

Algorithmes d'exploration et de mouvements

GMIN20A

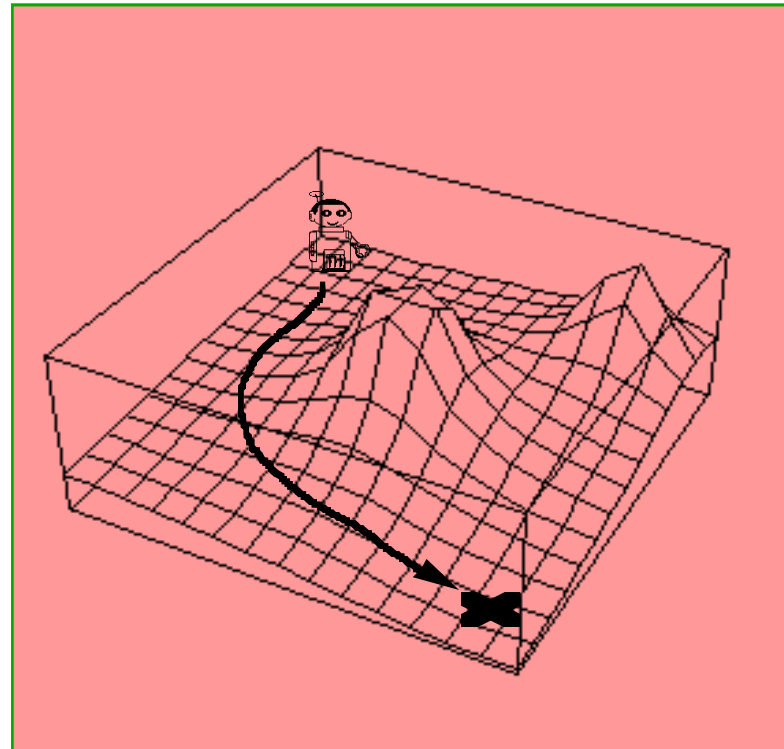
Champs de potentiel et évitement de collision #2

V2.2 – Mars 2017

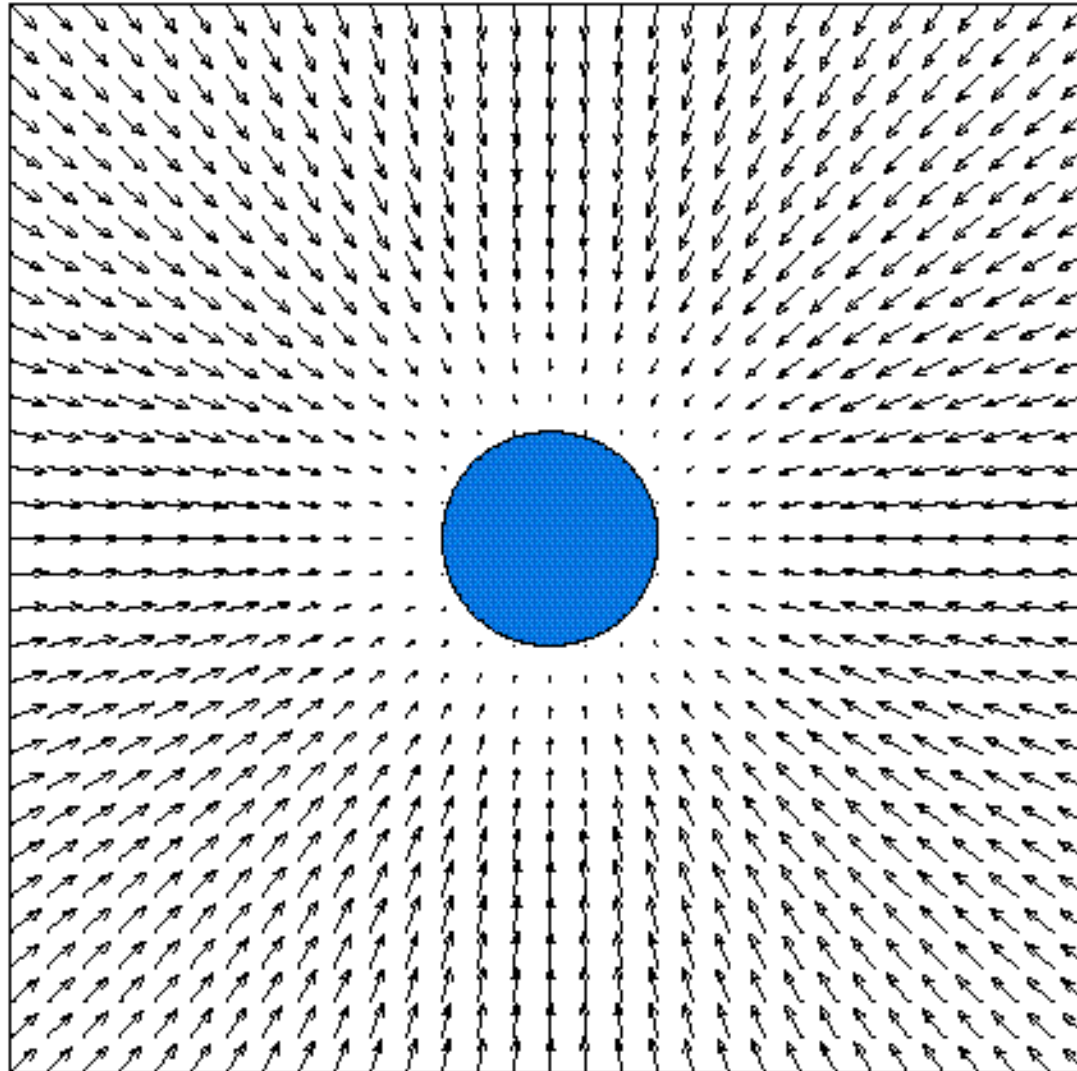
Jacques Ferber

Champs de potentiels et mouvements

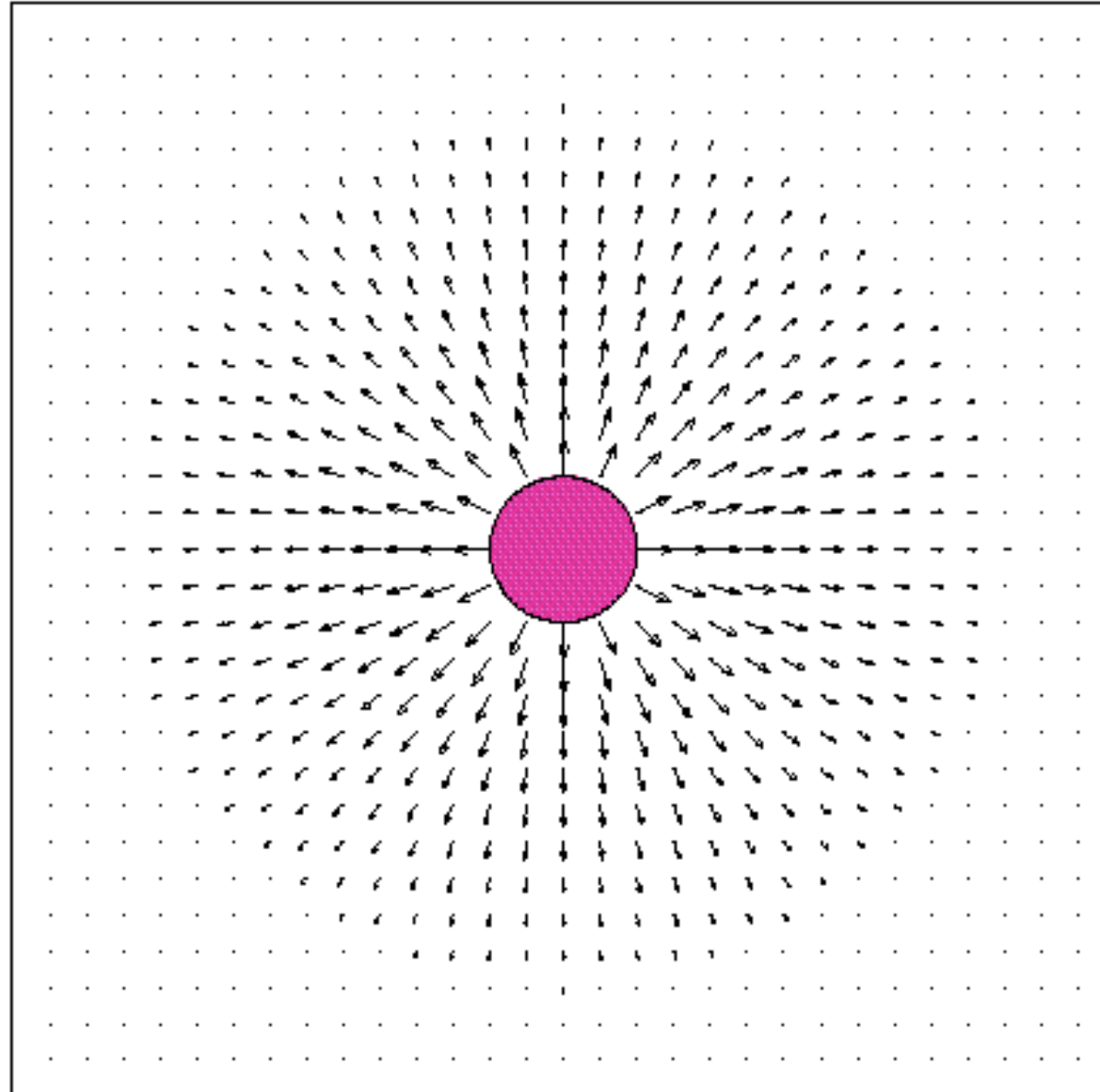
*L'agent se déplace dans un champ de forces
La position à atteindre est un attracteur, et les obstacles sont des
éléments répulsifs*



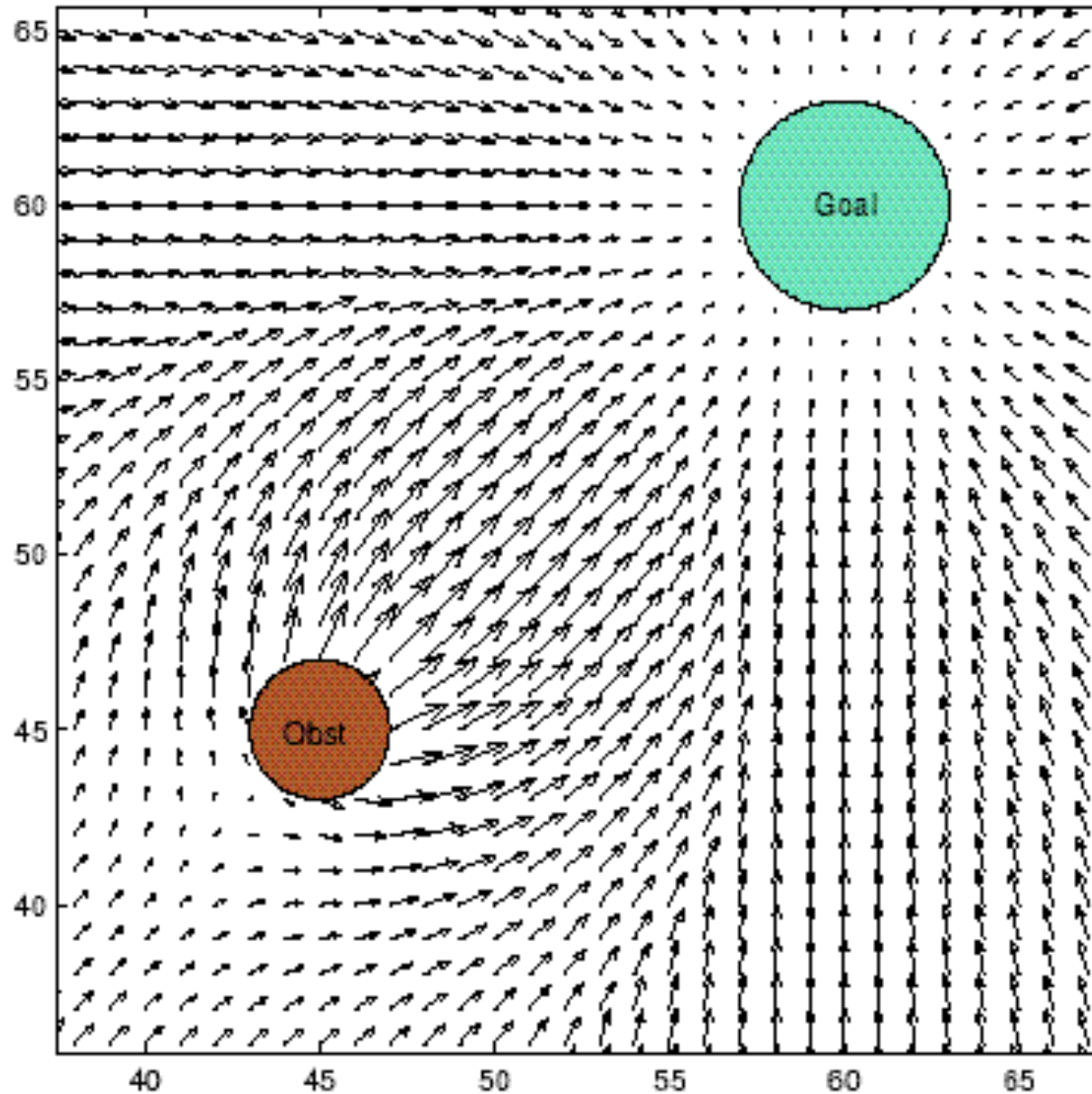
Champ de force attractif



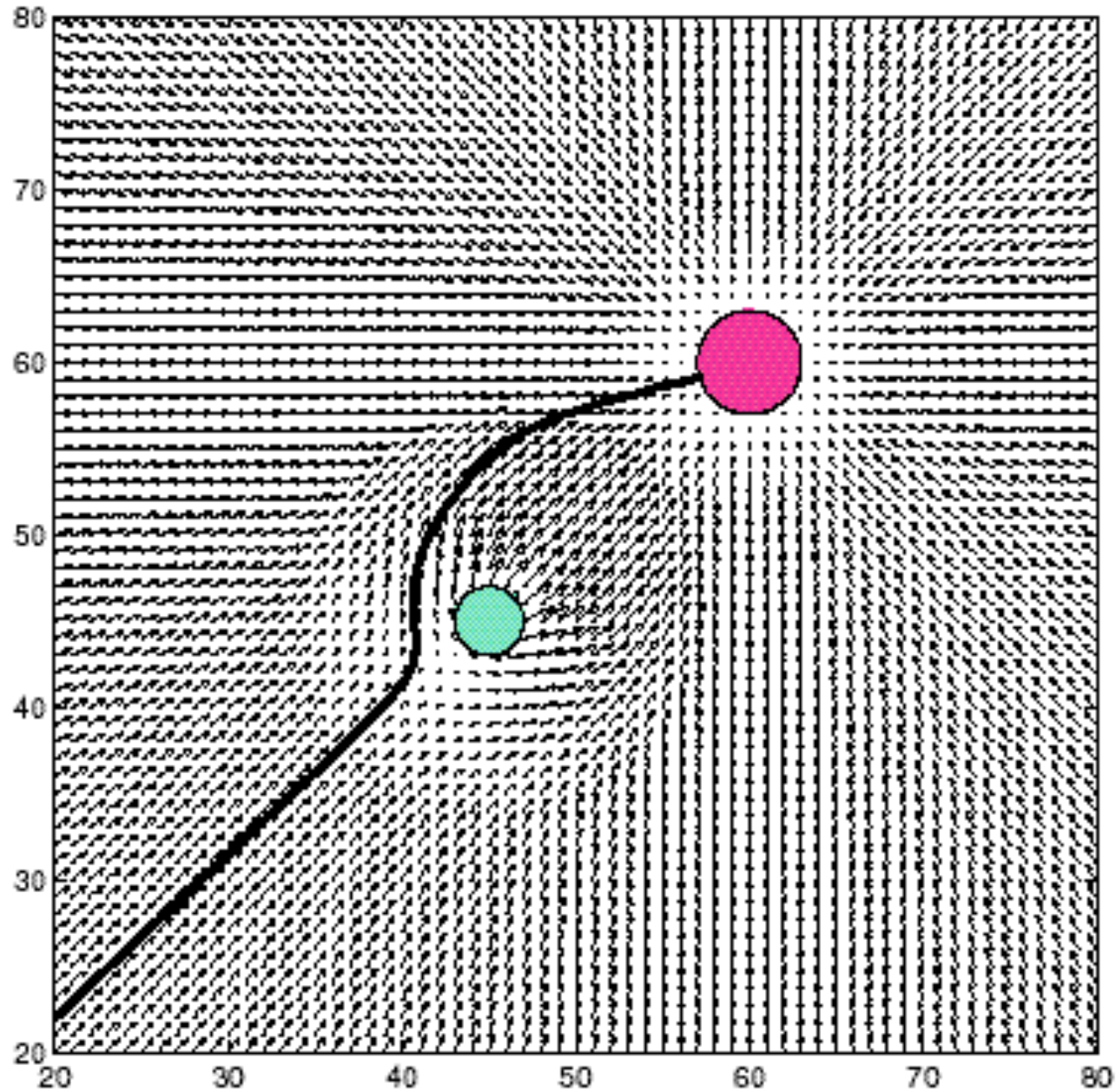
Champ de force répulsif



Somme vectorielle des deux champs



Trajectoire résultante de l'agent



Champs de potentiel

- ◆ Un champ de potentiel est une fonction qui associe à tout points (x,y) un nombre considéré comme la valeur du champ en ce point: $P(x,y)$
- ◆ Le gradient d'un champ de potentiel est un champ vectoriel définit :

$$(x, y) \rightarrow (\Delta x, \Delta y) = \nabla P(x, y)$$

$$(\Delta x, \Delta y) = \nabla P(x, y) = \left(\frac{\partial P}{\partial x}, \frac{\partial P}{\partial y} \right)$$

Construction du champ: attraction et répulsion

Les forces sont définies comme le gradient d'un champ de potentiel

$$F(x,y) = -\text{grad}(P(x,y))$$

Les buts sont représentés comme des champs attractifs.

Les obstacles sont représentés comme des champs répulsifs

Le mouvement est obtenu par une combinaison de champs attractifs et répulsifs

$$P(x,y) = U_{\text{attr}(x,y)} + U_{\text{repul}(x,y)}$$

- ◆ **Mouvement:** il suffit de « descendre » le champ en suivant le gradient, la ligne de plus grande pente

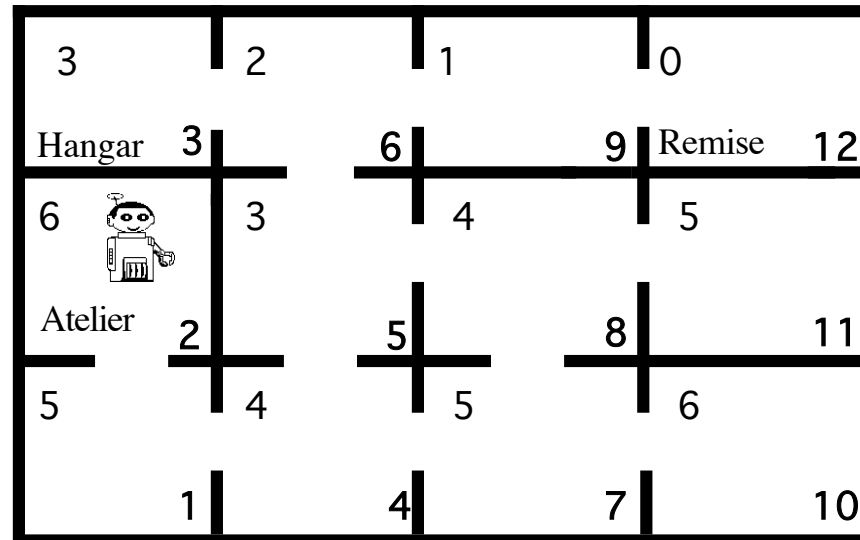
◆ *Minimum local*

- Les forces attractives et répulsives peuvent s'annuler
- Les formes convexes constituent des cul de sac

Comment créer un meilleur champ de potentiel?

◆ Algorithme d'inondation ou de vague

- Consiste à « inonder » un espace, en diffusant un champ de potentiel:



◆ Intérêt:

- Permet d'éviter les minima locaux

Algorithme

◆ Algorithme...

to inonde

si case-courante \neq but et case \neq d'obstacle

let v-old potential

let case1 max-one-of des voisins de case [potential]

let p ([potential] of case1) - 1

if (potential \leq p) and (p \neq old-pot) [

set potential p

ifelse potential $<$ 0

[set potential 0]

[set continue true]

]

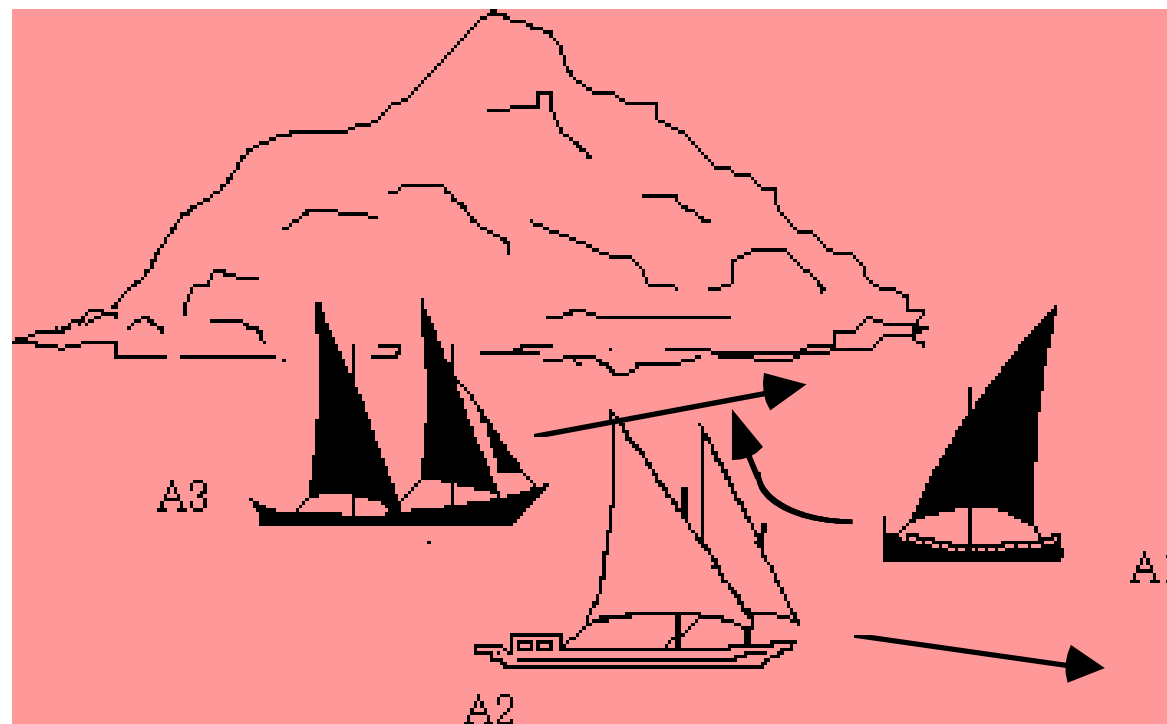
While continue [

set continue false

ask patches [inonde]

Démo champ de potentiel et algorithme d'inondation

Evitement de collision



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

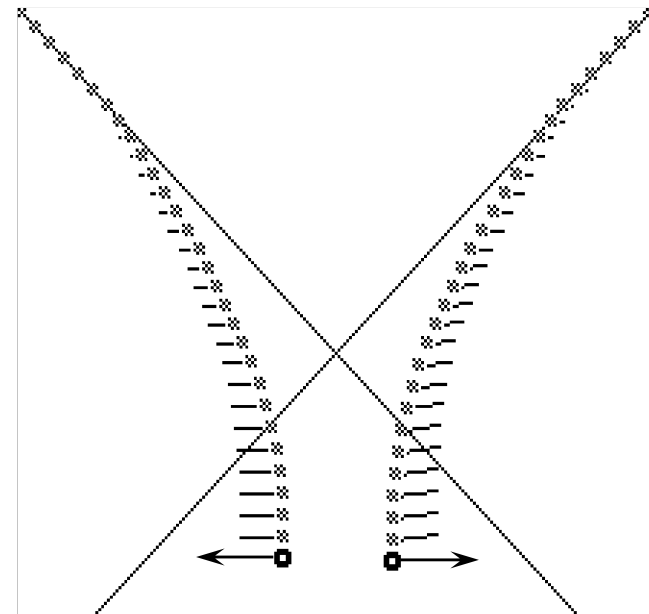
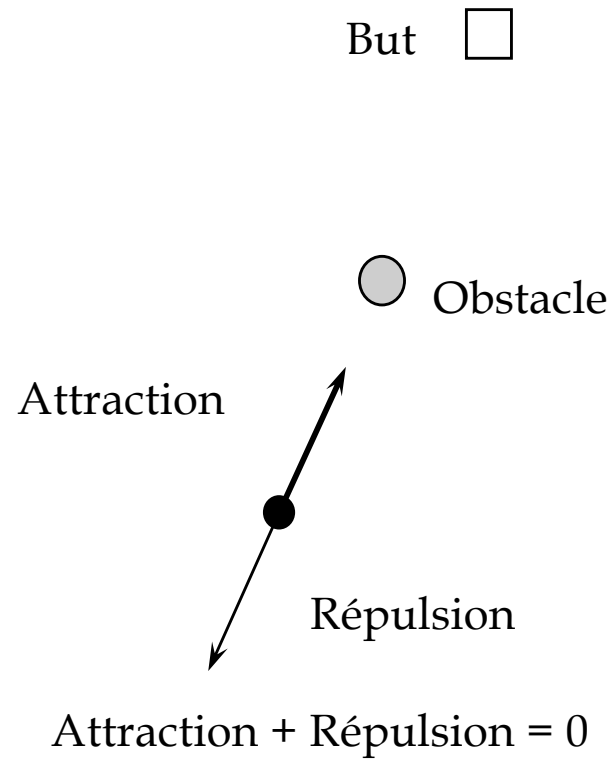
to flee
  let obstacles-in-front obstacles in-cone 3 angle-flee
  if (any? obstacles-in-front)
    [
      rt 180
      rt random 10
      lt random 10
    ]
end
```

Smart avoidance

◆ 2^{ème} solution: éviter...

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

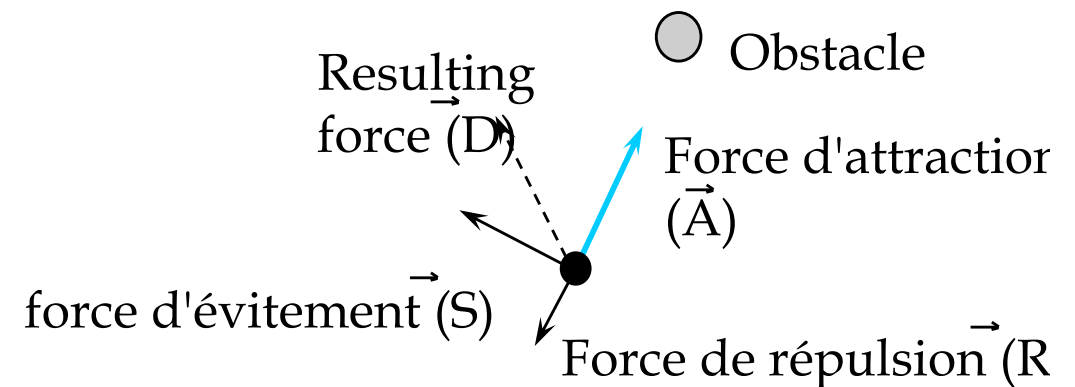

Mais la répulsion n'est pas l'évitement



Solution: utiliser des forces d'évitement

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

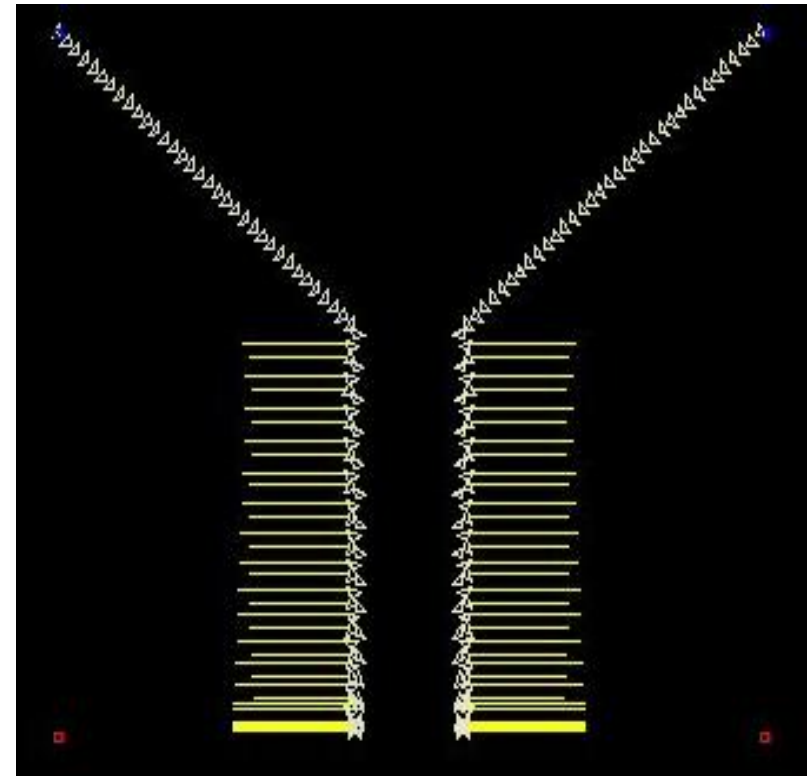
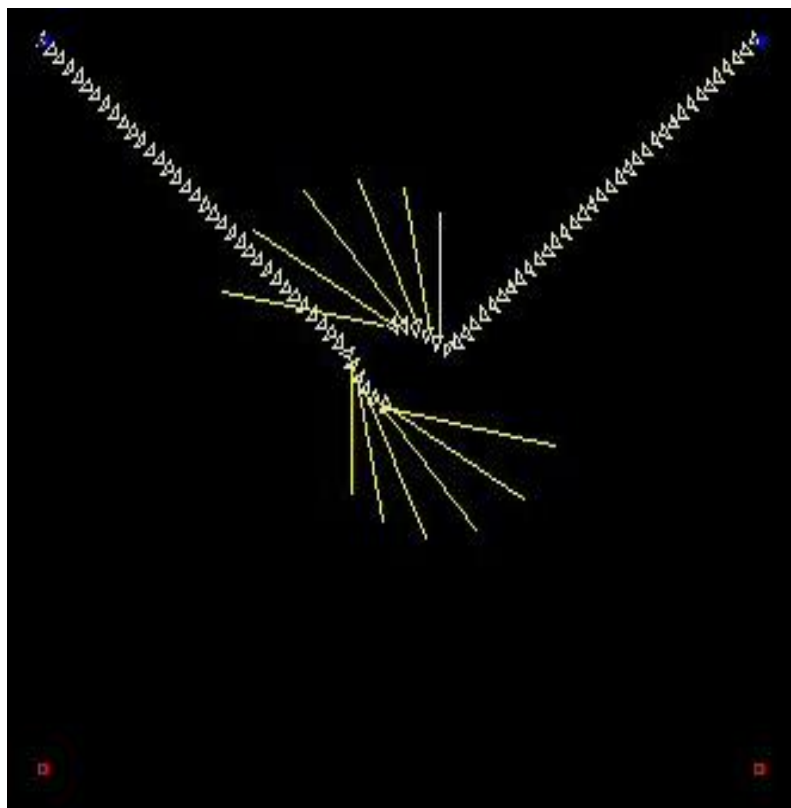
But



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

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

Exemples



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

Retour sur la définition vectorielle de comportements

◆ Définition de la notion de vecteur en NetLogo

```
to-report multVect [a vector]
  report (list (item 0 vector * a) (item 1 vector * a))
end
```

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

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

```
to-report getAngleFromVect[v]
  report atan item 1 v item 1 0
end
```

```
to-report getLengthFromVect[v]
  let x item 0 v
  let y item 1 v
  report sqrt (x * x) + (y * y)
end
```

Combiner plusieurs comportements

◆ Flocking + évitement d'obstacles

- Composer l'approche vectorielle avec le vecteur d'évitement de collision
- Combinaison de vecteurs
 - $D = aR + (1 - a) F$
 - Où R est le vecteur de répulsion et 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) $a = k/\text{dist}(\text{self}, \text{obstacle})$

Opérations vectorielles

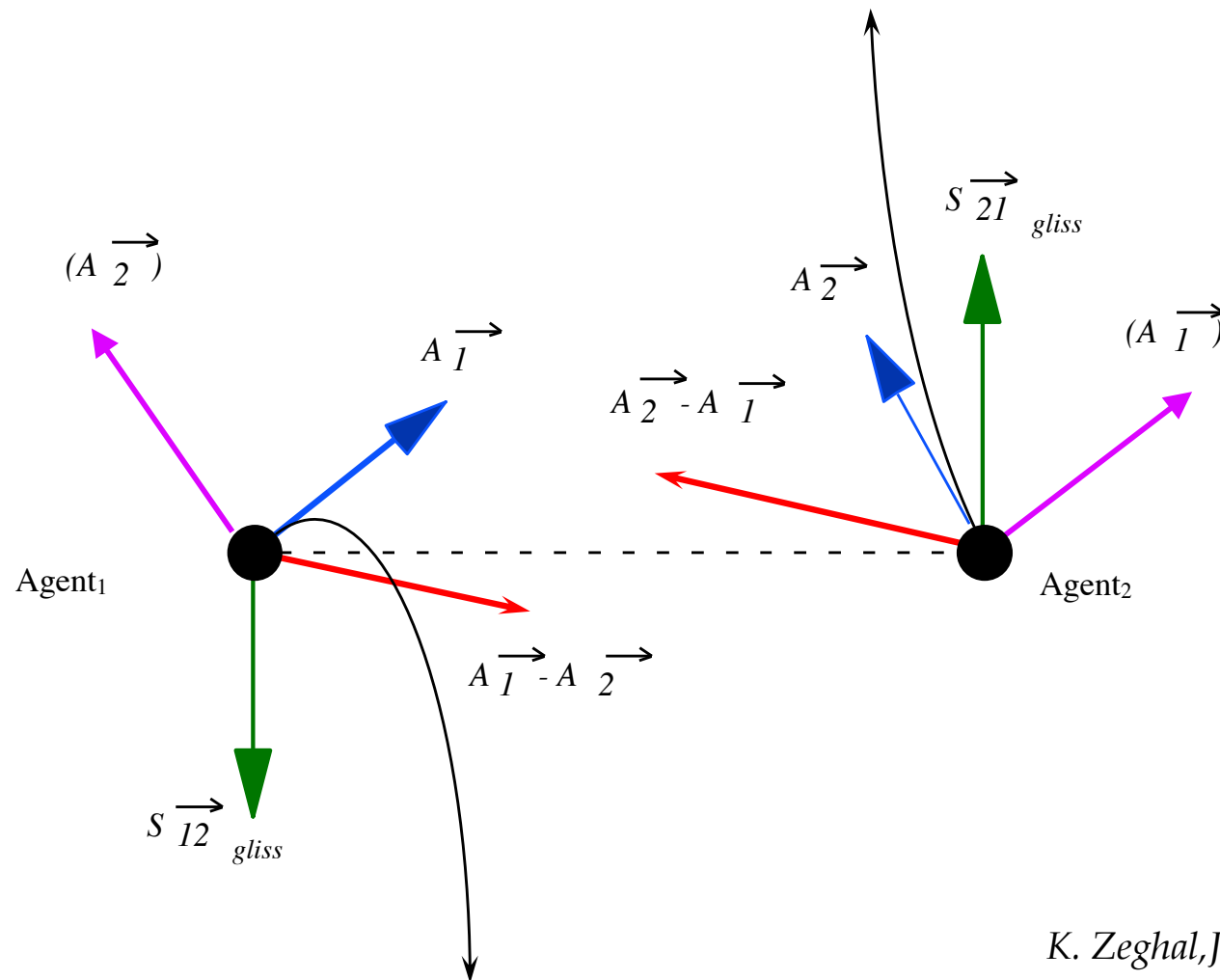
◆ $V4 = aV1 + bV2 + cV3$ où

```
let v1 vectorFromPolar angl len1  
let v2 vectorFromPolar ang2 len2  
let v3 vectorFromPolar ang3 len3
```

```
let v4 addVect (multVect a v1) addVect (multVect b v2)(multVect c v3)
```

```
let ang4 getAngleFromVect v4  
let len4 getLengthFromVect v4
```

Forces d'évitement symétrique



Calculer un vecteur perpendiculaire à un autre

- ◆ **Vc est perpend à V si $V \cdot Vc = 0$**

$$x \cdot x_c + y \cdot y_c = 0$$

et le vecteur est aussi de longueur 1:

$$x_c^2 + y_c^2 = 1$$

- ◆ **Calcul**

- **Solution générale**

$$x_c = \frac{-y}{\sqrt{x^2 + y^2}}$$

$$y_c = \frac{x}{\sqrt{x^2 + y^2}}$$

- **2 solutions:**

☞ $x_c = -y$ et $y_c = x$

☞ $x_c = y$ et $y_c = -x$

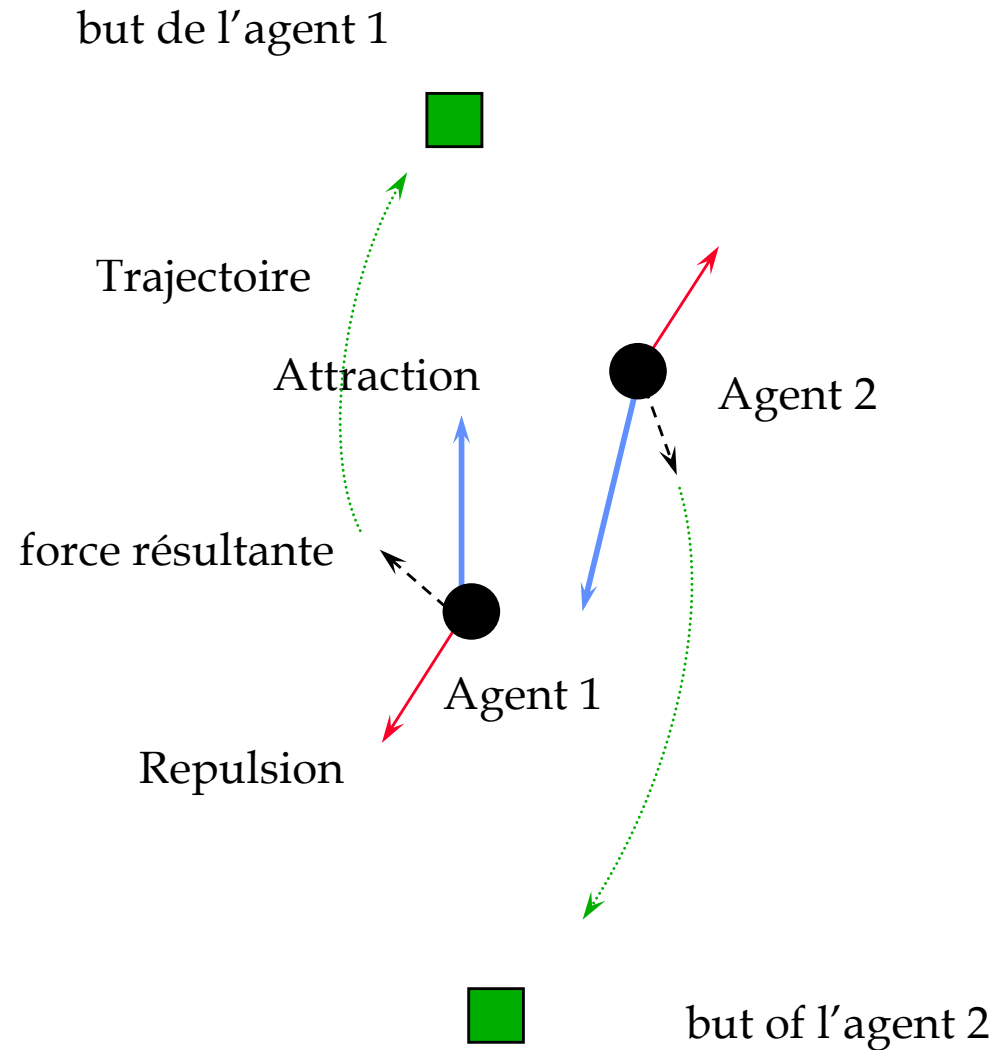
Si v est normalisé

```
to-report orthoVect1 [ang]
  let v vectorFromPolar ang
  let xc (- item 1 v)
  let yc item 0 v
  report (list xc yc)
end
```

```
to-report orthoVect2 [ang]
  let v vectorFromPolar ang
  let xc item 1 v
  let yc (- item 0 v)
  report (list xc yc)
end
```


Approche par champ de force

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



Soustraction vectorielle

◆ $V1 - V2 = V1 + (-V2)$

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

Emergence de structures dynamiques

