

Algorithmes d'exploration et de mouvements

HMIN213

Flocking et évitement

V2.4 – Février 2018

Jacques Ferber

Mouvements de foule



Types de mouvements de foule

- ◆ **Coordination de mouvements**
 - Flocking (escadrille)
 - Entourer
 - Suivre, éviter
 - Se mettre en ligne

- ◆ **Formation dépendant de la nature des agents**
 - Ex: agents d'exploration, chasseurs, croiseurs
 - Se mettre dans une formation quelconque
 - ☞ Carré, rond, triangle, en fonction des rôles, etc..

Qu'est ce qu'une escadrille, troupeau (flock)

definition: (flock) un groupe d'oiseaux, de poissons de mammifères, d'humains (d'agents) qui avancent ensemble en formation.



Boids!

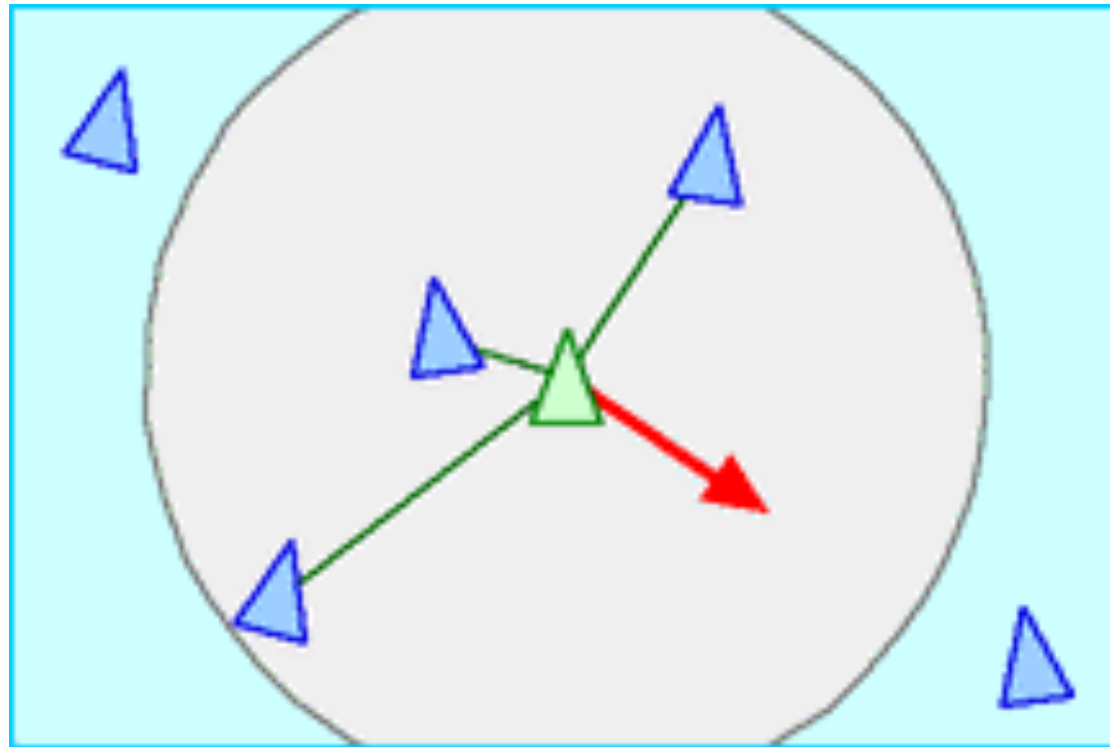
- ◆ “boids” vient de “bird-oids” - Article fondateur de Craig Reynolds en 1986.
- ◆ Mécanisme semblable à des systèmes de particules, mais avec une orientation
- ◆ Mouvement dirigé par le comportement (Behavior-based motion)
 - Algorithme distribué, pas de calcul centralisé.
- ◆ Vitesse constante
- ◆ Contrainte uniquement en terme de rotation
- ◆ Tous les jeux videos utilisant des agents en formation utilisent cette approche, car elle est très simple !!

Mouvement de troupeau (flocking)

- ◆ **Les Boids doivent se coordonner avec leur voisins**
- ◆ **Deux tendances principales**
 - Rester près des autres
 - Eviter les collisions avec les autres
 - Se mettre dans la même direction
- ◆ **Dans la nature, les mouvements ont évolués**
 - Prédation
 - Trouver de la nourriture
 - Rencontre amoureuse/sexuelle (mating)

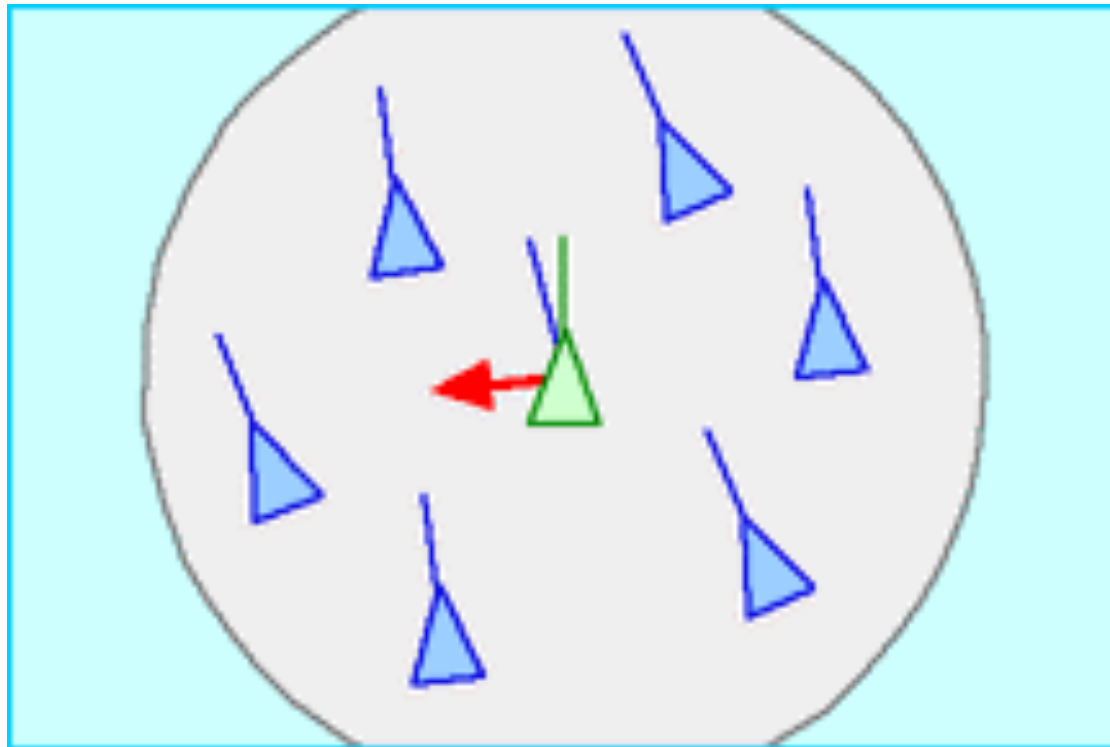
Flocking les 3 règles

- ◆ 1- Répulsion: pour éviter les collisions avec les voisins on part de l'autre côté



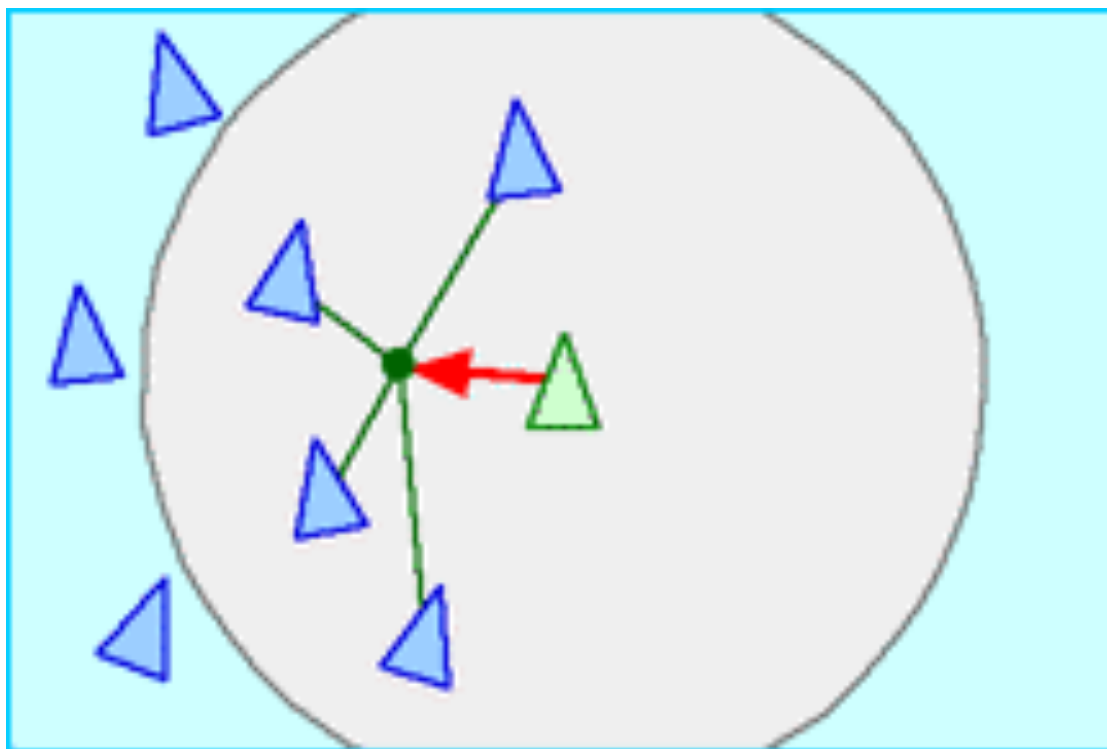
2 – Alignement : s'aligner avec les autres

- ◆ **Align:** Essayer de s'adapter à la vitesse et direction des voisins



3 - Centrage

- ◆ **Centrage (cohere):** essayer d'être le plus près des voisins, d'être plus au centre du troupeau..



Composer et arbitrer ces comportements

◆ Technique simple (implémentée dans NetLogo):

Soit E l'ensemble des voisins

Soit x le plus proche de self parmi E

Si $\text{dist}(\text{self}, x) < \text{dist-min}$

s'éloigner de x

sinon

s'aligner avec E

centrer vers E

aller vers E

Centrer vers (E)

$\text{dir} = \text{Somme}(\text{direction}(x))$ pour tout x de E

$\text{tourner-vers}(\text{dir}) //$ au plus de k degré

Formation en V: le vol des oies sauvages

Nathan, A. & Barbosa, V. C (2008)

- ◆ Règle 1: si agent est trop loin des autres agents, il accélère pour se rapprocher du plus proche.
- ◆ Règle 2: si un agent est suffisamment près d'un autre, il va venir sur l'un de ses côtés, pour que sa vue ne soit pas obstruée
- ◆ Règle 3: Si un agent est trop proche d'un autre, il ralentit
- ◆ Règle 4: quand les trois autres conditions sont remplies, l'agent adapte sa vitesse et direction à ses voisins visibles

NetLogo models library: Flocking Vee Formations

Flocking en V

```
to adjust ;; ajuster la direction et position par rapport aux autres
  set closest-neighbor min-one-of visible-neighbors [distance myself]
  let closest-distance distance closest-neighbor
  ;; if I am too far away from the nearest bird I can see, then try to get near them
  if closest-distance > updraft-distance [
    turn-towards (towards closest-neighbor)
    set speed base-speed * (1 + speed-change-factor)
    set happy? false
    stop
  ]

  ;; if my view is obstructed, move sideways randomly
  if any? visible-neighbors in-cone vision-distance obstruction-cone [
    turn-at-most (random-float (max-turn * 2) - max-turn)
    set speed base-speed * (1 + speed-change-factor)
    set happy? false
    stop
  ]

  ;; if i am too close to the nearest bird slow down
  if closest-distance < too-close [
    set happy? false
    set speed base-speed * (1 - speed-change-factor)
    stop
  ]

  ;; if all three conditions are filled, adjust
  ;; to the speed and heading of my neighbor and take it easy
  set speed [speed] of closest-neighbor
  turn-towards [heading] of closest-neighbor
  set happy? true
end
```


Approche vectorielle

◆ Idée :

- Composer l'ensemble des motivations et décisions d'actions sous la forme de vecteurs
- Utiliser de simples compositions vectorielles linéaires (la plupart du temps) ou non-linéaires pour déterminer le mouvement

◆ Comportement B est déterminé par la somme des comportements élémentaires B_i

$$\vec{B} = \alpha_1 \vec{B}_1 + \dots + \alpha_n \vec{B}_n$$

$$\vec{B} = \sum_{i=1}^n \alpha_i \vec{B}_i$$

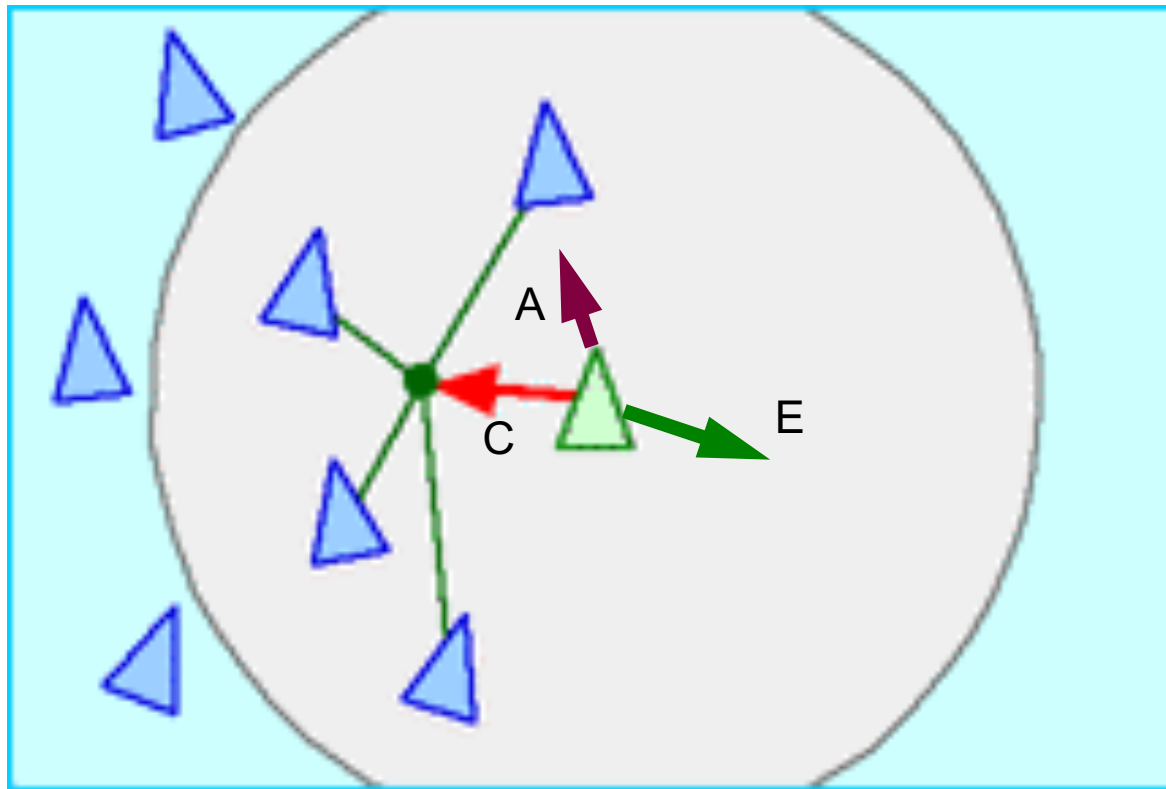
Application au flocking

- ◆ Chaque comportement (évitement, alignement, centrage) définit un vecteur
 - On somme ces vecteurs en fonction de leur importance:

$$\vec{D} = a \vec{A} + e \vec{E} + c \vec{C}$$

- A = alignement, E = Répulsion, C = centrage

Flocking vectoriel



A = alignement,
E = Répulsion,
C = centrage
D = direction
résultante

$$\vec{D} = a \vec{A} + e \vec{E} + c \vec{C}$$

Fonctions vectorielles en NetLogo

```
to-report angleFromVect [vect]
  let a atan item 0 vect item 1 vect
  report a
end
```

```
to-report vectFromAngle [angle len]
  let l (list (len * cos angle) (len * sin angle))
  report l
end
```

```
to-report multiplyScalarvect [factor vect]
  report (list (item 0 vect * factor) (item 1 vect * factor))
end
```

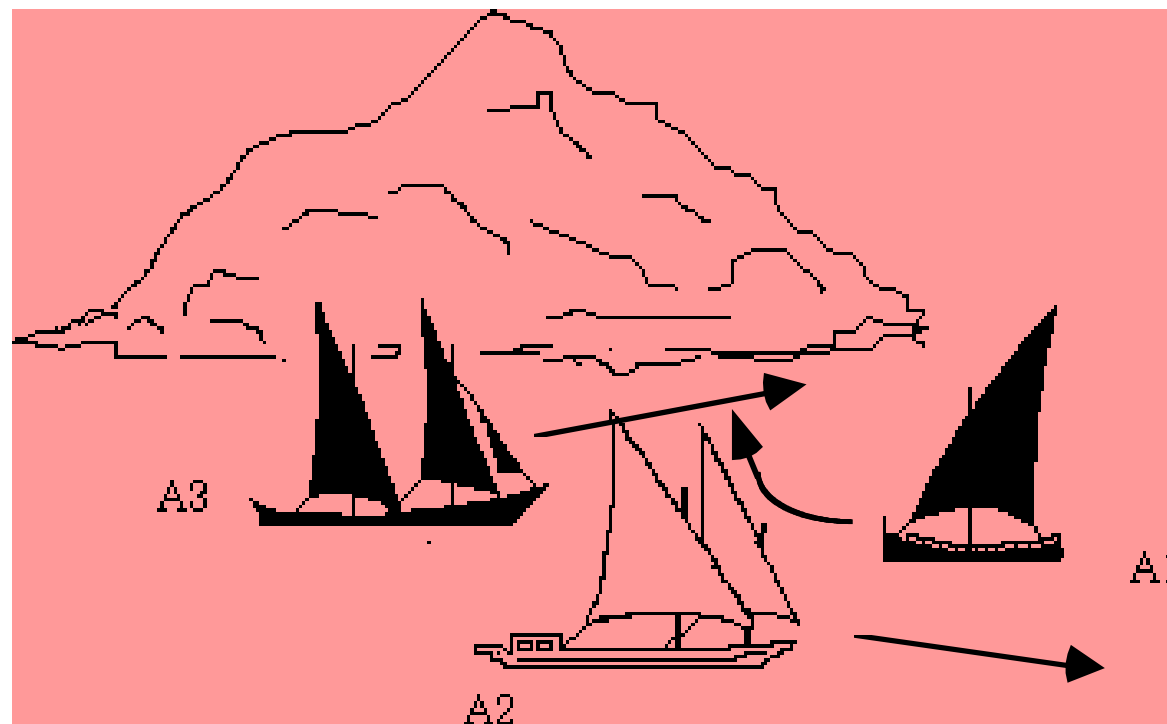
```
to-report additionvect [v1 v2]
  report (list (item 0 v1 + item 0 v2) (item 1 v1 + item 1 v2) )
end
```


Exemple de fonction en version vectorielle

```
to-report vectCohere
  let x-component mean [sin (towards myself + 180)] of flockmates
  let y-component mean [cos (towards myself + 180)] of flockmates
  report (list x-component y-component)
end
```

```
to-report vectAlign
  let x-component sum [dx] of flockmates
  let y-component sum [dy] of flockmates
  report (list x-component y-component)
end
```

Evitement de collision



Evitement d'obstacles statiques

- ◆ 1^{ère} solution = répulsion

- ◆ Deux approches

- Priorité: d'abord flocking, ensuite répulsion, ou l'inverse

- Combinaison de vecteurs

$$\mathbf{D} = a\mathbf{R} + (1 - a)\mathbf{F}$$

Où \mathbf{R} est le vecteur de répulsion et \mathbf{F} celui du flocking.

- a est le coefficient de contrôle. Peut varier en fonction inverse de la distance à l'obstacle (quand l'obstacle est droit devant) $a = k/\text{dist}(\text{self}, \text{obstacle})$

Retour sur l'évitement d'obstacles et le flocking

◆ Algorithme général

```
to flock ;; turtle procedure
  find-flockmates
  if any? flockmates
    [ find-nearest-neighbor
      ifelse distance nearest-neighbor < minimum-separation
        [ separate ]
        [ align
          cohere ] ]
  avoid-obstacles ;; ce qui change..
end
```


◆ Fuir: partir dans le sens opposé

```
to avoid-obstacles
; avoid anything nearby that is not black
  set obstacles patches in-cone vision angle-avoidance
    with [pcolor != black]
  if (any? obstacles)
    [flee]
end

to flee
  let obstacles-in-front obstacles in-cone 3 angle-flee
  if (any? obstacles-in-front)
    [
      rt 180
      rt random 10
      lt random 10
    ]
end
```

Smart avoidance

- ◆ Mais si pas possible, on fait au mieux (et il y a des crashes)

```
to smart-avoidance
  turn-at-most 180 max-avoidance-turn
  flee
end
```

Répulsion

L'agent hésite entre
répulsion et aller vers
l'objectif

but de l'agent 1



Objectif



Obstacle



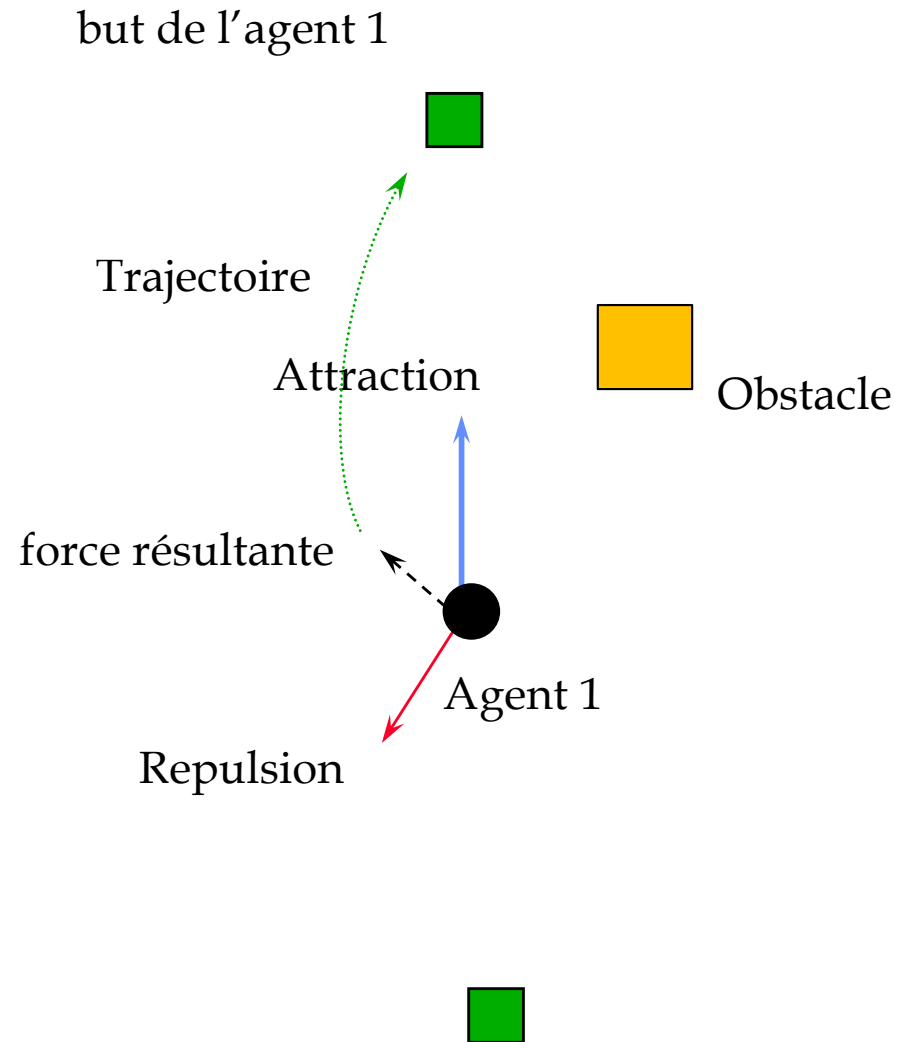
Agent 1

Repulsion



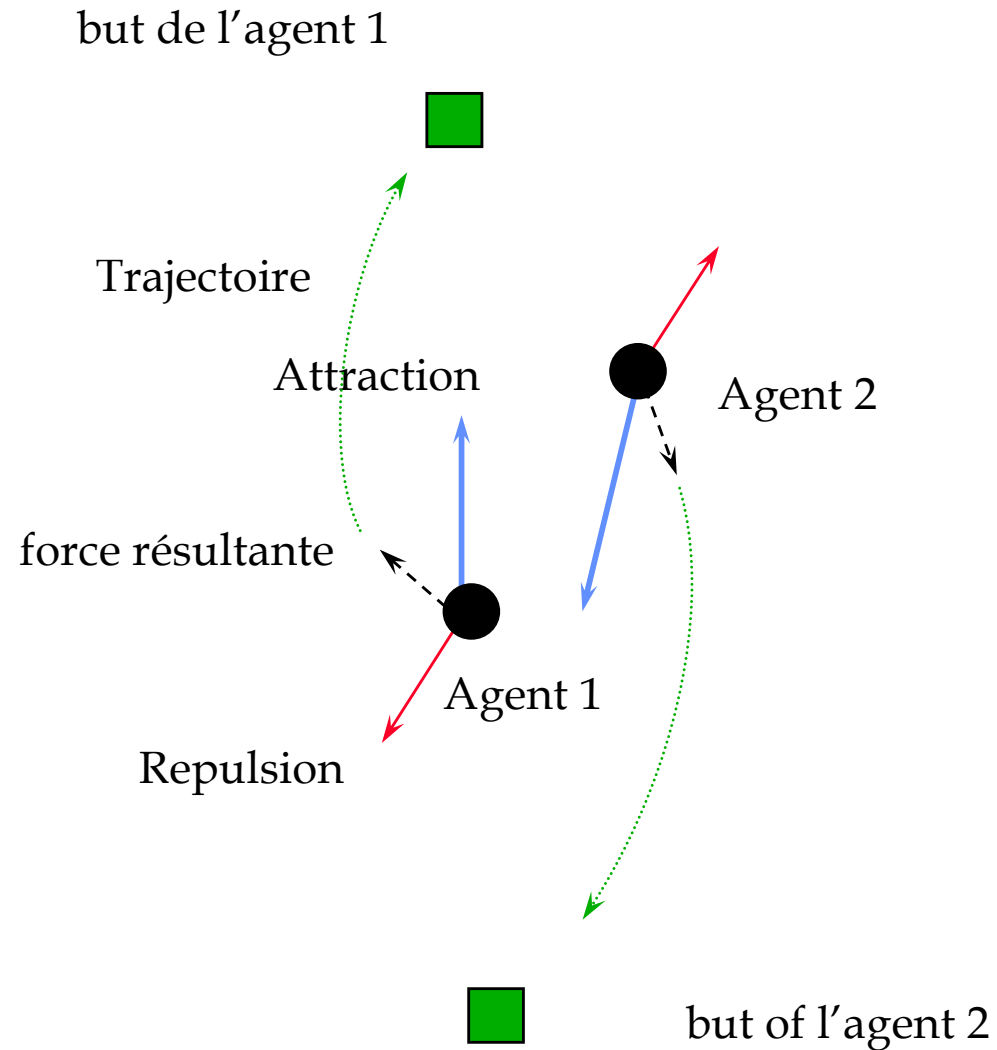
Combinaison vectorielle

L'agent combine deux vecteurs, chacun correspondant à un comportement

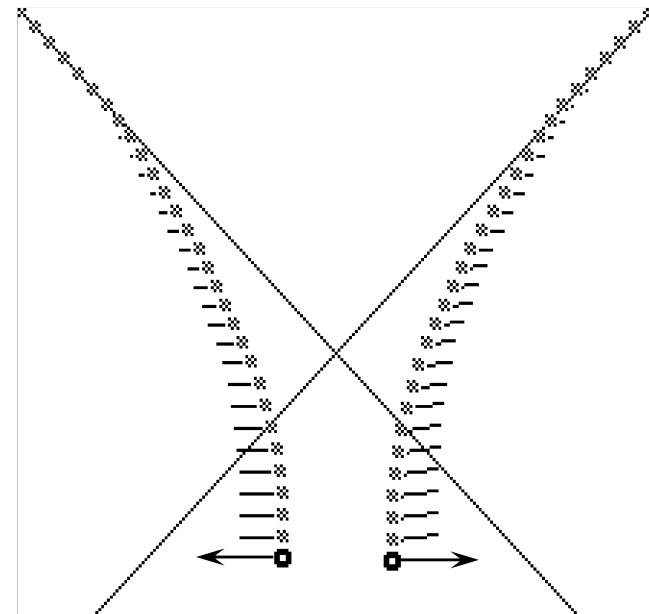
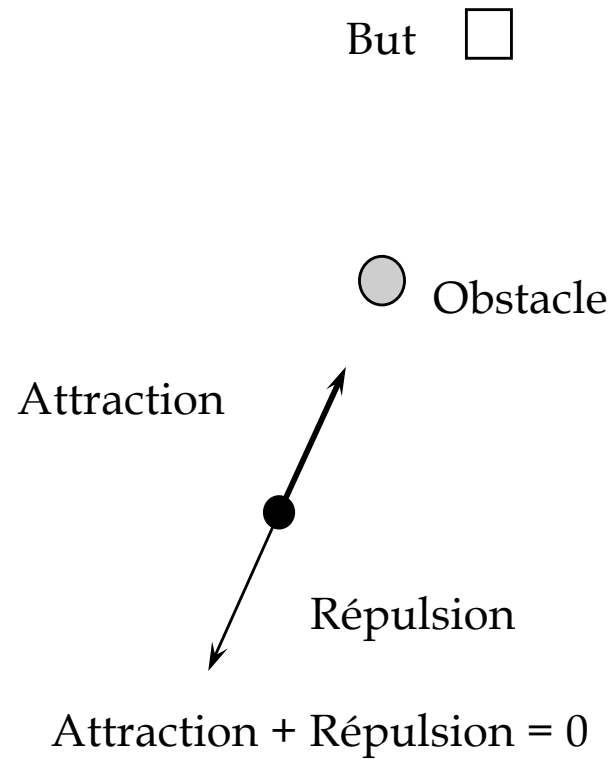


Approche par champ de force

Chaque agent est considéré comme étant un obstacle pour l'autre



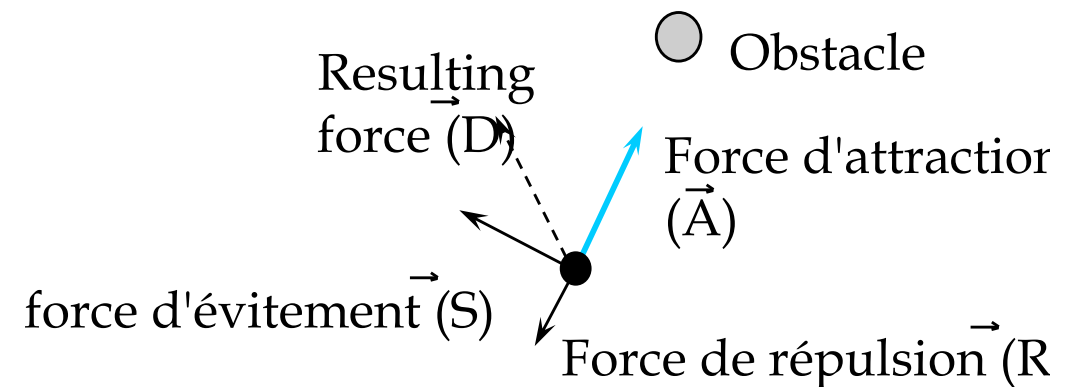
Mais la répulsion n'est pas l'évitement



Solution: utiliser des forces d'évitement

Eviter signifie rester à "bonne" distance des obstacles en se dirigeant vers le but

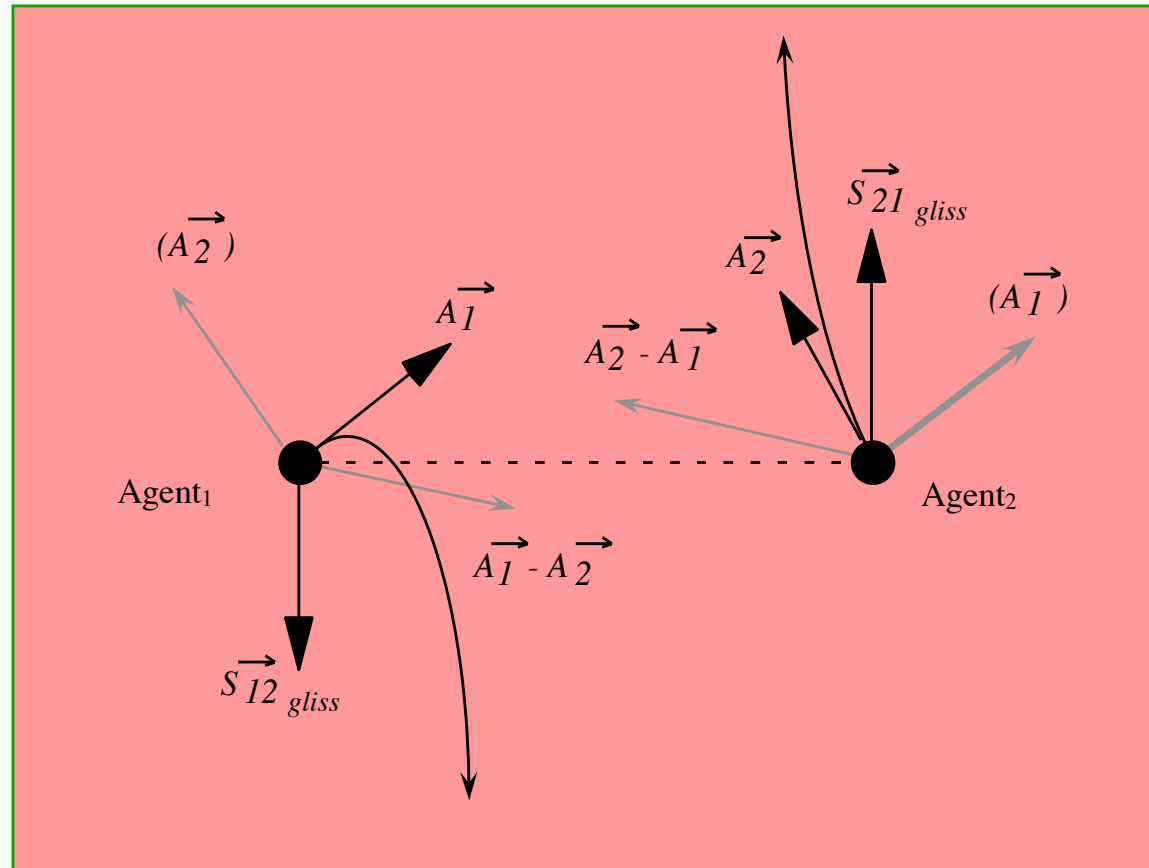
But



$S(p)$ est tel que $dir(\vec{R}(p)).dir(\vec{S}(p)) = 0$

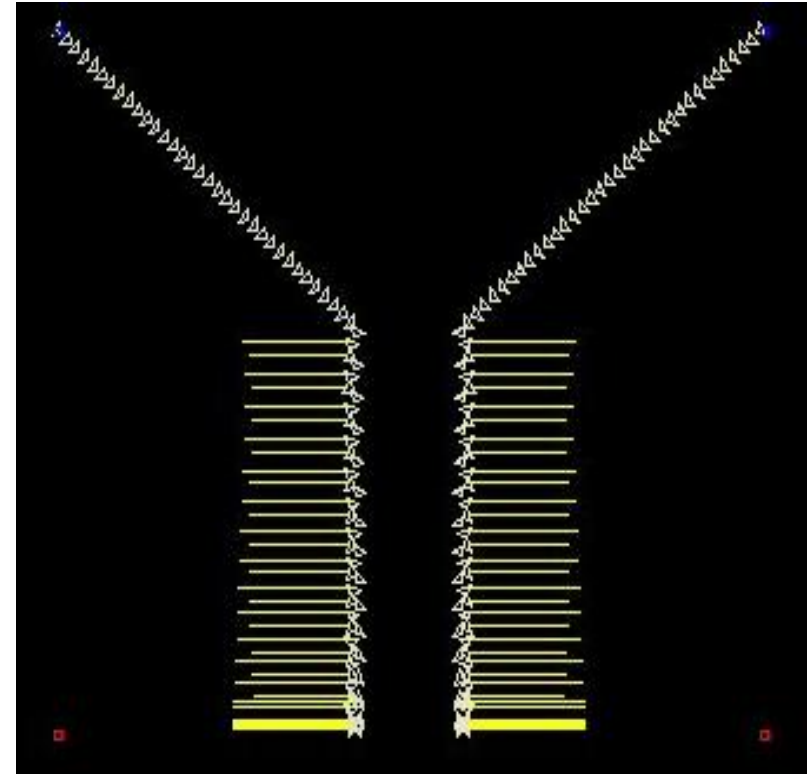
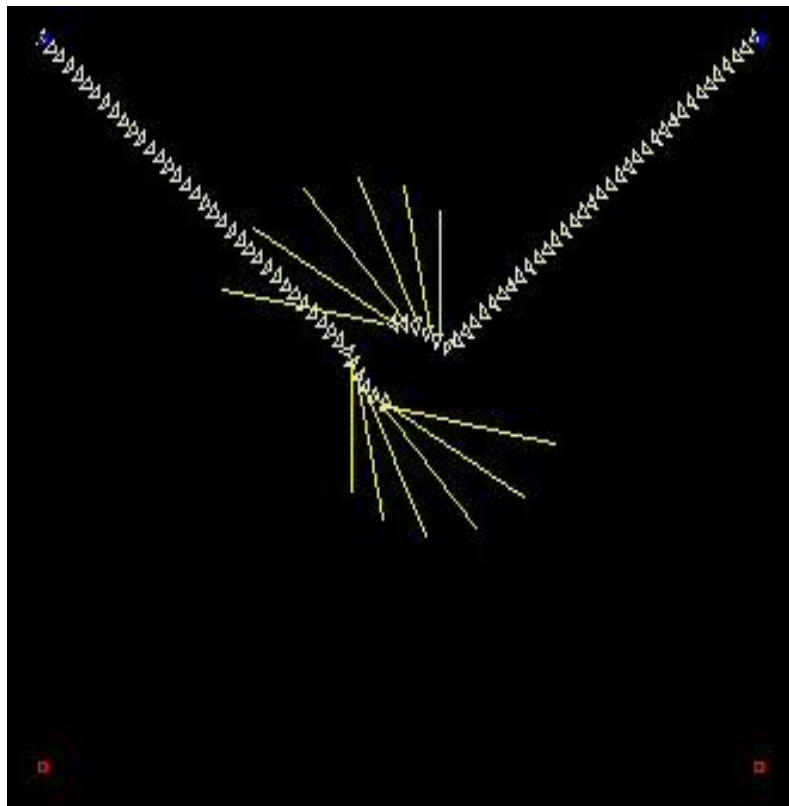
$$\vec{D}(p) = \alpha \vec{A}(p) + \beta \vec{R}(p) + \gamma \vec{S}(p)$$

Forces d'évitement symétrique



K. Zeghal, J. Ferber 1992

Exemples



*D'après un projet réalisé à
l'UTBM sous la dir. d'O. Simonin*

Emergence de structures dynamiques

