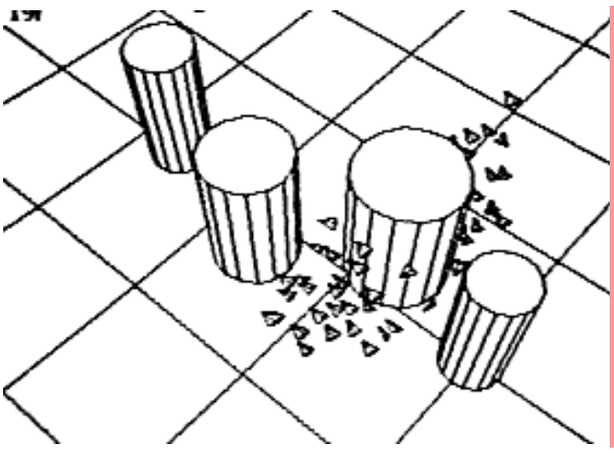
Université de Montpellier / Faculté Des Sciences

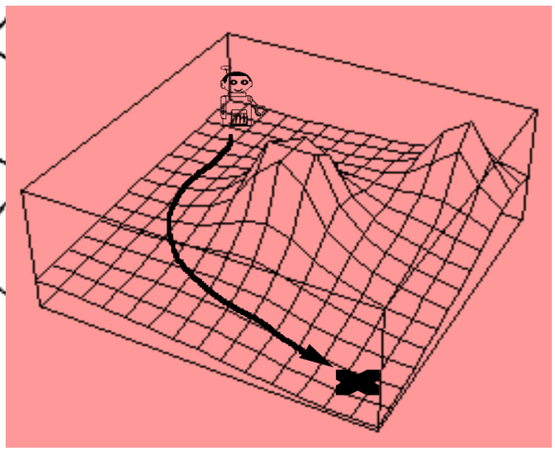
Navigation vectorielle

Algorithmes de mouvement — HMIN 233

Suro François
(adaptation des cours de Jacques Ferber)

8 décembre 2020





1 Évitement de collisions statiques

On souhaite ajouter un comportement d'évitement d'obstacles au comportement de flocking vu au tp précédent. Pour réaliser cela, on utilisera le fichier NetLogo `Flocking-obstacles.nlogo` où il suffit d'implémenter la procédure *avoid-obstacles*

1.1 Fuite

En partant de la base du flocking standard (choisir d'abord le modèle standard et ensuite le modèle vectoriel), implémentez un algorithme d'évitement standard qui consiste à aller dans la direction opposée quand un agent rencontre un obstacle. On parle alors de fuite.

1.2 Évitement

Quand un obstacle entre dans son rayon de perception l'agent cherche à éviter l'obstacle. Pour cela l'agent se détourne de sa direction initiale d'un angle inférieur à un angle max (*max-avoidance-turn*). On verra que dans ce cas, il est possible que les agents entrent en collision avec les obstacles. Dans ce cas, implémentez, lorsque la distance est vraiment trop courte (environ 2 à 3 unités), un comportement de fuite.

Notez les différences entre modèle standard (algo) et vectoriel

2 Champ de potentiel fluide

On suppose qu'il existe un ensemble d'obstacles fixes et de buts fixes. On va essayer de créer dynamiquement un champ de potentiel considéré comme une vague (flooding = inondation). Chaque but propage des valeurs qui vont se diffuser dans l'environnement et simplement contourner les obstacles statiques.

Vous adapterez ensuite l'algorithme d'inondation de manière à prendre en compte un potentiel négatif des obstacles.

Enfin vous testerez votre algorithme en faisant circuler un ensemble d'agents sur ces champs de potentiel.

Pour réaliser cela, on utilisera le fichier NetLogo `Flooding.nlogo`.

Principe de l'algorithme (fonction *compute-potential*) :

```
While continue [
   set continue false
   ;; si un des patches juge qu'il faut continuer
   ;; a inonder, il mettra continue a true
   ask patches [inonder]
]
```

Étapes :

1. Écrivez le code de la fonction inonder qui crée un champ de potentiel qui contourne les obstacles.
2. Écrivez le comportement des agents pour suivre ce champ de potentiel
3. Adaptez la fonction inonder de manière à faire en sorte que les obstacles soient des repoussoirs de quelques pixels, en utilisant le même principe : les obstacles génèrent un champ de potentiel négatif.

3 Évitement de collisions dynamiques

On considère maintenant que chaque agent possède un rayon de répulsion, un cercle définit par une taille fixe (le rayon du cercle). On suppose qu'il existe des agents verts et des agents bleus de manière à avoir des groupes de couleur différente.

Modifiez les algorithmes d'évitement d'obstacles pour qu'ils prennent en compte l'évitement des agents les uns vis-à-vis des autres, et notamment que les bleus évitent les verts et réciproquement.

3.1 Répulsion d'agents

Implémentez, avec l'algorithme "standard" l'évitement par répulsion entre les agents bleus et verts.

3.2 Évitement d'agents

À partir de vos TP précédents sur l'évitement proprement dit, par priorité ou contrôle par vecteur, proposez une technique anti-collision entre les agents.

3.3 Évitement d'agents par "forces de glissement"

Implémentez l'algorithme fonctionnant à partir des forces de glissement (on utilisera la technique vectorielle vue auparavant). Ici le vecteur E est défini comme étant perpendiculaire au vecteur qui relie l'agent à l'obstacle (heading $\pm$ 90). Le côté du glissement est donné par la différence vectorielle (voir cours).