# HMIN233 - Algorithmes d'exploration et de mouvement

## Évitement et champs de potentiels

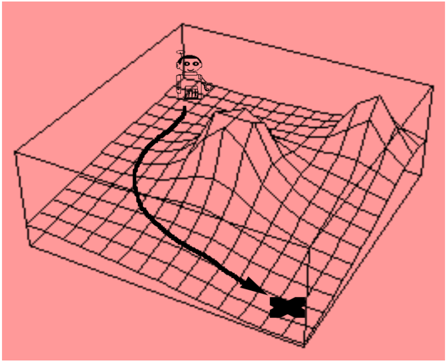
Suro François
(adaptation des cours de Jacques Ferber)

Université de Montpellier
Laboratoire d'informatique, de robotique
et de microélectronique de Montpellier

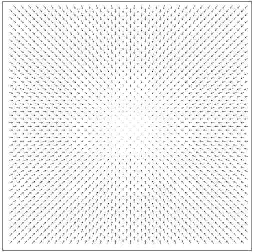
Janvier 2021

# Champs de potentiels et mouvements

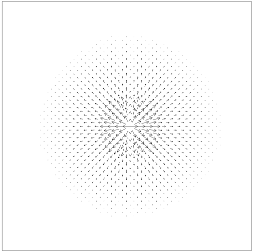
L'agent se déplace dans un champ de forces. La position à atteindre est un attracteur, et les obstacles sont des éléments répulsifs.



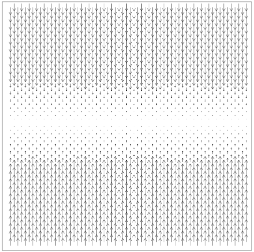
# Architecture AuRA



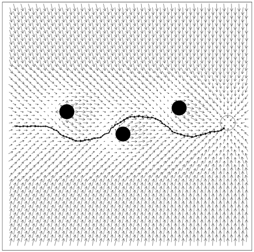
(A) Move-to-goal



(B) Avoid-static-obstacle

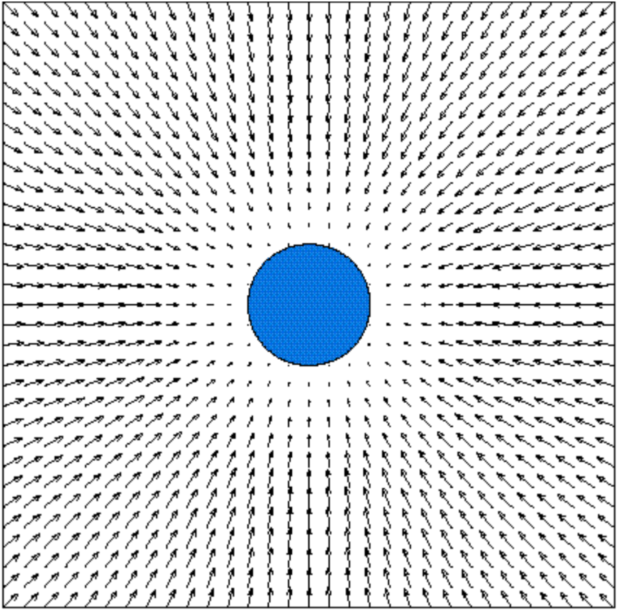


(C) Stay-on-path

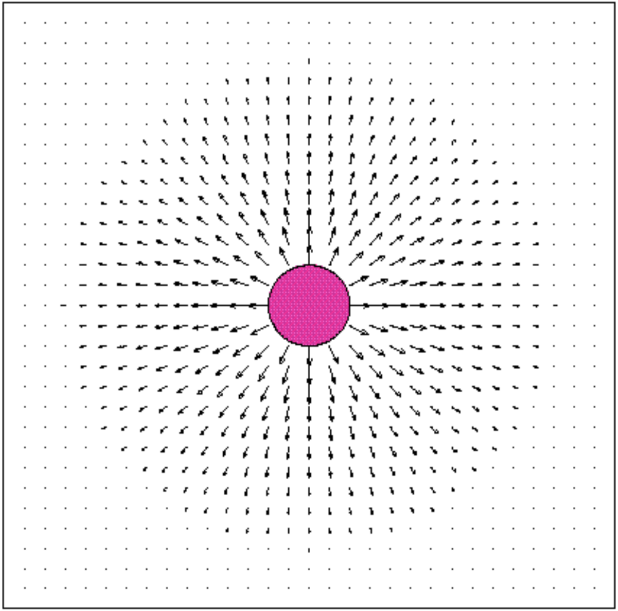


(A)+(B)+(C)

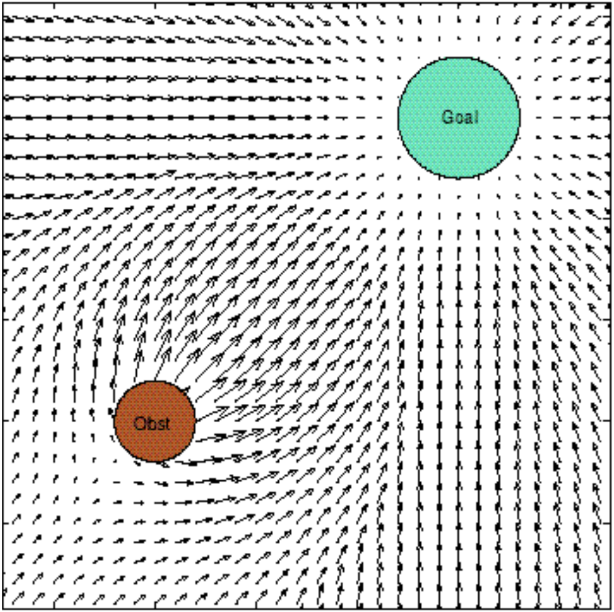
# Champ de force attractif



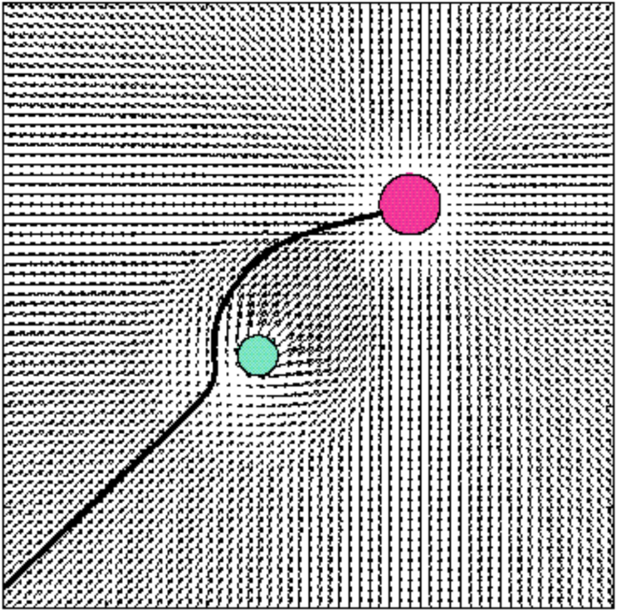
# Champ de force répulsif



# 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 :

$$(x, y) \mapsto P(x, y)$$

$$\nabla P(x, y) : (\delta x, \delta y) \mapsto \left( \frac{\delta P}{\delta x}, \frac{\delta P}{\delta y} \right)$$

# Construction du champ: attraction et répulsion

## Suivre un gradient de potentiel

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

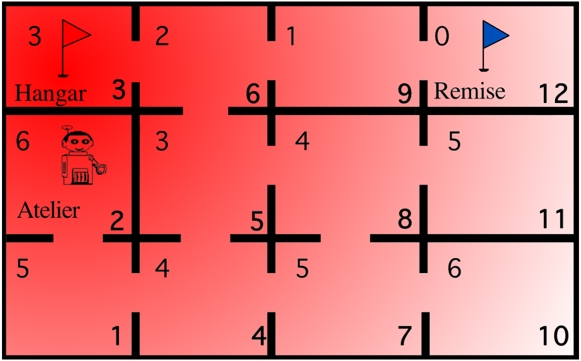
$$F(p) = grad(U(p))$$

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:

$$U(p) = Uattr(p) + Urepul(p)$$

**Mouvement**: il suffit de "descendre" le champ en suivant le gradient, la ligne de plus grande pente.

# Problèmes avec les champs de potentiels



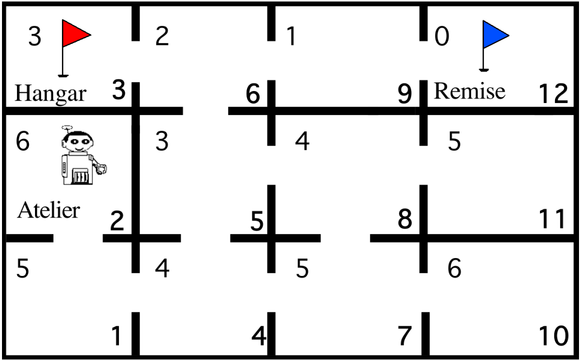

## Minimum local

- 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

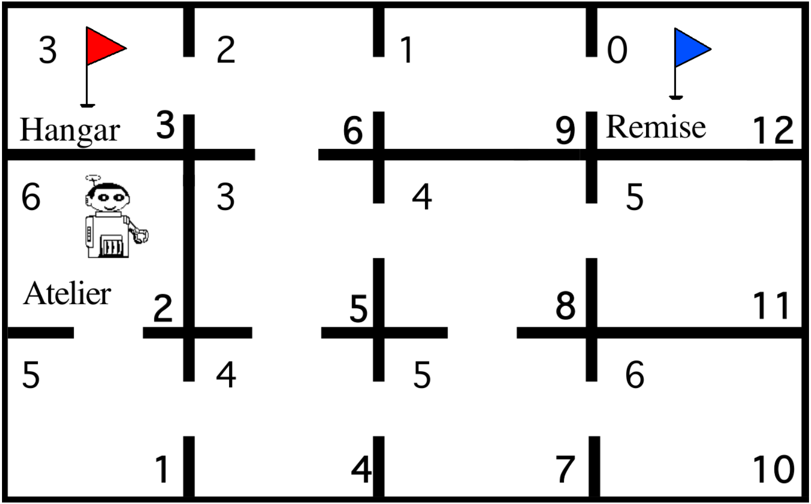
Consiste à "inonder" un espace, en diffusant un champ de potentiel:



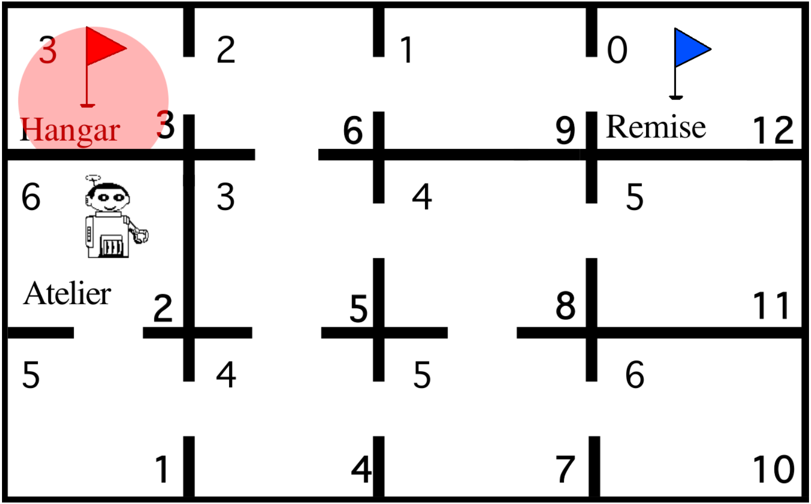
# Algorithme d'inondation

```
to inonde-case
   if type != but and type != obstacle[
      let case-max max-one-of neighbors [potential]
      let new-potential ([potential] of case-max) - 1
      if (potential < new-potential)[
         set potential new-potential
         ifelse potential < 0
         [set potential 0]
         [set continue true]
      ]
   ]
end

to inonder
   While continue [
      set continue false
      ask patches [inonde-case]
   ]
end
```

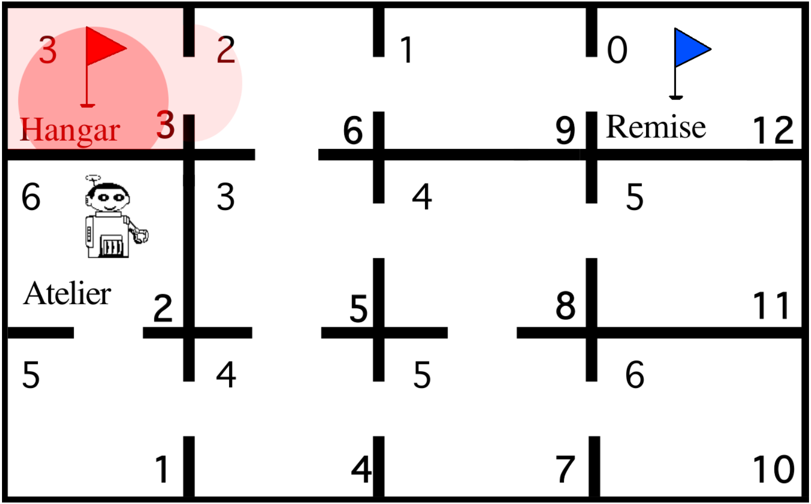
# Démo champ de potentiel et algorithme d'inondation



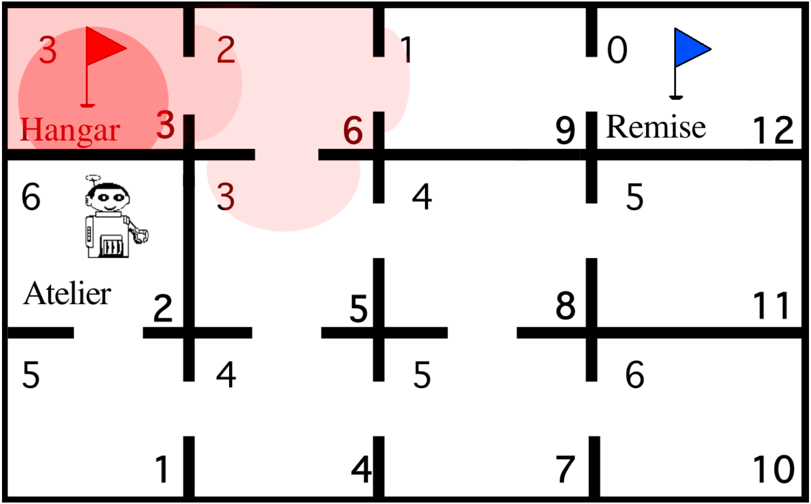
# Démo champ de potentiel et algorithme d'inondation



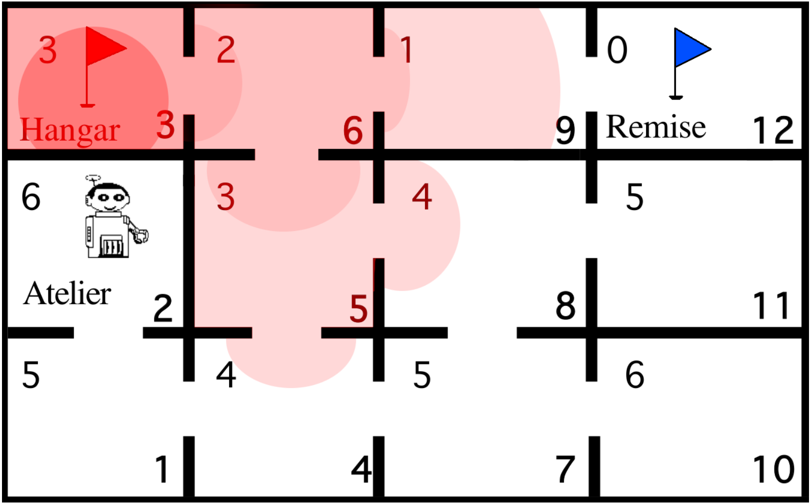
# Démo champ de potentiel et algorithme d'inondation



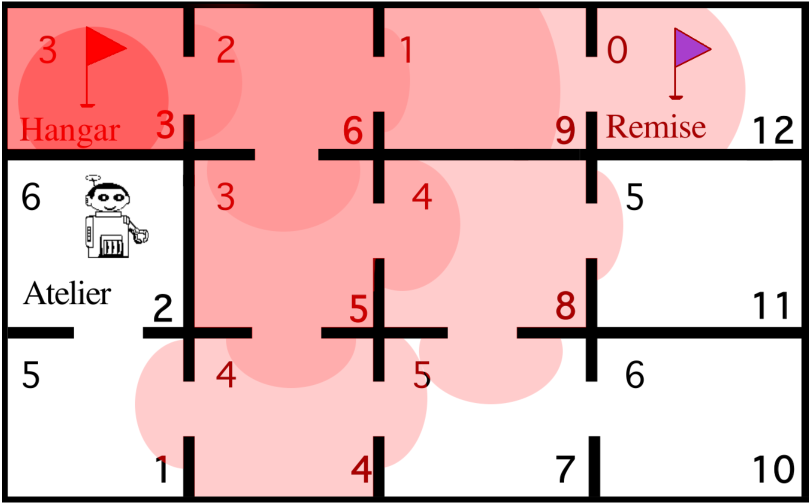
# Démo champ de potentiel et algorithme d'inondation



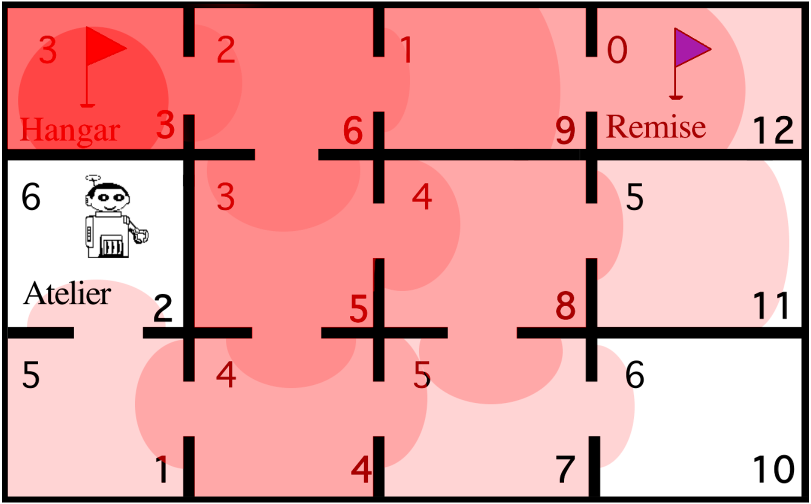
# Démo champ de potentiel et algorithme d'inondation



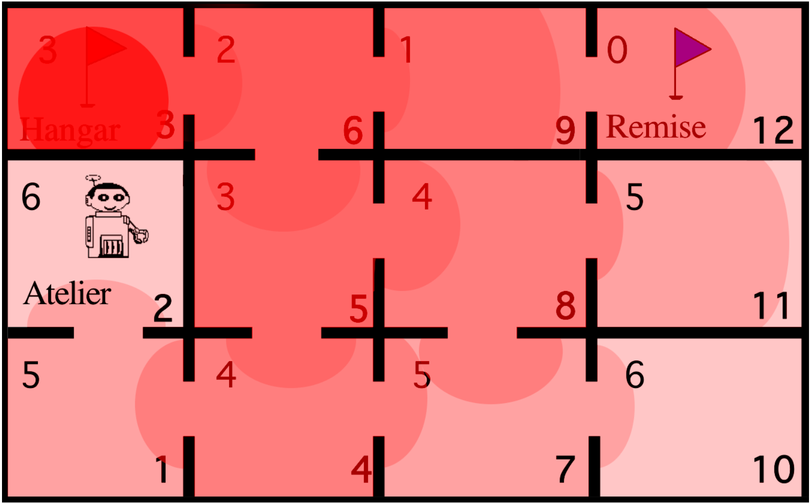
# Démo champ de potentiel et algorithme d'inondation



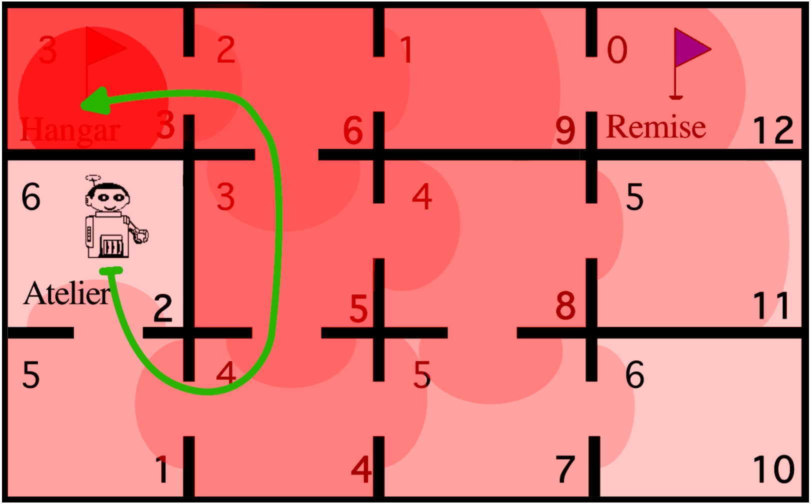
# Démo champ de potentiel et algorithme d'inondation



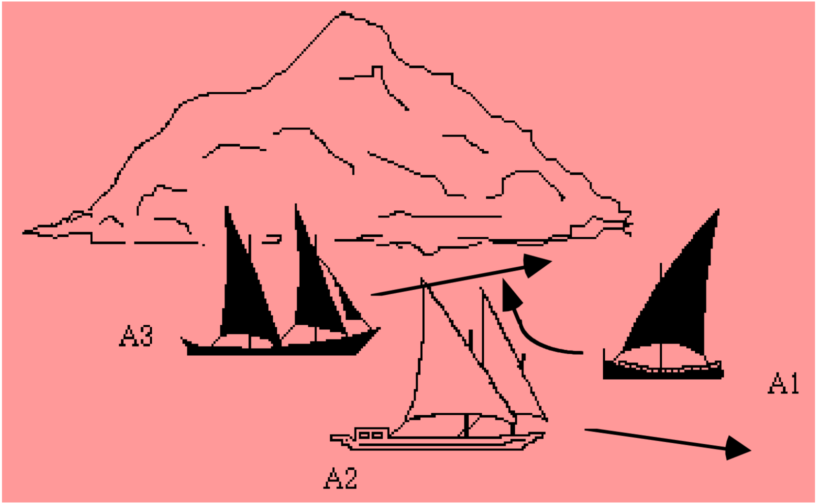
# Démo champ de potentiel et algorithme d’inondation



# Démo champ de potentiel et algorithme d'inondation



# Évitement de collision



# Retour sur l'évitement d'obstacles et le flocking

## Flocking + évitement d'obstacles

Composer l'approche vectorielle avec le vecteur de répulsion.

## Combinaison de vecteurs

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

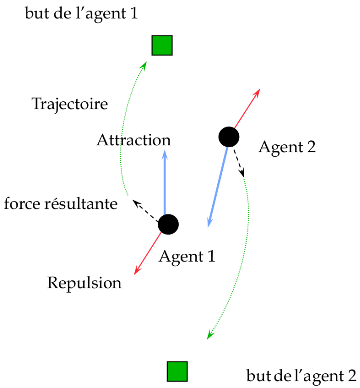
Où $\vec{R}$ est le vecteur de répulsion et $\vec{F}$ celui du flocking. **a** est le coefficient de contrôle. Peut varier en fonction inverse de la distance à l'obstacle (quand l'obstacle est droit devant) $a = k/dist(self, obstacle)$.

# 1 ère solution: fuite/répulsion

## Fuir: partir dans le sens opposé

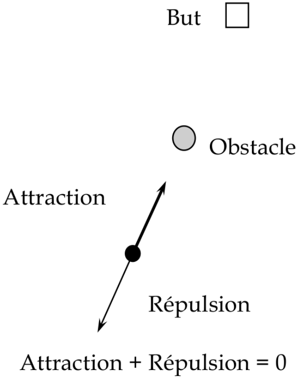
```
to flee
   let obstacles-in-front obstacles in-cone 3 angle-flee
   if (any? obstacles-in-front)
   [
      rt 180
      rt random 10
      lt random 10
   ]
end

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

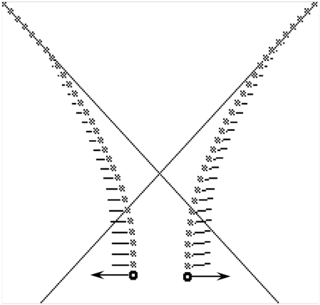
# Composition de vecteurs




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

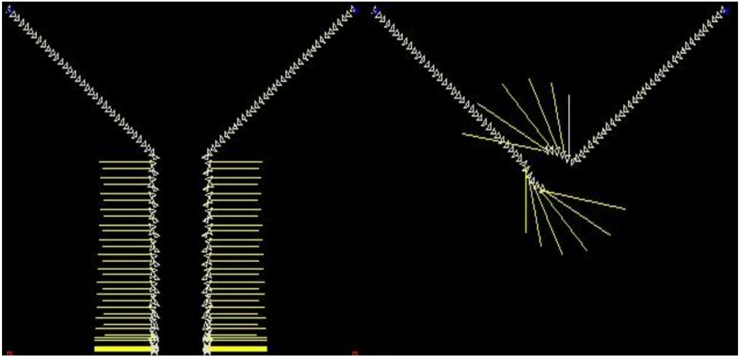
# Mais la répulsion n'est pas l'évitement



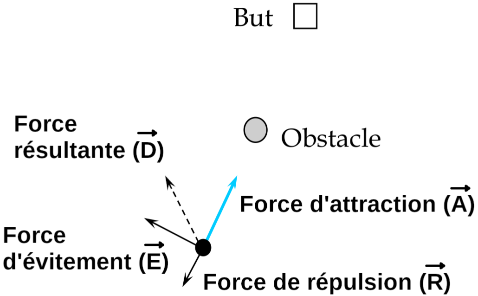



# L'évitement

**Attraction+Répulsion** **Attraction+Répulsion+Evitement**



# Force d'évitement




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

$$\vec{E}(p) := \vec{R}(p).\vec{E}(p) = 0$$

$$\vec{D}(p) = a\vec{A}(p) + r\vec{R}(p) + e\vec{E}(p)$$

# Calculer un vecteur perpendiculaire à un autre

## Vc est perpendiculaire à V si V.Vc = 0

Produit scalaire (Dot product) $->x.x_c + y.y_c = 0$

## Calcul

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

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

## Si le vecteur est normé (longeur=1):

$$\sqrt{x^2+y^2} = 1$$

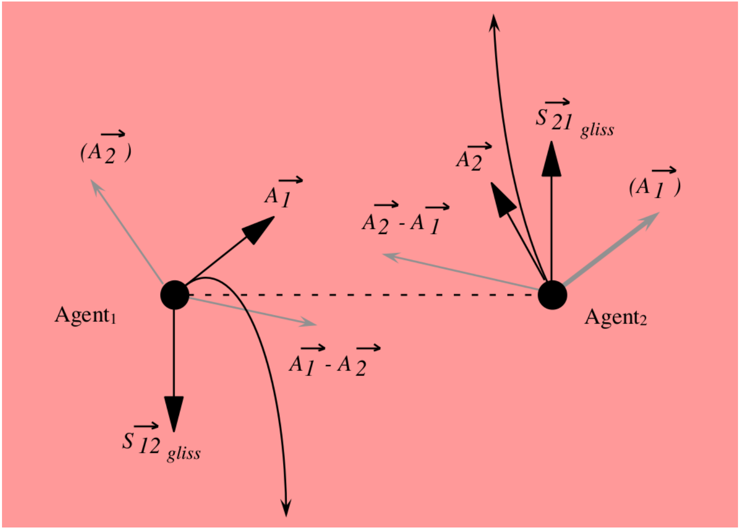
$$x_c = -y \text{ et } y_c = x$$

$$x_c = y \text{ et } y_c = -x$$

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
```

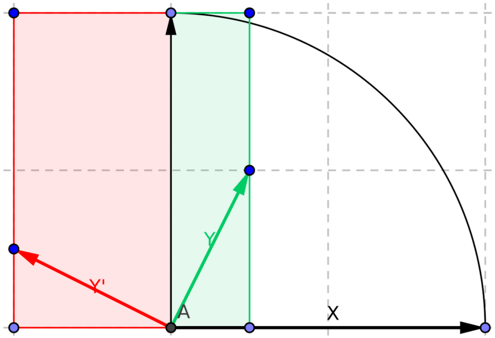
# Évitement dynamique: Forces d'évitement symétrique



# Produit scalaire

## Vc est perpendiculaire à V si V.Vc $= 0$

Produit scalaire (Dot product) $- > x.x_c + y.y_c = 0$




**X.Y' < 0   X.Y > 0**

# Opérations vectorielles

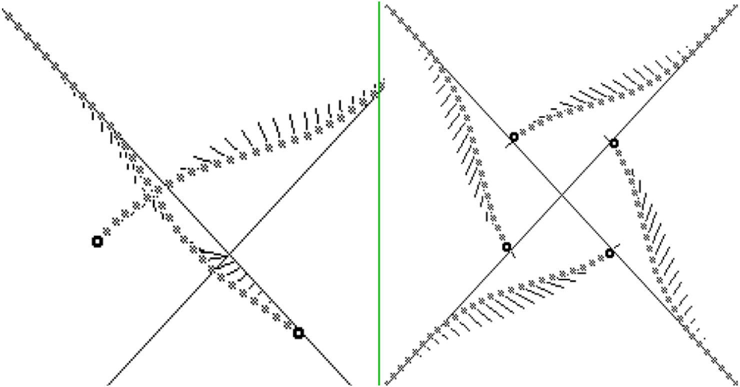
V4 $=$ aV1 $+$ bV2 $+$ cV3 où

```
let v1 vectorFromPolar ang1 len1
let v2 vectorFromPolar ang2 len2
let v2 vectorFromPolar ang3 len3

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

let ang4 getAngleFromVect v4
let len4 getLengthFromVect v4
```

# Fonctions vectorielles 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
```

# Emergence de structures dynamiques



# TP: Navigation

- Évitement de collisions statiques
  $\rightarrow$ *fuite / smart avoid*
- Navigation par champ de potentiel
  $\rightarrow$ *inondation*
- Évitement de collisions dynamiques
  $\rightarrow$ *force d'évitement*