

Warbot: la guerre des robots

Description et programmation

Jacques Ferber
ferber@lirmm.fr

LIRMM - Université de Montpellier II

Novembre 2017

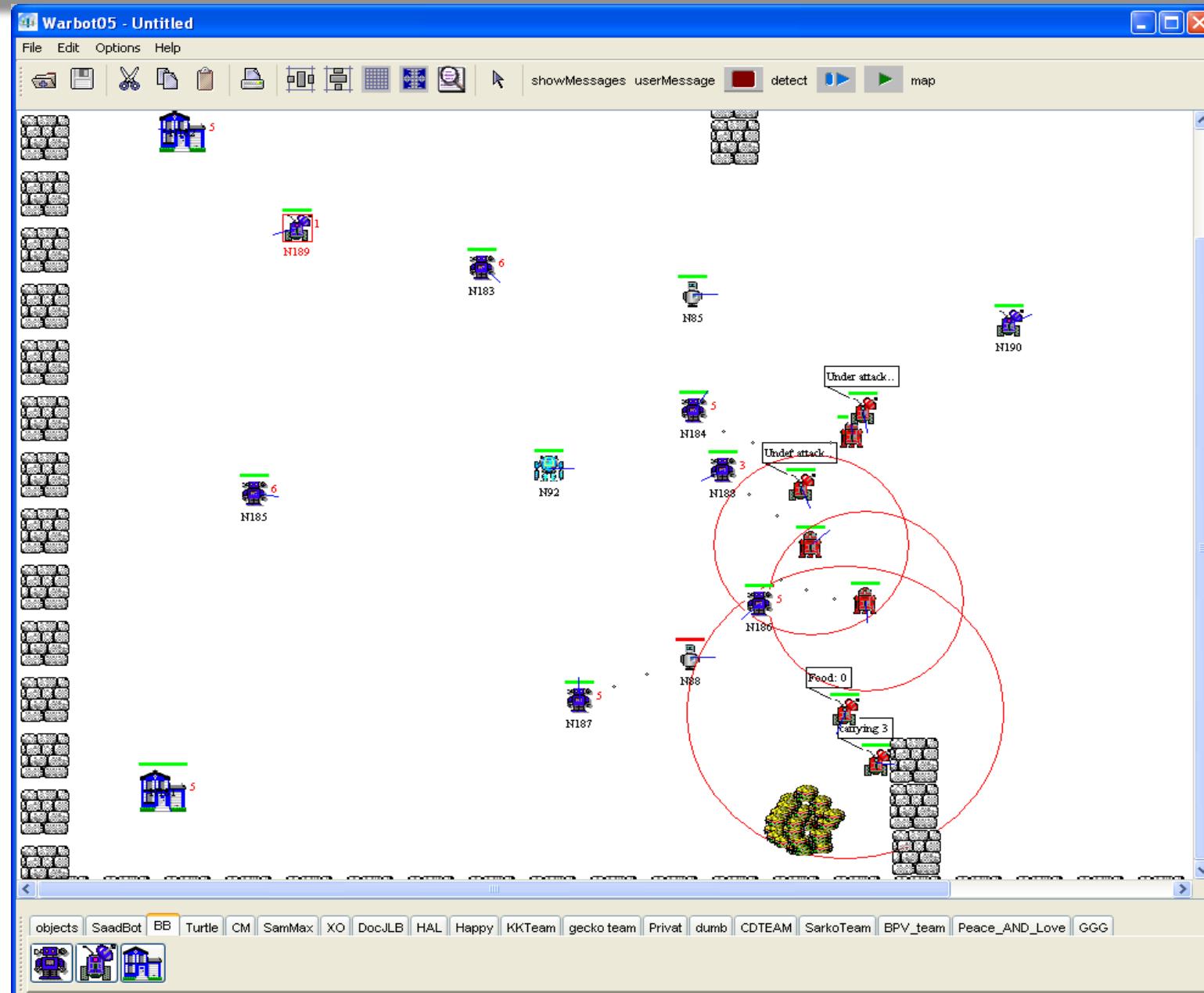
Plan

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

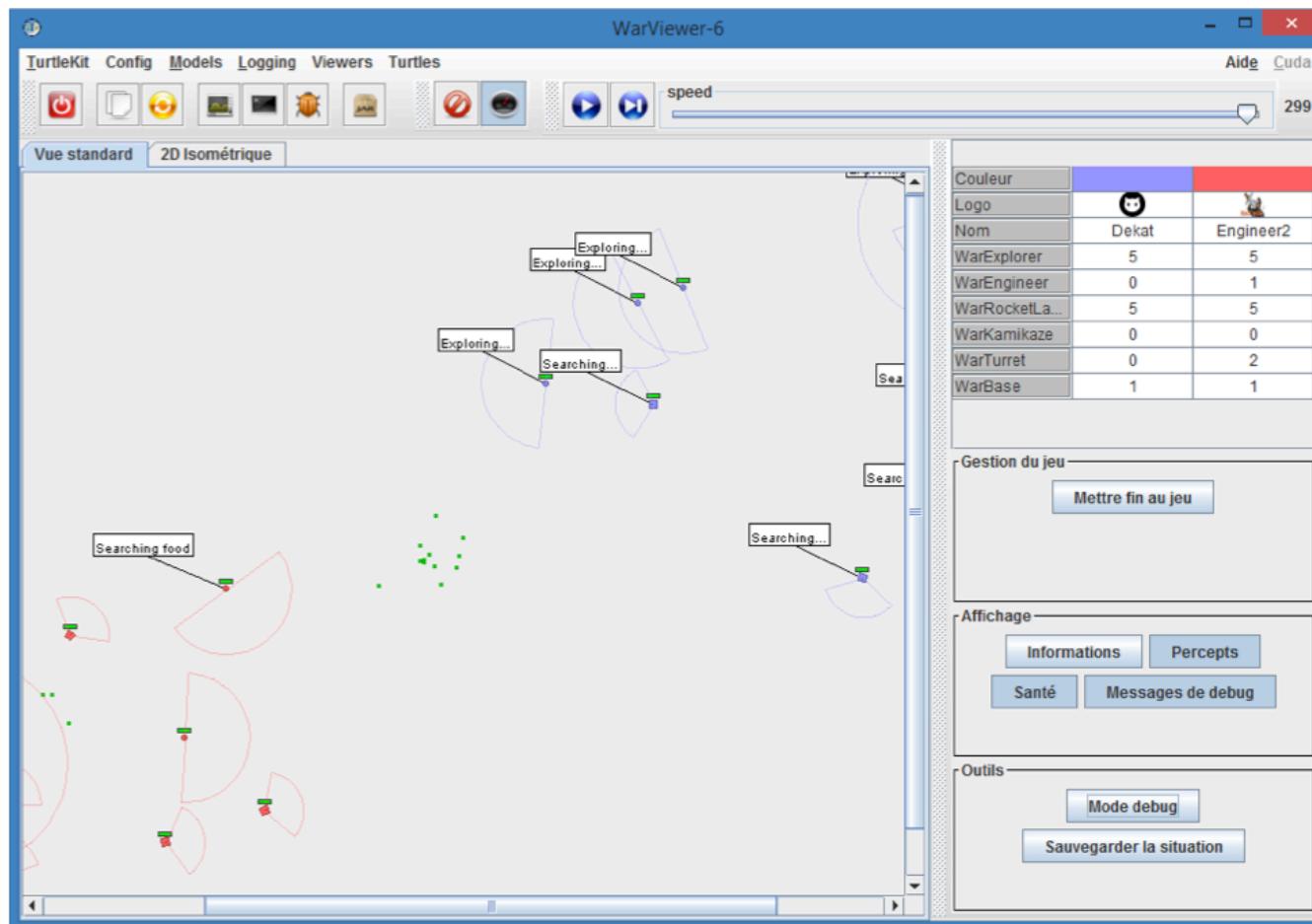
Principes

- ◆ 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



Warbot 3 - Nouvelle interface



Les types de corps (initialement)

◆ 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 robots "généraux"

WarBase:

- AngleOfView: 360.0 DistanceOfView: 80.0 Cost
 : 1000 MaxHealth: 6000 BagSize: 1000000
 Armor: 10

WarTurret:

- AngleOfView: 180.0 DistanceOfView: 50.0 Cost
 : 1200 MaxHealth: 4000 BagSize: 10
 TicksToReload: 5 Armor: 10

WarExplorer:

- AngleOfView: 180.0 DistanceOfView: 50.0 Cost
 : 200 MaxHealth: 200 BagSize: 4 Speed
 : 2.0 Armor: 0

WarEngineer:

- AngleOfView: 150.0 DistanceOfView: 30.0 Cost
 : 1000 MaxHealth: 1500 BagSize: 4
 Speed: 1.0 MaxRepairsPerTick: 20 Armor: 0

Les "guerriers"

WarLight:

- AngleOfView: 120.0 DistanceOfView: 40.0
Cost: 250 MaxHealth: 200 BagSize: 4
Speed: 1.8 TicksToReload: 1 Armor: 1

WarHeavy:

- AngleOfView: 120.0 DistanceOfView: 30.0
Cost: 500 MaxHealth: 800 BagSize: 4
Speed: 0.8 TicksToReload: 5 Armor: 20

WarRocketLauncher:

- AngleOfView: 120.0 DistanceOfView: 20.0
Cost: 1000 MaxHealth: 200 BagSize: 4
Speed: 1.0 TicksToReload: 25 Armor: 1

WarKamikaze:

- AngleOfView: 150.0 DistanceOfView: 20.0
Cost: 3000 MaxHealth: 3000 BagSize: 4
Speed: 1.0 Armor: 0

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

◆ Il n'y a pas d'information globale/partagée

- Tout est local

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()** : tente de manger l'entité e (si elle est comestible, sinon il ne se passe rien).
- **boolean take()** : 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 disparaît de l'environnement.
- **boolean drop()** : dépose l'entité disponible dans le sac du robot. Lorsqu'un robot dépose quelque chose, cela réapparaît dans l'environnement à l'endroit où se trouve le robot.
- **boolean give()**: donne à une autre unité

◆ Tirs (uniquement les lanceurs de missiles)

- **void fire()**: tire (en fonction de l'unité)

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(AgentAddress agent, String act, String[] cont):** envoie un message de performatif act 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 rafraîchis 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 »

● Méthodes

- ☞ **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 = getPercepts();
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
}
```

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**
- ◆ **(cf. le cours sur les architectures réactives)**

Implémentation

```
def actionWarExplorer():
    if getPerceptsEnemiesWarBase():
        broadcastMessageToAll("EnemyBase", "")

    for percept in getPerceptsFood():
        sendMessageToExplorers("FoodFound", "")
        setDebugString("View Food");
        if(pickableFood(percept) and isNotBagFull()):
            setDebugString("Take food")
            followTarget(percept)
            return take()
        elif (isNotBagFull()):
            followTarget(percept)

    if(isBagFull()):
        setDebugString("Bag full return base");
        __percept = getPerceptsAlliesWarBase();

        if(haveNoTargets(__percept)):
            for message in getMessages():
                if(isMessageOfWarBase(message)):
                    followTarget(message)

                    sendMessageToBases("whereAreYouBase", "");
            else :
                __base = __percept[0];

                if(isPossibleToGiveFood(__base)) :
                    giveToTarget(__base);
                    return give();
                else:
                    followTarget(__base);
                    return move();
            else:
                setDebugString("Look for food");

            if (isBlocked()) :
                RandomHeading()

        return 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

Utilisation d'une FSM

- ◆ Python: voir exemple: WarExplorer_FSM.py
- ◆ Java: FSM_Exemple