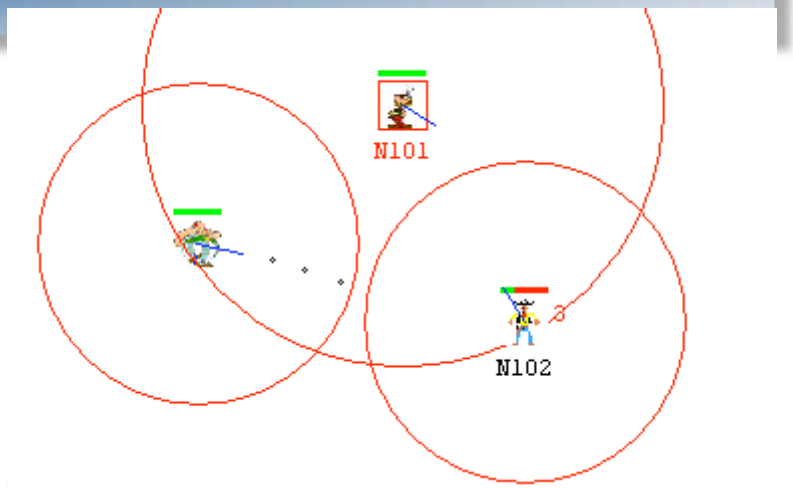
Ex. dans Warbot

- **Explorateurs**

Si un lanceur de missiles ennemi est repéré : il faut prendre la direction opposée à l'ennemi, sans toutefois s'obstiner si dans cette direction se trouve un obstacle…
Si l'agent est attaqué : il doit demander de l'aide
Si la base ennemie est trouvée : envoi des coordonnées de cette base à tous les lanceurs de missiles
Déplacement aléatoire

Plus fin.. dans Warbot

- **Définition de tâches avec priorité dans les percepts**



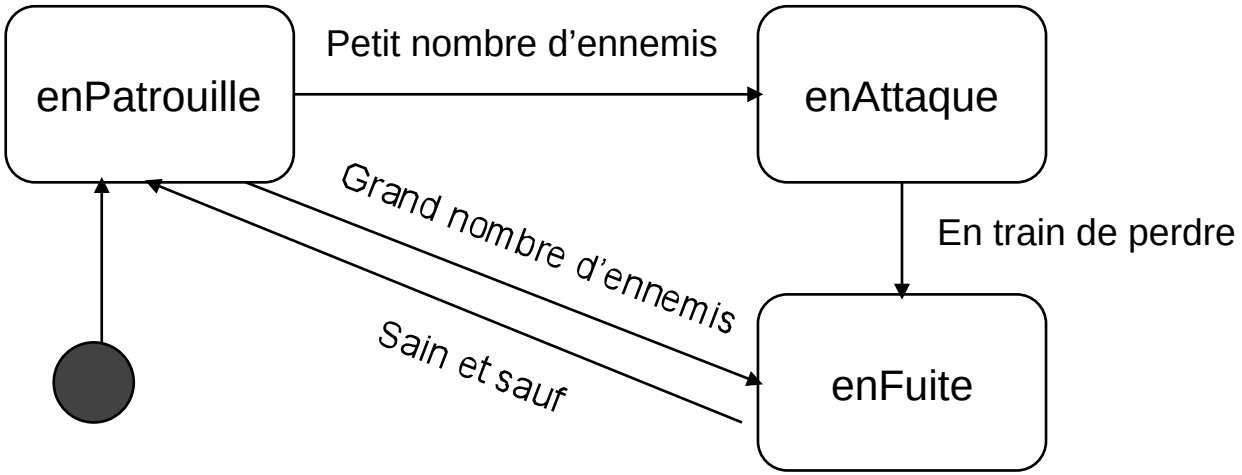

```
if (lesBasesEnnemis.size() > 0) {priorite = 1 ; println("P1");}
else if (lesRockets.size() > 0) {priorite = 2; println("P2");}
else if (lesKillersEnnemis.size() > 0 && allBaseTrouvee==false) {priorite = 3 ;println("P3");}
else if (mesKillersAmisMin.size() > 0) {priorite = 4 ;println("P4");}
else if (mesExplorersAmisMin.size() > 0) { priorite = 5 ; println("P5");}
else if (lesObstaclesMin.size() > 0){if(parcourir==false){priorite = 6 ; println("P6");}else println("P6:
else if (isBagFull()) {priorite = 7 ;println("P7");}
else if (laNourriture.size() > 0) {priorite = 8 ;println("P8");}
else if (mesBasesAmisMin.size() > 0) {priorite = 9 ;println("P9");}
else if (allBaseTrouvee && lesKillersEnnemis.size() > 0 && mesKillersAmis.size()>0)
        {priorite = 10 ; println("P10");}
else if (allBaseTrouvee && lesKillersEnnemis.size() > 0 )
        {priorite = 11 ; println("P11");}
else if (allBaseTrouvee && mesKillersAmis.size()>0) {priorite = 12 ; println("P12");}
```

Contrôle en fonction des priorités

```
//_____________________ACTIONS _____________________________________
bouge = true ;
switch(priorite)
{
        case 1 :  msgComElements(lesBasesEnnemis, PerceptBase) ; priorite = 0 ; break ;
        case 2 :  parcourir=false ; analyseRockets(lesRockets, lesKillersEnnemis); priorite = 0 ; break ;
        case 3 :  parcourir=false ; eviterEnnemis(lesKillersEnnemis,90,130,90); priorite = 0 ; break ;
        case 4 :  parcourir=false ; eviterCollision(mesKillersAmis); priorite = 0 ; break ;
        case 5 :  parcourir=false ; eviterCollision(mesExplorersAmis); priorite = 0 ; break ;
        case 6 :  eviterObstacles(lesObstacles, 130, 65); priorite = 0 ; break ;
        case 7 :  parcourir=false ; gestionNourriture(mesBasesAmis, laNourriture); priorite = 0 ; break ; /
        case 8 :  parcourir=false ; gestionNourriture(mesBasesAmis, laNourriture) ; priorite = 0 ; break ;
        case 9 :  parcourir=false ; eviterCollision(mesBasesAmis); priorite = 0 ; break ;
        case 10 : parcourir=false ;
                  if(mesKillersAmis.size()>=lesKillersEnnemis.size())msgAmisAttaquer(mesKillersAmis, lesKil
                  else msgAmisFuir(mesKillersAmis, lesKillersEnnemis);
                  priorite = 0 ; break ;
        case 11 : parcourir=false ; suivreElement(lesKillersEnnemis); priorite = 0 ; break ;
        case 12 : parcourir=false ; suivreElement(mesKillersAmis); priorite = 0 ; break ;
        default : if(dirDGExplorer!=""){
                        if(parcourir==false||boolDebJeu==true){// La base me donne ma direction
                                println("parcourir==false||boolDebJeu==true");
                                if (dirHBExplorer == "bas") setHeading(90);
                                else setHeading(270);
                                parcourir = true ;
                                boolDebJeu=false;
                         }
                         if(parcourir==true){
                         println("Je parcoure ma zone");
                         parcourirZone(lesObstacles);
                         }
                }
```

Machines à états finis (FSM, automates)

- **Etats**: un état ou une activité de l'agent
- **Evénement**: quelque chose qui se passe dans le monde extérieur (ou intérieur à l'agent) et qui peut déclencher une action
- **Action**: quelque chose que l'agent fait et aura pour conséquence de modifier la situation du monde et de produire d'autres événements
  - Souvent l'action est impliquée par l'état



Implémentation de FSM #1

```
void Garde::FSM(TypeEtat etat){
  switch(etat){
    case etat_enFuite:
         echapperEnnemis();
         if (sauf())
            changerEtat(etat_Patrouille);
         break;
    case etat_Patrouille :
         patrouiller();
         if (menace())  {
            if (plusFortQueEnnemis())
                  changerEtat(etat_attaque);
            else
                  changerEtat(etat_enFuite);
          } break;
    case etat_attaque :
         seBattre();
         if (ennemisVaincus())
            changerEtat(etat_enFuite);
         else if (plusFaibleQueEnnemis())
            changerEtat(etat_enFuite);
        break;
  }//end switch
}
```

- **Coder l'automate directement dans le code**
  - grand 'switch' en fonction des états

Implémentation à l'aide de tables

- **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	Sauf	Patrouille
Patrouille	Patrouiller	Menace ET ennemisPlusfort	enFuite
--		Menace Et ennemisMoinsfort	enAttaque
enAttaque	seBattre	ennemisVaincus	Patrouille
--		ennemisPlusFort	enFuite

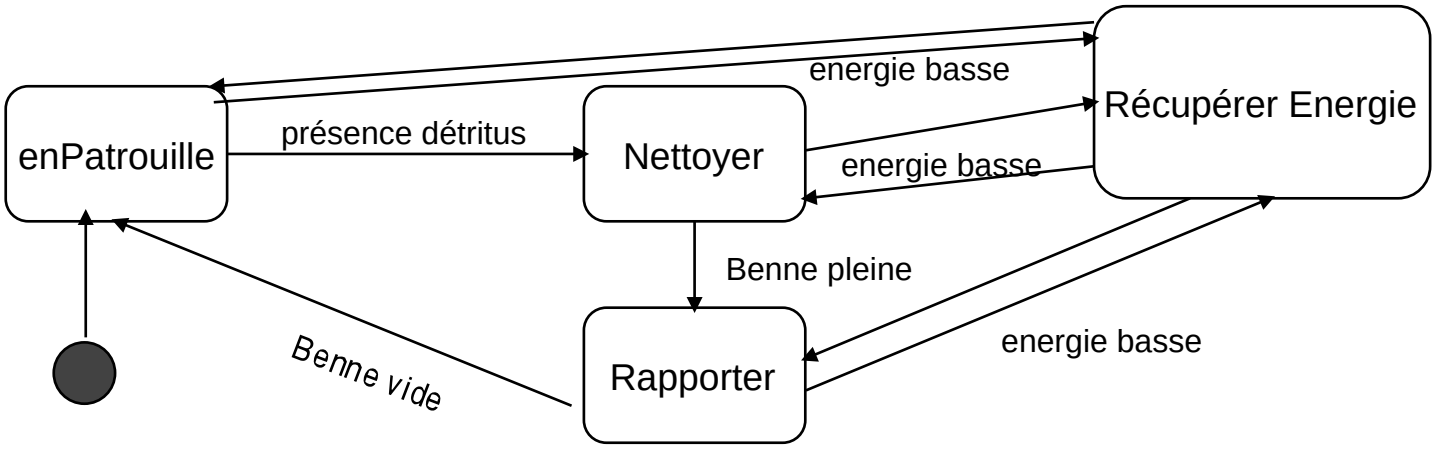
Approche Objet

- **Classe Etat**
  - Chaque état est implémenté sous la forme d'une classe
  - Création ou recherche dans un dictionnaire

```
class Etat {
    abstract void exec(Agent ag);
}

class EtatEnFuite extends Etat {
    void exec(Agent ag){
        ag.echapperEnnemis();
        if (ag.sauf())
           ag.changerEtat(
                Etat.look("EtatPatrouille"));
    }
}

Class Agent {
   State etatCourant;

   void do(){
      etatCourant.exec();
   }
   void changerEtat(Etat etat){
      etatCourant = etat;
   }
   boolean sauf(){… }
   void echapperEnnemis() { … }
}
```

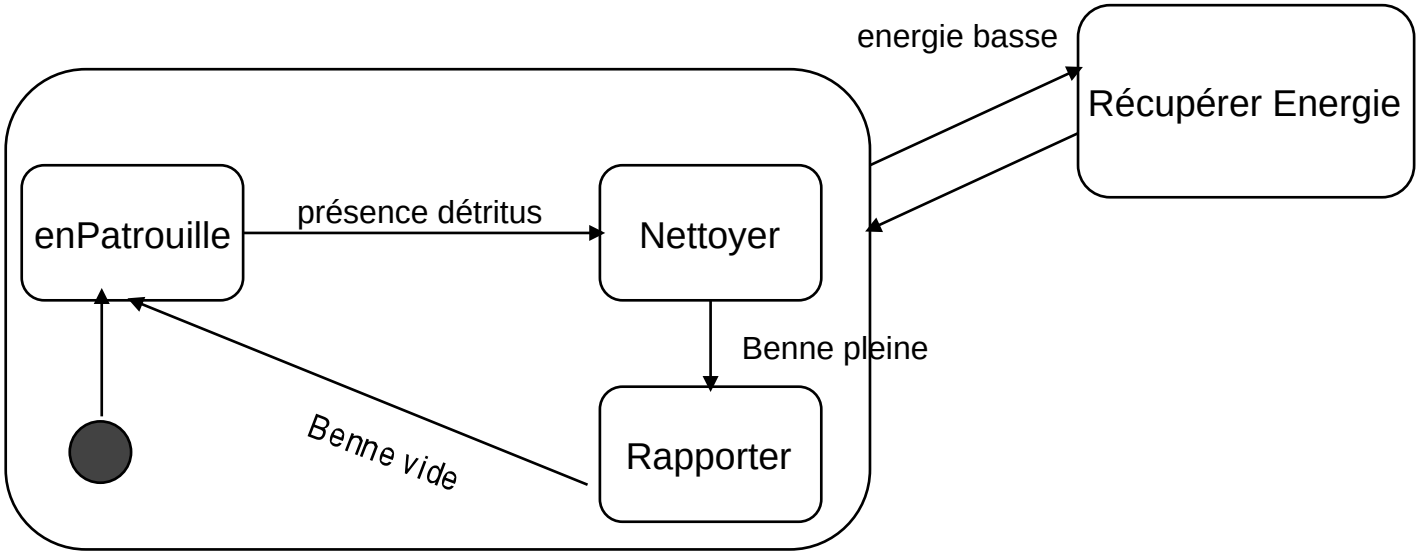
Machine à états finis hiérarchique

- **Si l'on a besoin de caractériser des "modes" comportementaux différents, exprimés de manière hiérarchique**
  - Ex:
    - ☞ Un robot doit récupérer de l'énergie quoi qu'il fasse par ailleurs



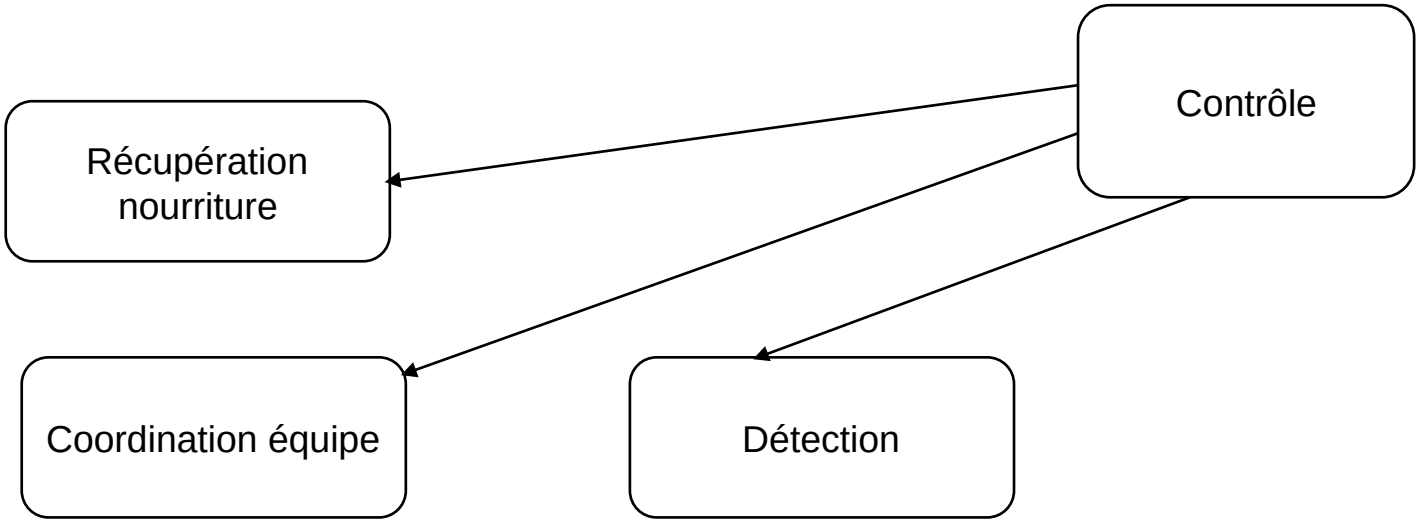
FSM hiérarchique

- **Construction d'activités correspondant à un automate**



Spécialisation

- **Plusieurs activités correspondant à des spécialisations temporaires**
  - Explorer dédié à la récupération de nourriture vs Explorer dédié à la coordination d'une équipe de tirs vs Explorer dédié à la détection des ennemis
  - Mais comment revenir au contrôle?



Problèmes des machines à états finis

- **Ne sait pas prendre en compte les actions réflexes**
  - Exemple: si je détecte la base ennemie, j'envoie les coordonnées de la base à tout le monde
  - Nécessite d'introduire des mécanismes supplémentaires d'actions réflexes
  - Difficulté de passage de 'modes' dans le cas d'activités en compétition