TELECOM Bretagne

Institut Mines-Télécom

La modélisation à base-d'agents avec Net-logo

SugarScape

Un laboratoire d’exploration d’hypothèse

Growing artificial societies Epstein et Axtell

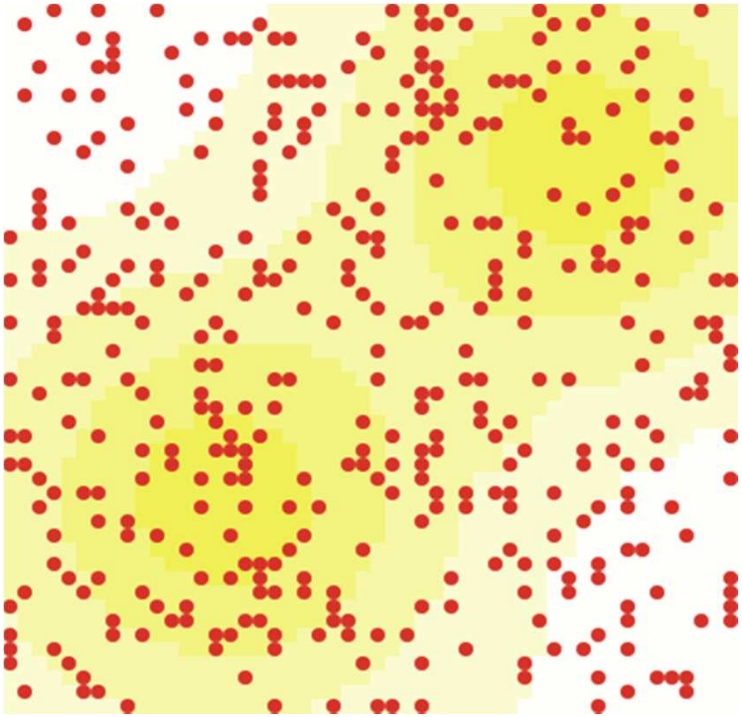
- ***Au début* était un monde hostile avec du sucre disposé sur des bouts de terrain :**

  - ✓ la topographie du monde avec deux monts sur une grille de 50 X 50 cellules
  - ✓ Chaque cellule contient entre 0 (blanc) et 4 unités de sucre (jaune foncé):
    - variable patch: max-psugar
  - ✓ Une règle de repousse G_α de α unités par pas de temps avec un maximum de max-psugar.
  - ✓ G_1 s'applique au départ

Growing artificial societies Epstein et Axtell

- **400 agents (turtles) chargés de la collecte pour survivre et définis par**

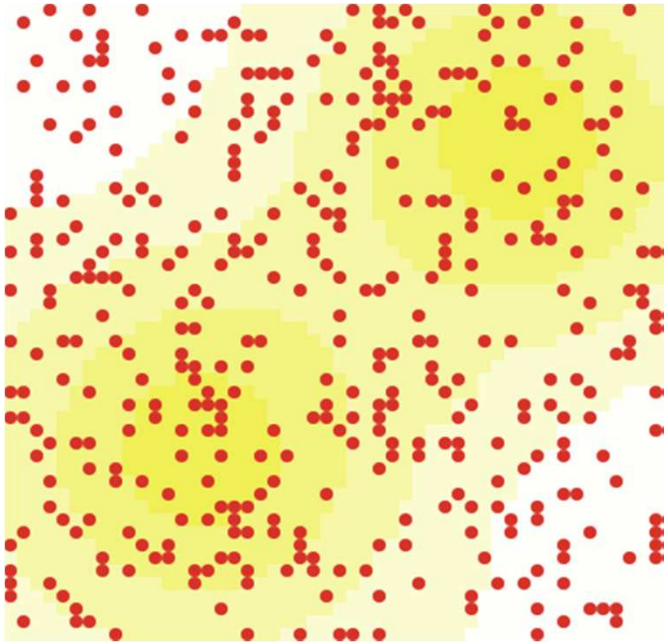
  - ✓ Leur location (x y) aléatoire à l'initialisation avec une seule position par agent
  - ✓ Un métabolisme en sucre: quantité de sucre détruit à chaque itération (période de temps) : entier entre 1et 4
  - ✓ Une vision v : distance visible pour l'agent. L'agent peut voir dans chacune des 4 directions cardinales : est , sud, ouest, nord (mais pas en diagonale). Elle est définie à l'initialisation comme une v.a. entière entre 1 et 6 pour chaque agent.



Growing artificial societies Epstein et Axtell

- **400 agents définis par**

- ✓ Une règle de déplacement M
  - Regarde aussi loin que possible (i.e. fonction de v) dans chacune des 4 directions et identifie les sites inoccupés ayant le plus de sucre
  - Si plusieurs sites, l'agent sélectionne le plus proche et se déplace sur ce site
  - Il cueille le sucre sur place et accroit sa richesse (wealth)
- ✓ (G_1 M) définissent les règles de la société , G_1 pour l'environnement et M pour les agents.



- **Ouvrir netlogo et aller dans file, models library et social science → sugarscape 2 constant grow back**
  - ✓ Etudier le fonctionnement et le code de ce modèle proposé par uri wilensky

Concept Netlogo: associé au modèle sugarscape

- **Voir dans le résumé les concepts suivants de netlogo**
  - ✓ ensemble d'agents
  - ✓ Listes

Quelques morceaux croustillants de sugarscape : run *command-task* run string

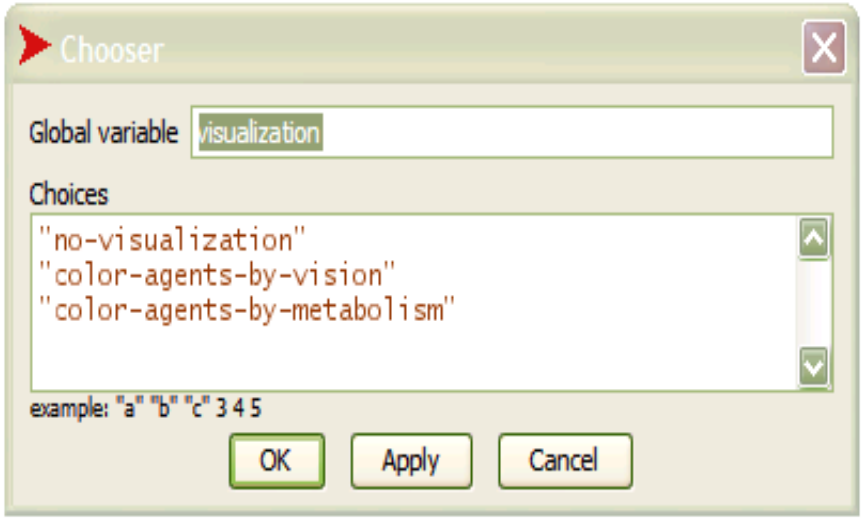
Run attend une commande ou une chaine de caractère (string) contenant une commande (le cas ici) . Puis l'agent execute la commande

run visualization ; renvoie au nom de la commande (visualization) ici un chooser avec trois chaines de caractère (strings) renvoyant à trois procédures

```
to no-visualization ; turtle procedure
  set color red
end

to color-agents-by-vision ; turtle procedure
 set color red - (vision - 3.5)
end

to color-agents-by-metabolism ;turtle procedure
  set color red + (metabolism - 2.5)
end
```



Quelques morceaux croustillants de sugarscape

```
 patches with [not any? other turtles-here]  ; retourne un ensemble d'agents de type patches et selectionne  tous ceux qui n'ont pas de turtles sur eux

foreach n-values vision [? + 1]
  [
    set vision-points sentence vision-points (list (list 0 ?) (list ? 0) (list 0 (- ?)) (list (- ?) 0))
  ]
```

foreach *<list> <commande> :*

pour chaque item de la liste exécute la <commande>

Ici la liste est n-values vision [? + 1]

→ si vision = 2, n-values 2 [? + 1] renvoie la liste [1 2]

La <commande > est ici

```
[
    set vision-points sentence vision-points (list (list 0 ?) (list ? 0) (list 0 (- ?)) (list (- ?) 0))
  ]
```

Quelques morceaux croustillants de sugarscape

Tache-commande est ici

```
[
   set vision-points sentence vision-points (list (list 0 ?) (list ? 0) (list 0 (- ?)) (list (- ?) 0))
 ]
```

→ Définit une liste vision-points :

→ sentence créée une liste des différents éléments : show sentence 1 2 => [1 2]

→ si la vision est 1 renvoie la liste de coordonnées [[0 1] [1 0] [0 -1] [-1 0]]

→ Si la vision est 2 :

→ Prend le premier élément de la liste n-values 2 = [1 2] soit 1

→ Renvoie la liste vision-points = [0 1] [1 0] [0 -1] [-1 0]

→ Prend le deuxième élément soit 2 et renvoie la liste [0 2] [2 0] [0 -2] [-2 0]

→ Sentence concatene la première liste vision-points = [0 1] [1 0] [0 -1] [-1 0] avec le liste [0 2] [2 0] [0 -2] [-2 0]

→ Le résultat est [[0 1] [1 0] [0 -1] [-1 0] [0 2] [2 0] [0 -2] [-2 0]]

→ Si vision = 4

→

[[0 1] [1 0] [0 -1] [-1 0] [0 2] [2 0] [0 -2] [-2 0] [0 3] [3 0] [0 -3] [-3 0] [0 4] [4 0] [0 -4] [-4 0]]

Quelques morceaux croustillants de sugarscape

```
let move-candidates (patch-set patch-here (patches at-points vision-points) with [not any? turtles-here])  ; turtle procedure
```

Renvoie un ensemble d'agents (patch-set) définit par le patch de la tortue demandant l'execution (patch-here) et des patches à distance de chaque item de la liste vision-points qui ne contiennent pas de turtles sur eux (with [not any? turtles-here]) .

Expérimentation à faire

Expérimentation: remplacement par décès

- **Exercice 1 : introduire les variables suivantes pour les tortues:**
  - ✓ age ; the current age of this turtle (in ticks)
  - ✓ max-age ; the age at which this turtle will die of natural causes
- **dans le setup**
  - ✓ Fixer max-age par une uniforme entre 60 et 100
  - ✓ Fixer age à zéro
- **Dans le go**
  - ✓ mettre en place une procédure qui augmente l'age à chaque tick
  - ✓ Ayant atteint l'age de la mort , remplacer le turtle par un nouveau né (aux conditions initiales aléatoires) et faire mourir le "veillard"

Experimentation: remplacement par décès

- **Exercice 1 (cont): mettre deux curseurs**
  - ✓ minimum-sugar-endowment
  - ✓ maximum-sugar-endowment
  - ✓ Expérimenter le changement de la distribution quand on fait varier le curseur
- **Pour vous assurer que cela marche, il faut mettre en place une procédure qui s'assure que** maximum-sugar-endowment est bien supérieur à minimum-sugar-endowment

```
if maximum-sugar-endowment <= minimum-sugar-endowment  [
user-message "Oops: the maximum-sugar-endowment must be larger than the minimum-sugar-endowment  "
stop
        ]
```

Expérimentation : migration

- **Exercice 2 : Migration sans saisonalité:**
  - ✓ Mettre en place un switch qui active une condition migration?
  - ✓ placer les turtles dans le quadrant sud-ouest (x < 24 and y< 24)
  - ✓ Mettre en place deux curseurs min-vision et max-vision
    - — Pour vous assurer que max-vision > min-vision il suffit de rentrer min-vision comme paramètre du curseur max-vision
  - ✓ Explorer le modèle
  - ✓ Netlogo code spécial: ifelse-value

Expérimentation : migration

- **Exercice 2 : migration avec saisonalité:**
  - ✓ Mettre en place un switch qui active une condition saisonalité? (saisonalité)
  - ✓ on définit deux nouvelles variables pour les patches:
    - Season ; la saison courante de l'hémisphère
    - old-season ; mémoire temporaire qui sera utilisée pour switcher de saison
  - ✓ L'hémisphère sud est en été à l'initialisation
  - ✓ Définir une procédure patch-change-season qui change de saison tous les 20 ticks: les cellules dans le quadrant nord prennent la saison des cellules du quadrants sud (et vice –versa)
  - ✓ En saison hiver la repousse (G_α) est de 0 et en été de 1
  - ✓ Rajouter un label à un des patchs sud et nord montrant la saison de l'hémisphère
  - ✓ Explorer le modèle

Expérimentation: reproduction en exercice personnel

- **Exercice 3: reproduction**
  - ✓ Un switch si reproduction?
  - ✓ Chaque agent reçoit un type « male » ou « femelle » (variable sexe)
  - ✓ Un agent est fertile si il a au moins autant de sucre qu'à la naissance (variable fertile?)
  - ✓ Règle de reproduction pour chaque agent fertile
    - chaque agent fertile sélectionne un voisin de VNM au hasard:
    - Si le voisin est fertile et de sexe opposé et au moins un des deux agents à une cellule vide autour de lui alors un bébé est né sur une de ces cellules