TP 2 - Programmation orientée agent

proposé par Jacques Ferber
2016-2018

On va essayer de modéliser un système de gestion de ressources en NetLogo.

1. Faire évoluer l'environnement

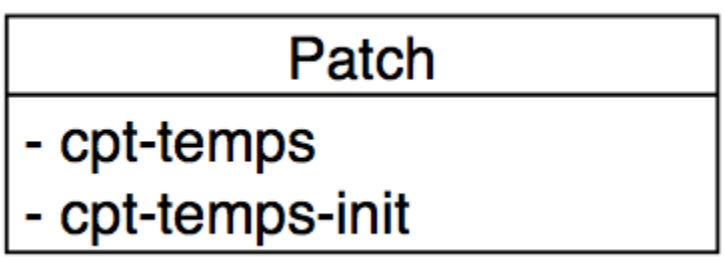
a) Dans un premier temps on va faire "pousser" des ressources dans un environnement.. Pour cela on va supposer que les patches peuvent faire pousser de l'herbe...

On supposera que chaque parcelle de terrain (représentée par un patch) peut être soit pleine d'herbe (verte), soit déserte (noir). On va supposer que l'herbe repousse naturellement chaque patch disposant de sa propre vitesse de croissance.

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' et inversement.

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

Le diagramme de classe du patch est le suivant:



Le principe est le suivant:

- à l'initialisation, le patche est dans l'état `'desert'` (couleur noire). On calcule une valeur initiale pour cpt-temps-init. Puis on met cette valeur initiale dans cpt-temps. A chaque pas on décrémente le compteur cpt-temps.
- Lorsque cpt-temps est à zéro on change l'état du patch (qui passe alors à `'prairie'`), puis on réaffecte la valeur `cpt-temps-init` à la variable `cpt-temps` et on continue.
- Lorsque `cpt-temps` arrive à zéro, le patche passe dans l'état `'desert'`, etc..

Implémentez de tels patches

> **Note**: dans la version 5 de NetLogo il est important de mettre une instruction '`clear-all`' au début et un 'reset-ticks' à la fin de votre procédure d'initialisation (`setup`). Cela donne donc ceci:
>
> ```
> to setup
>     clear-all
>     ...     ;; initialisation
>     ...          ;; du programme
>     reset-ticks
> end
>
> to go
>     ..          ;; la procedure qui sera appelee a céquà cycle...
>     tick
> end
> ```

b) On supposera maintenant qu'il n'y a plus deux états possibles, mais une variable continue qui caractérise la quantité de plantes qu'il y a sur un patch (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)

Pour visualiser la taille des plantes, on pourra utiliser la fonction `scale-color` de NetLogo qui retourne une couleur proportionnelle à une valeur (aller regarder sa définition dans le manuel de programmation de NetLogo). Par exemple, pour mettre la couleur d'un patche en correspondance avec la valeur de l'attribut '`taille-plante`', quelque chose comme cela devrait faire l'affaire (Attention: c'est juste une proposition: essayez avec d'autres valeurs que 70 et faites en sorte ensuite que le gradient de couleur soit relié à la taille maximale des plantes (`taille-plante-max`) tout en conservant une belle couleur verte sur les patches.

```
        set pcolor scale-color green taille-plante 0 70
```

Visualisez ainsi des patches qui poussent et croissent à partir de valeurs aléatoires initiales. On supposera que lorsque la valeur de l'attribut `taille-plante` est à taille-plante-max le patch voit la taille des plante décroire, et quand elle arrive à zéro (ou négatif), le patch devient désert et il recontinue sa croissance ensuite.

2. Créer des consommateurs de ressources

On fait maintenant venir des vaches qui viennent brouter l'herbe. On suppose qu'une vache prend un peu de la valeur de `taille-plante`, c'est à dire qu'elle diminue la quantité d'herbe qui se trouve sur un patche.. Chaque fois qu'une vache tombe sur un patche vert, elle diminue sa valeur taille-plante d'une valeur `consommation-vache` qui est définie de manière globale (c'est un paramètre auquel on associe un slider). Evidemment l'herbe repousse ensuite.

Faites en sorte que les vaches se déplacent aléatoirement et broutent.. Voyez ce qui se passe en fonction des paramètres temps-croissance et consommation-vache.. Qui va gagner de l'herbe ou des vaches?

Visualiser la quantité totale de vaches et d'herbe à l'aide d'un moniteur d'interface de type 'plot'.

> **Note**: pour visualiser une courbe sous NetLogo : à la fin de la procédure to go, ajouter la ligne suivante:
>
> ```
>         update-plots
> ```
>
> Et créez un "plot" (comme on crée un bouton) et dans le pen (le crayon) "default" (dont vous pouvez changer le nom en "herbe" par exemple), écrivez l'expression suivante
>
> ```
>    plot sum [taille-plante] of patches
> ```
>
> Cela signifie qu'à chaque tour, il affiche dans le "plot", pour la courbe "herbe", la totalité d'herbe qui existe dans votre terrain..
>
> Pour plus d'informations sur la manière de visualiser une courbe sous NetLogo, voir la documentation de l'interface de NetLogo.

3. Troupeaux de vaches

Faites en sorte que vos vaches se déplacent en troupeau.. Utilisez la technique que vous avez mis au point lors de la poursuite des reines par les abeilles. Faites un "taureau" et un ensemble de vaches qui le suivent (comme pour les reines et les abeilles). En même temps qu'elles avencent et suivent le taureau, elles se repaissent de l'herbe qu'elles croisent.. Voyez l'évolution de votre terrain en fonction du déplacement des vaches..

4. Prédateurs

a) On crée maintenant un ensemble de prédateurs, des lions, qui viennent manger les vaches (ils ne mangent pas les taureaux!!). Ces lions avancent de manière aléatoires, et dès qu'ils repèrent une vache dans un certain rayon de perception (de 3 à 20 par exemple, mettre cela aussi sous la forme d'un paramètre et d'un slider), ils lui foncent dessus, et s'il se trouve sur le même patche que la vache, il la mange (En NetLogo, le code `other <breed>-here` retourne toutes les tortues de type <breed> qui se trouvent sur le même patch, la tortue courante ayant été omise. Attention, même s'il y a plusieurs vaches sur le même patch, il ne doit en manger qu'une!! Regardez dans la doc la définition de `other` et de `turtles-here`). La primitive pour tuer un agent est 'die'. Si l'agent se "tue" lui-même, il suffit de faire

```
die
```

Si on demande à un autre agent de se tuer, on peut le faire en utilisant la commande:

```
ask <agent> [die]
```

Regardez ce qui se passe en fonction du nombre initial de lions et de vaches.

b) On donne maintenant des capacités de reproduction aux lions et aux vaches. On suppose d'abord qu'ils disposent tous les deux d'une variable énergie qui est décrémentée à chaque tick (à chaque pas de temps). Cette valeur est incrémentée lorsque les vaches mangent l'herbe (elles prennent un certain quota d'énergie à chaque fois qu'elles prennent de l'herbe) et lorsque les lions mangent les vaches (les lions prennent un certain quota d'énergie à chaque fois qu'ils mangent une vache). Ces quotas d'énergie correspondent à des paramètres `gain-brouter` pour les vaches et `gain-devorer` pour les lions. Leur attribut énergétique s'exprime alors sous la forme:

```
        set energie energie + gain-brouter
```

pour la vache chaque fois qu'elle broute et idem pour le lion à chaque fois qu'il mange une vache (avec la valeur `gain-devore`r dans ce cas). On suppose qu'ils se reproduisent de manière aléatoire, ce nombre aléatoire étant pris dans l'étendue `0..reproduction-vaches` pour la reproduction des vaches (et de la même manière pour les lions). On suppose aussi que la valeur énergétique d'un animal est divisée par deux lorsqu'il se reproduit.

En NetLogo, la commande `hatch 1 [... ]` permet de créer une copie de la tortue courante, l'enfant se trouvant au même endroit que le parent.

Jouez un peu avec les variables et observez l'évolution de votre monde dans lequel de l'herbe pousse, des vaches et des lions évoluent... A l'école des dieux, savez vous créer un monde qui perdure?