

Robot Ramasseurs De Balles

Système multi-agents

2016-2017

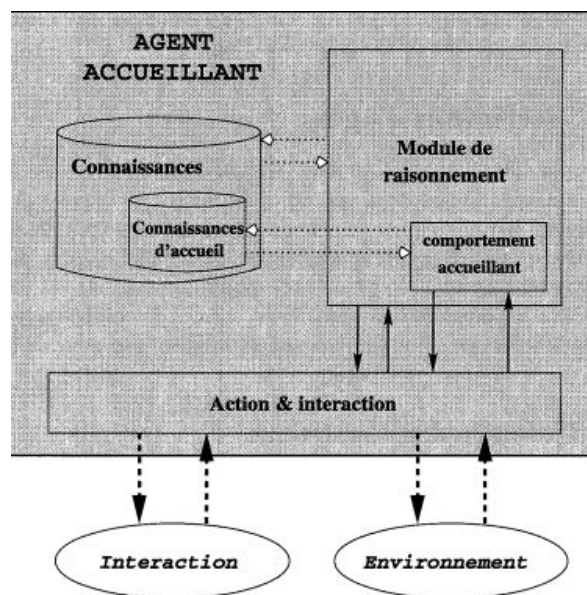
1. Introduction

a. Systèmes multi-agents:

Les systèmes multi-agent sont composés d'entités informatiques distribuées qui interagissent entre elles. Il détermine les règles permettant de concevoir des systèmes intelligence artificiel fondé sur des règles. Il s'intéresse aux comportements collectifs produits par des interactions de plusieurs entités autonomes et flexibles. Ces entités peuvent opérer de façons collectives pour accomplir les tâches complexes. Malgré que les caractéristiques de modélisation dépendent de l'application, mais il y a bien certains caractéristiques qui sont commune aux agents : **La coopération, la coordination et la communication.**

b. Un agent:

C'est une entité autonome, réelle ou abstraite qui est capable d'agir sur elle-même et sur son environnement, qui, dans un univers multi-agents, peut communiquer avec d'autres agents et dont le comportement est la conséquence de ses observations, de ses connaissances et des interactions avec d'autres agents. Cependant, les agents peuvent être classés en deux catégories principales selon leur comportement et leur complexité de raisonnement afin de séparer les agents dites « intelligentes » des agents moins « intelligentes ». On parle donc des agents cognitifs et agents réactifs.

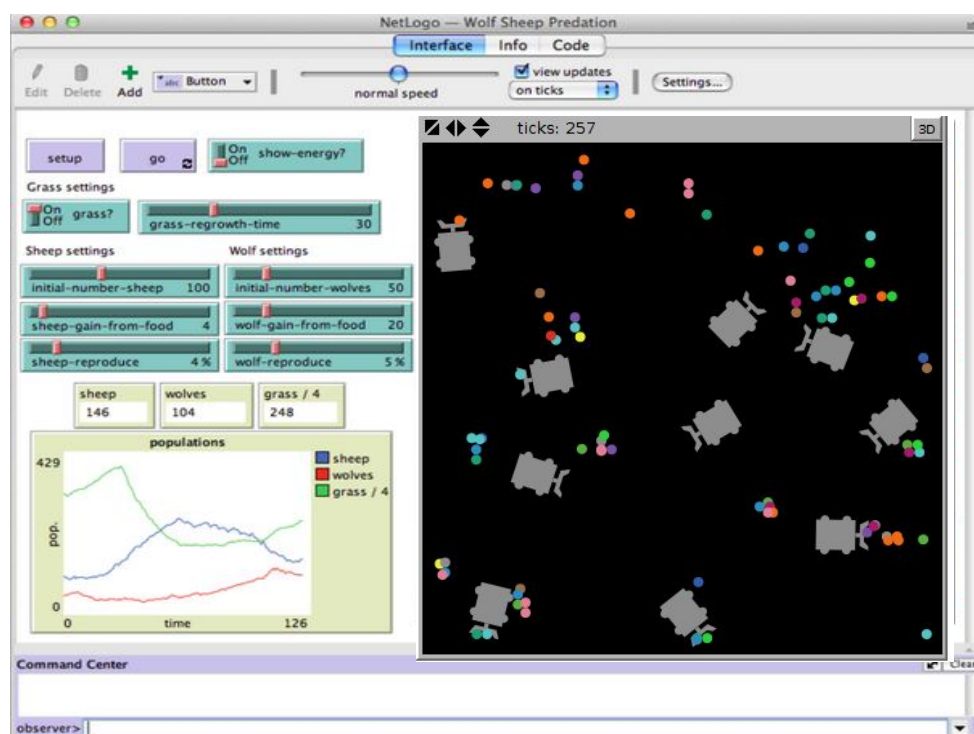


- **Les agents cognitifs** : sont fondés sur la coopération d'agents capables à eux seuls d'effectuer des opérations complexes. Un système cognitif comprend un petit nombre d'agents qui disposent d'une capacité de raisonnement sur une base de connaissances pour traiter les informations diverses liées au domaine d'applications. Ces types d'agents peuvent être assimilés à des systèmes experts distribués.
- **Un agent réactif** : évolue parmi un nombre important de ses semblables et c'est le résultat des interactions entre leur activité qui donne une impression de comportement global « intelligent ». Ces agents ne possèdent de

représentations de leur environnement. De ce fait toutes les informations relatives à leur comportement se trouvent dans l'environnement et leurs réactions dépendant uniquement de la perception qu'ils peuvent en avoir.

2. Présentation de l'outil Netlogo

Un des outils largement utilisé pour la simulation des systèmes multi-agent est le logiciel Netlogo qui est un logiciel open source fourni sous la licence GPL. Il est à la fois un langage de programmation et un environnement de modélisation pour le développement des systèmes multi-agents. De base, il a été développé en Java par l'Université Northwestern pour des fins pédagogiques, et de nos jours Netlogo est beaucoup utilisé dans le milieu professionnel et cela est dû à la facilité d'utilisation qui permet de modéliser des phénomènes sans pour autant avoir des connaissances approfondies en programmation. Il est particulièrement adapté à la modélisation de systèmes complexes sur un parcours de temps.

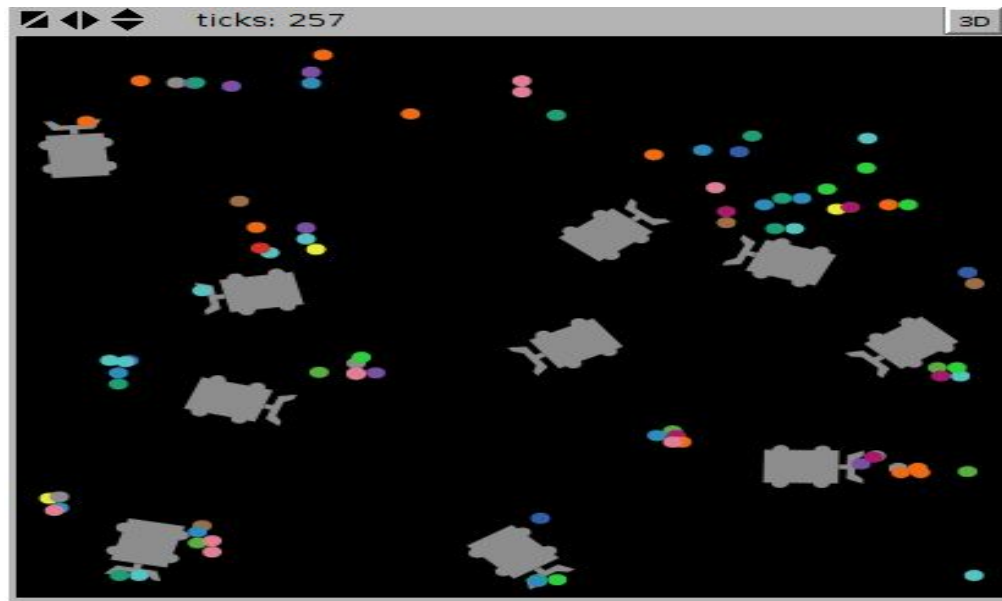


3. Description du modèle

Dans le cadre du cours de système multi-agents, il nous a été demandé de simuler des robots ramasseurs de balles. La simulation à base d'agents a été largement utilisée pour modéliser des phénomènes complexes comme par exemple le ramassage des balles de tennis ou golf, des phénomènes sociaux, biologiques, géographiques... Cependant, dans la réalisation de cette simulation plusieurs contraintes ont été imposées. Des robots capables d'effectuer en toute autonomie de trouver des balles et les transporter pour former un tas composé de trois balles. Ils devraient être capables de transporter une à deux balles en même temps, et devraient être aussi dans l'impossibilité de transporter trois balles en même temps. De ce fait, lorsqu'il rencontre une troisième balle, les agents devraient se voir dans l'obligation d'abandonner ce tas. Le programme doit compter le nombre de tas formés par chaque robot et l'afficher l'ensemble de ces résultats dans un plot.

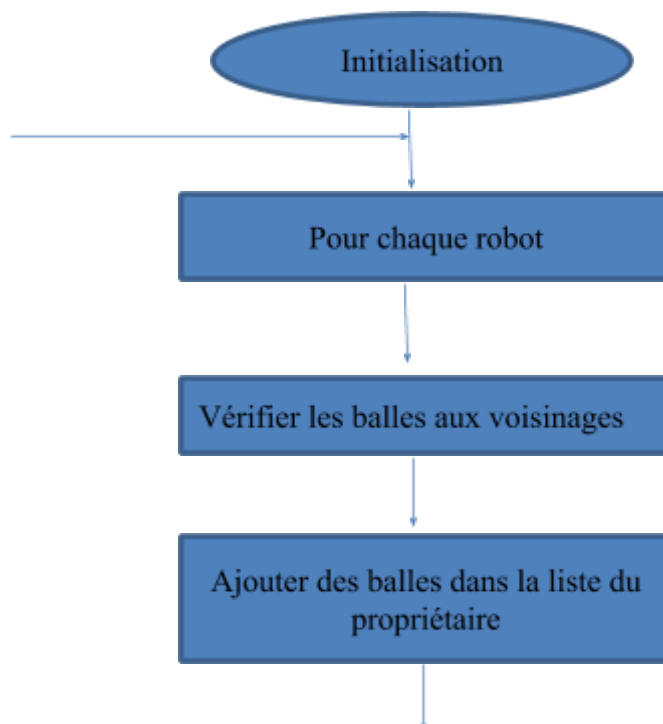
4. Solution proposée

Le programme comprend deux parties : l'initialisation par la fonction setup et la boucle par la fonction go. L'étape d'initialisation dans le logiciel netlogo est très importante pour le bon déroulement du programme. Ce programme contient l'initialisation des robots mobiles dont le nombre total dépend de l'utilisateur avec un nombre comprise entre 1 à 100 et l'initialisation des balles qui se font aléatoirement et répartie sur toute la surface. Dans la boucle go le programme parcourt chaque robot et vérifie le voisinage autour, s'il trouve une ou deux balles alors il transporte ces balles en se déplaçant d'un tick. Lorsqu'il rencontre une troisième, alors il considère comme que les trois balles sont lourdes à transport et il abandonnait ce tas pour faire une rotation de 180 degrés qui correspondent à la direction opposé de l'étape précédente.



5. Diagramme de l'algorithme

Afin de mieux comprendre le déroulement du programme, j'ai réalisé un organigramme du programme.



```
graph TD; A[Si le nombre de balle >= 3, alors  
déposer le tas et incrémenté le tas  
total du robot];
```

Une idée que je trouve intéressant pour améliorer ce projet serait de gérer la communication pour éviter les collisions entre les robots eux. De plus, gérer des pertes d'énergie des robots.

Conclusion

En conclusion, Ce programme permet de simuler des robots complétement autonomes qui détecter et ramasser des balles pour former un tas.

D'un point de vue plus général, Ce projet a grandement consolidé mes connaissances sur les notions d'agents cognitifs et réactifs dans le domaine de systèmes multi-agents. Il m'a permis d'effectuer des progrès considérables dans les domaines que je ne connais pas. De plus j'ai pu découvrir un langage et être familier au langage et l'environnement du logiciel Netlogo.

J'ai également rencontré des problèmes sur le principe du déplacement de la balle par le robot.

```
breed [ Robots robot ]
Robots-own [robot-tas robot-nb-tas devine-x devine-y devine-deplacer]
```

```
breed [ Pieces piece ]
Pieces-own[piece-propietaire]
```

```
globals [
  box-edge
]
```

```
to setup
  clear-all
  set-default-shape pieces "circle"
  set-default-shape robots "robot_logo"
  set box-edge max-pxcor - 1
```

```
  setup-pieces Nb-pieces
  setup-robots Nb-Robots
```

```
  reset-ticks
end
```

```
to setup-pieces [ number ]
  create-pieces number [
    set piece-propietaire -1
    set size 2
    position-randomly
  ]
]
```

```
end
```

```
to setup-robots [ number ]
  create-robots number [
    set color white
    jump random-float max-pxcor
    set size 17
    set robot-tas 0
    set robot-nb-tas 0
    set devine-x 0
    set devine-y 0
  ]
end
```

```
to position-randomly
  setxy one-of [1 -1] * random-float (box-edge - 0.5 - size / 2)
  one-of [1 -1] * random-float (box-edge - 0.5 - size / 2)
end
```

```
to depose-tas[value-propietaire]
```

```
  ask pieces with[piece-propietaire = value-propietaire][  
    set piece-propietaire -1  
  ]  
end
```

```
to go
```

```
  ask Robots[
```

```
    let gauche random 30  
    let droite random 30  
    let value-propietaire who
```

```
    ifelse robot-tas > 2
```

```
    [  
      set robot-nb-tas (robot-nb-tas + 1)  
      set robot-tas 0  
      depose-tas who  
      rt 180  
      fd 8  
    ]  
    [  
      set devine-deplacer 0  
      set devine-deplacer (provide-move gauche droite)  
      while[devine-deplacer != 1][  
        set devine-deplacer (provide-move gauche droite)  
      ]  
      let coord-devine-x devine-x  
      let coord-devine-y devine-y
```

```
      show sum [count turtles-here] of neighbors
```

```
      set robot-tas (go-to-objectif xcor ycor coord-devine-x coord-devine-y robot-tas  
value-propietaire)
```

```
      lt gauche  
      rt droite  
      fd 1
```

```
    ]  
  ]  
  tick  
end
```

```
to-report provide-move [gauche droite]
```

```
if not can-move? 1 [  
  rt 80  
  report -1]  
lt gauche  
rt droite  
fd 8
```

```
if not can-move? 1 [  
  fd -8  
  rt (- droite)  
  lt (- gauche)
```

```
  rt 80  
  report -1  
]  
fd -3  
set devine-x xcor  
set devine-y ycor  
fd -5  
rt (- droite)  
lt (- gauche)
```

```
  report 1  
end
```

```
to-report go-to-objectif[coord-x coord-y coord-devine-x coord-devine-y nb-tas  
value-propietaire]
```

```
let tas 0  
let diff-x (abs(coord-x - coord-devine-x))  
let diff-y (abs(coord-y - coord-devine-y))  
ask pieces with[piece-propietaire = value-propietaire][  
  if ((xcor <= (coord-x + 5)) and (xcor >= (coord-x - 5)))[  
    set tas (tas + 1)  
    if(nb-tas > 2)  
      [ stop  
        report tas  
      ]  
    set xcor coord-devine-x  
    set ycor coord-devine-y
```

```
  if(tas = 1)[  
    ifelse(diff-x > diff-y)  
      [ set ycor (ycor + 1) ]  
      [ set xcor (xcor + 1) ]  
  ]  
  if(tas = 2)[
```



```

        ifelse(diff-x > diff-y)
          [ set ycor (ycor - 1) ]
          [ set xcor (xcor - 1) ]
      ]
  ]
]
if(tas < 3)[
  ask pieces with[piece-propietaire = -1][

    if ((ycor <= (coord-y + 5)) and (ycor >= (coord-y - 3)))[
      if ((xcor <= (coord-x + 4)) and (xcor >= (coord-x - 4)))[
        set tas (tas + 1)
        if(tas > 2)
          [ stop
            report tas
          ]
        set xcor coord-devine-x
        set ycor coord-devine-y
        set piece-propietaire value-propietaire

      ]
    ]

    if(tas = 1)[
      ifelse(diff-x > diff-y)
        [ set ycor (ycor + 1) ]
        [ set xcor (xcor + 1) ]
    ]
    if(tas = 2)[
      ifelse(diff-x > diff-y)
        [ set ycor (ycor - 1) ]
        [ set xcor (xcor - 1) ]
    ]
  ]
]
]
]
report tas
end

```