Université de Montpellier / Faculté Des Sciences

Relation à l'environnement + machines à états finis
Les boeufs musqués

Programmation orientée agent — HMIN 108

Suro François
(adaptation des cours de Jacques Ferber)

29 septembre 2020

On cherche à modéliser le comportement grégaire d'animaux, comme celui des boeufs musqués, qui doivent lutter contre le froid en se serrant les uns contre les autres, lutter contre des loups et se reproduire. On retrouve ce comportement aussi chez les manchots et les éléphants de mer.

Le TP est très long, essayez toutefois d'aborder les machines à état finis

1 Environnement et nourriture

On suppose que les boeufs musqués cherchent à manger. On va commencer par implémenter un système de croissance d'herbe.

1.1 Croissance simple

Dans un premier temps, on suppose que les patches n'ont que deux états : l'état "plein d'herbes" et l'état "désert". Mais le temps de passage de l'état désert à l'état "herbe" dépend d'un compteur de temps *cpt-temps*, propre à chaque patch, qui indiquera le temps qu'il lui reste à attendre avant de passer de l'état "désert" à l'état "herbeux".

Ce compteur est initialisé à une valeur définie de manière aléatoire pour chaque patch entre 1 et *temps-croissance-max* qui sera associé à un slider.

Le principe est le suivant :

1. à l'initialisation, le patch est dans l'état "desert" (couleur noire). On calcule une valeur initiale pour cpt-temps (entre 1 et *temps-croissance-max*). A chaque pas on décrémente le compteur *cpt-temps*.
2. Lorsque *cpt-temps* est à zéro on change l'état du patch (qui passe alors à "herbeux", en changeant la couleur à vert).

1.2 Ressource

On supposera maintenant qu'il n'y a plus deux états possibles, mais une quantité de ressource disponible. Au lieu d'utiliser la couleur, nous utiliserons un attribut pour représenter cette information (attribut *taille-plante*). Cet attribut sera incrémenté à chaque fois que le compteur *cpt-temps* revient à zéro.

On attribuera une valeur maximum pour la taille des plantes (*taille-plante-max*) qui sera associée à un slider (par exemple entre 50 et 200)

l'algorithme devient :

1. à l'initialisation, le patch possède une quantité de ressource (entre 0 et taille-plante-max/2 par exemple). On calcule une valeur initiale pour *cpt-temps* (entre 1 et *temps-croissance-max*). A chaque pas on décrémente le compteur cpt-temps.
2. Lorsque cpt-temps est à zéro on change l'état du patch (on ajoute 1 aux ressources), puis on réinitialise le compteur de croissance (par exemple à *temps-croissance-max*).

L'utilisation d'un attribut dédié nous a aussi permis de séparer une information essentielle de modèle d'une information d'affichage (l'attribut couleur). Cependant, pour pouvoir observer notre modèle nous avons besoin d'afficher cette information. Créez une procédure dédiée qui demande à chaque patch de changer sa couleur en fonction de ses ressources, par exemple :

```
set pcolor scale-color green taille_plante 0 (taille_plante_max * 2)
```

1.3 Consommation

On suppose que les boeufs disposent d'un attribut *energie* et qu'il existe une variable globale qui correspond à la quantité de nourriture qu'ils consomment (*consommation-herbe* sous la forme d'un slider) et une variable globale qui correspond à la quantité d'énergie gagné quand ils mangent (*gain-energie* sous la forme d'un slider).

à chaque cycle, les boeufs perdent une certaine valeur d'énergie. Ceci est un comportement de "gestion" (de régulation, involontaire) qui est actif dans tous les cas de figure et se produit même quand d'autres comportements sont actifs. La valeur de cette perte d'énergie sera mise sous la forme d'un slider (conso-energie). Lorsque la variable d'énergie arrive à zéro, les boeufs meurent.

Quand la valeur de l'attribut *energie* descend en dessous d'une certaine valeur (energie-min), ils se mettent en quête de nourriture. Implémenter la procédure quete-nourriture qui leur fait rechercher de la nourriture.

Pour manger, il faut que le patch possède plus de ressource (d'herbe) que la valeur de *consommation-herbe*. Si c'est le cas, retirez *consommation-herbe* à *taille-plante* (les ressource du patch) et ajoutez *gain-energie* à l'énergie du boeuf.

2 Machines à états finis et comportements

La procédure *quete-nourriture* va constituer un état de la FSM de nos boeufs. Cet état va se déclencher dès que l'énergie est faible. Quand le niveau d'énergie est suffisant, cet état va transitionner vers un des autres états suivants.

Pour représenter l'état actuel de l'agent, ajoutez un attribut *etat* a votre agent qui prendra en paramètre le nom d'une procédure correspondant à l'état.

L'état sera exécuté en utilisant la primitive *run* de NetLogo qui appelle une procédure par son nom sous forme de chaîne de caractères.

```
set etat "quete-nourriture"
run etat
```

est équivalent à exécuter la procédure *quete-nourriture*.

2.1 se regrouper

On veut créer un phénomène de regroupement. Pour cela on désire que les boeufs musqués soient capables de se regrouper mais sans se retrouver sur le même patch. Il est donc nécessaire qu'ils s'éloignent s'ils sont trop près et qu'ils restent à la bonne distance pour former un troupeau serré d'animaux.

Implémenter un phénomène de regroupement entre les agents à partir d'une détection des autres agents (on utilisera la primitive in-radius vue en cours et en TP). On supposera qu'ils avancent de manière aléatoire tout en cherchant à rester en contact avec les autres.

Visualiser le regroupement des boeufs en fonction du paramètre distance-min qui gère la distance minimale que les boeufs musqués peuvent avoir entre eux (lorsque leur distance devient inférieure à distance-min, ils se repoussent, comme des porc-épic..)

Les boeufs ont maintenant deux états : un état correspondant à l'activité regrouper, qui est actif tant que l'énergie est suffisante, et *quete-nourriture* qui se déclenche dès que dès que l'énergie est faible.

Implémenter les boeufs avec les transitions d'un état vers l'autre.

2.2 se reproduire

On suppose maintenant qu'il existe deux types de boeufs : des enfants et des adultes (il est préférable de mettre le statut "enfant/adulte" sous la forme d'un attribut). Les adultes peuvent produire de nouveaux enfants, tous les *temps-generation* (slider). D'autre part, les enfants peuvent devenir adulte lorsque le temps enfance est passé (gérez reproduction et enfance sous la forme d'un compteur que l'on décrémente, comme on l'a vu pour d'autres cas, par exemple la croissance des plantes).

Lors de la génération d'un nouvel enfant, l'énergie du parent est divisée en deux, et l'enfant récupère la moitié de l'énergie du parent.

La primitive *hatch* permet de créer un nouvel agent :

```
hatch-boeufs 1 [set color blue]
```

crée un nouveau boeuf et change sa couleur à bleu.

2.3 Affronter les loups

On suppose maintenant qu'il existe des loups qui cherchent à attaquer les boeufs. Lorsque les boeufs sont attaqués (lorsqu'un loup se trouve dans leur rayon de perception), ils arrêtent de manger et deviennent "bleus" (ou "vert" de trouille) pendant un temps.

Dans cet état de peur, les enfants fuient les loups, et les adultes font face aux loups en restant immobile (comme le montre la photo sur cette page).

Enfin, on suppose que :

1. les loups peuvent manger des boeufs enfants s'ils se trouvent sur la même case. la primitive *die* fait mourir un agent :

   ```
   ask turtle 0 [die]
   ```

2. les loups ne peuvent pas avancer si un adulte se trouve devant eux, et doivent reculer ou faire le tour (ils sont intimidés).

Implémentez ce nouvel état (*avoir-peur*) qui traduit deux comportements différents, selon que l'on est adulte ou enfant.

3 Le grand défi de la simulation

Mettez tout cela ensemble et ajustez les paramètres pour que votre population survive.

Si vous avez du courage, ajouter un système d'énergie, de mort et de reproduction pour les loups.

Que se passe-t-il quand une espèce se reproduit trop ? pas assez ? Que se passe-t-il quand les prédateurs disparaissent ? les proies ?

Il est extrêmement difficile d'obtenir un équilibre durable (sans tricher en tout cas).

Bravo si vous êtes arrivés jusque là...