Oscar BIDOU & Thomas SPIRY

Simulation d'un comportement Avec MESA

Wolf Sheep édition

Table des matières

Table des figures

Connaissances utilisées

- Modéliser un phénomène ou un problème notamment avec des outils informatiques associés aux outils mathématiques ou des sciences cognitives.
- Mettre en synergie des connaissances pluridisciplinaires pour proposer des solutions innovantes, en combinant informatique, psychologie, biologie et technologies cognitives.
- Conduire un projet de manière autonome ou en équipe
- Programmation Python 3

Simulation

Objectif de Simulation

Cette simulation a pour but de valider l'hypothèse de la sélection naturelle, dans laquelle les espèces c'étant le mieux adapté survivent et transmettent cette capacité à leurs descendants. Les modèles centrés sur les agents (ABM) sont des modèles de simulation utilisés pour représenter les actions et interactions d'agents autonomes, comme des individus ou des groupes et ils permettent de comprendre le comportement global d'un système en fonction de règles simples suivies par les agents. Ils peuvent modéliser des systèmes naturels, comme des interactions prédateurs-proies. Par exemple, pour simuler notre population de loups (wolfs) et de moutons (sheeps), un ABM permettrait d'étudier comment des règles globales de chasse de reproduction et de déplacement se décident selon des décisions locales, (d'évitement par exemple chez les moutons afin de ne pas croiser un loup) et comment ils influencent l'équilibre de l'écosystème.

Explication du modèle

Le modèle que nous avons créé se base sur le modèle préexistant dans la librairie MESA, appelé wolf_sheep. Ce modèle introduit trois types d'agents : les loups, les moutons et l'herbe. Les loups mangent les moutons, tandis que les moutons mangent de l'herbe. Les moutons et les loups se déplacent de manière aléatoire, sans stratégie, et se reproduisent lorsqu'ils dépassent un certain seuil énergétique. Il est à noter que l'environnement simulé est discret pour des raisons d'optimisation.

Afin de modéliser la survie de la stratégie la plus forte, nous avons commencé par créer une stratégie linéaire individuelle pour chaque loup et chaque mouton. Pour simplifier le

coût computationnel de la simulation, nous avons opté pour un perceptron simple. De plus, nous avons enlevé l'agent herbe : en effet on suppose que les moutons trouveront de la nourriture là où ils sont en restant immobiles sur leur case. Les réseaux de neurones prennent en entrée l'environnement perçu par l'agent et produisent en sortie un déplacement.

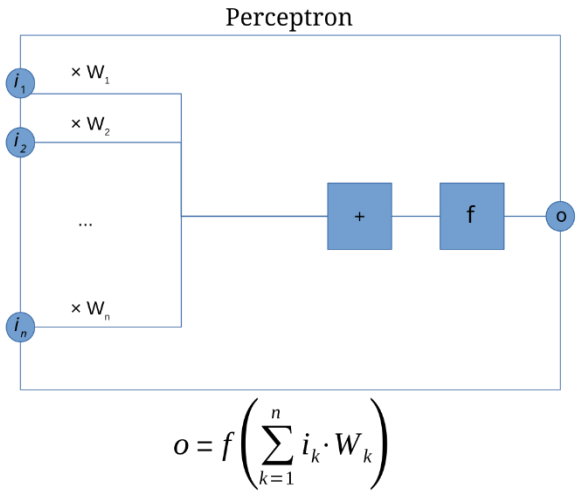


Figure 1 : Modèle d'un perceptron

Lors de l'initialisation, les poids des réseaux de neurones de chaque agent sont distribués uniformément entre -0.1 et 0.1. Les perceptrons utilisent une fonction d'activation "ReLU" et n'ont pas de biais. La perception de l'environnement se fait via la fonction get_neighborhood de mesa.model.grid. Dans un carré centré sur l'agent de taille 5x5, une case vide est représentée par 0, un loup par 1 et un mouton par 2. Le perceptron renvoie une valeur comprise entre 0 et 8.0 :

- 0 : déplacement à droite
- 1 : déplacement haut droite
- 2 : déplacement bas droite
- 3 : déplacement bas
- 4 : déplacement bas gauche
- 5 : déplacement gauche
- 6 : déplacement haut gauche
- 7 : déplacement haut
- 8 : ne rien faire

Lors de la reproduction, les agents transmettent leurs poids à leur progéniture avec un bruit gaussien de moyenne 0 et de variance 0.005, modélisant ainsi la mutation aléatoire. Les descendants des agents sont placés dans une case voisine de celle de l'agent parent. Les moutons comme les loups perdent de l'énergie à chaque pas de temps, mais les

loups en récupèrent lorsqu'ils mangent des moutons. Les moutons, quant à eux, ne perdent pas d'énergie lorsqu'ils ne font rien (sortie 8 du perceptron), mais en gagnent.

De cette manière, les stratégies les plus efficaces se reproduisent davantage, tandis que les comportements moins performants disparaissent. La variation aléatoire des poids des perceptrons permet un apprentissage générationnel des agents et une optimisation progressive des stratégies.

Afin de suivre l'évolution du modèle, nous imprimons à chaque pas de temps le nombre de loups et de moutons vivants. Simultanément, nous représentons la différence moyenne des poids des loups et des moutons par rapport à 0. Cette mesure permet de suivre l'évolution globale des perceptrons. Les poids étant initialisés selon une distribution uniforme de moyenne 0 et variant selon une distribution gaussienne de moyenne 0, si la théorie de la survie du plus adapté était fausse, nous ne devrions pas observer de valeurs significativement supérieures à 0. Il n'est cependant pas certain d'observer une convergence de cette mesure. En effet, si un comportement devient très dominant à un certain moment, les agents de l'autre type évolueront rapidement et modifieront ainsi le comportement optimal des agents du premier type.

Lorsque le nombre de moutons devient très important, le nombre de loups augmente rapidement. Ce grand nombre de loups provoque ensuite une chute drastique du nombre de moutons, affamant ainsi presque tous les loups. Les moutons peuvent alors se reproduire sans crainte. Chacun de ces cycles permet une sélection naturelle des comportements les plus performants et entraîne l'élimination des comportements déviants.

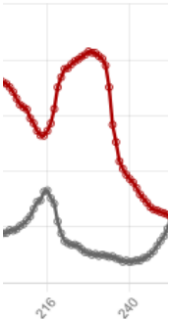


Figure 2 : Cycle de sélection naturel

Nous avons mis en paramètres du modèle la taille de la grille, le nombre initial de moutons, le nombre initial de loups, le gain de nourriture des loups lorsqu'ils mangent un mouton, le gain de nourriture des moutons lorsqu'il reste immobile.

Des paramètres invariables, les loups et les moutons nécessite 30 d'énergie pour se reproduire et perdent la moitié de celle-ci lors de la reproduction. Les moutons ont un nombre maximal de 1000 et les loups 2500 pour une raison d'optimisation computationnel.

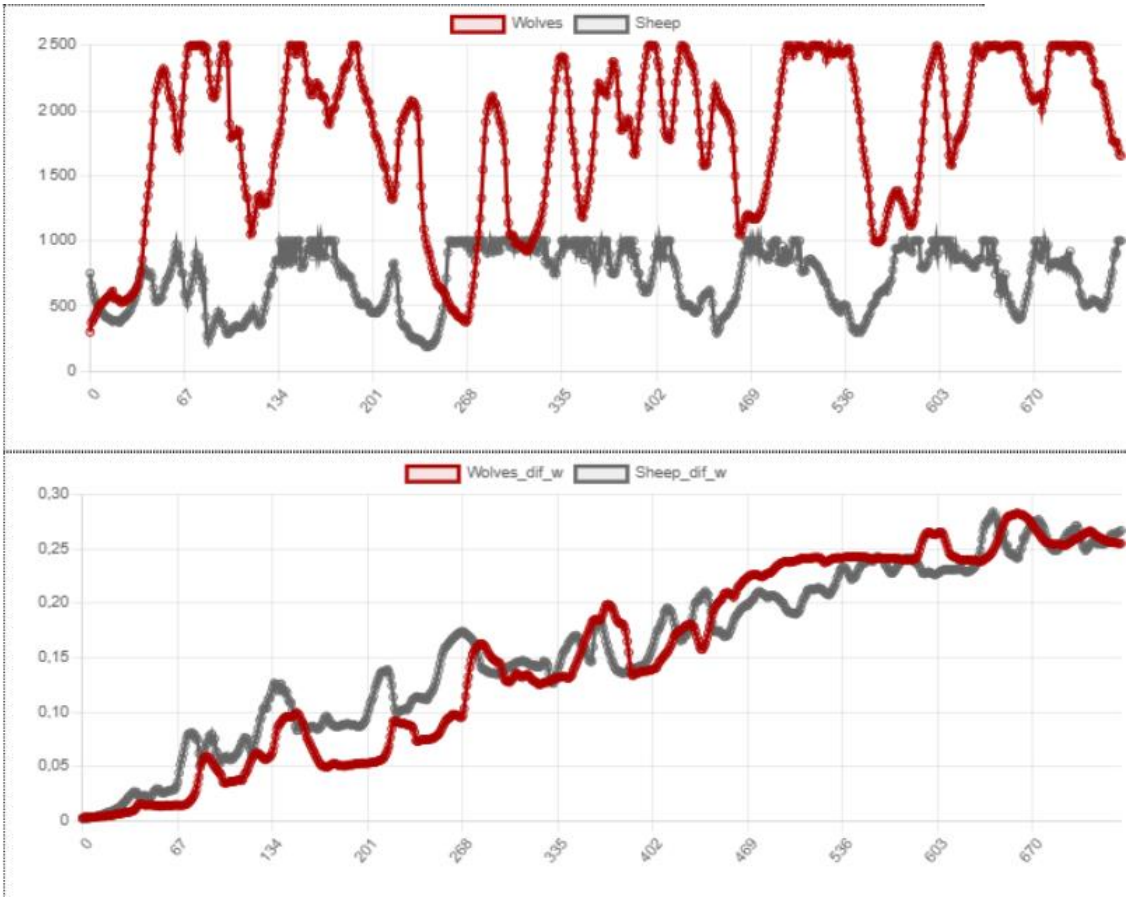
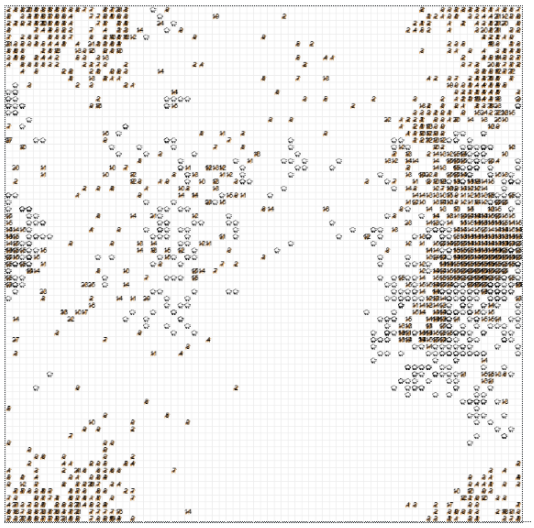


Figure 3 : Exemple de convergence des poids



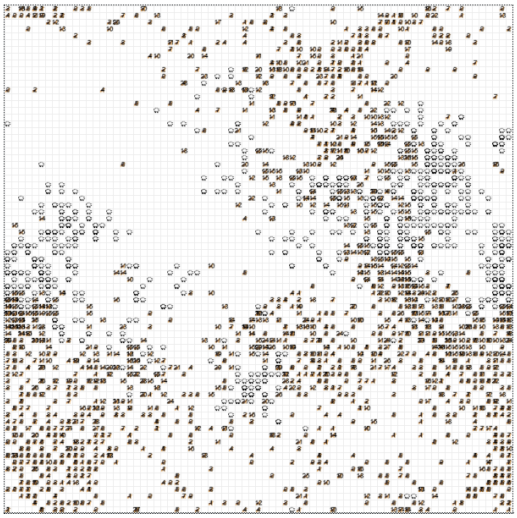


Figure 4 : Exemple de la simulation

Installation

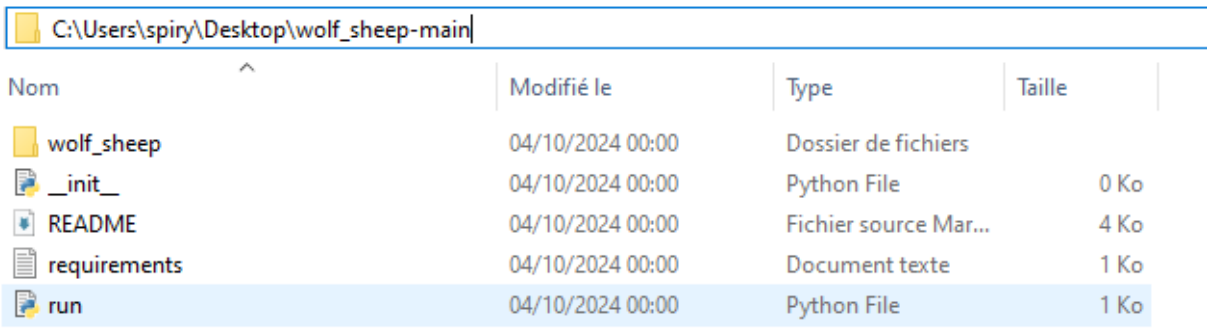
Pour installer les dépendances, utilisez pip install mesa et le fichier requirements.txt dans ce répertoire. Vous pouvez utiliser les commandes git bash suivantes pour le cloner directement :

$ git clone https://github.com/Oscar-B2001/wolf_sheep.git

$ cd mesa

Comment exécuter

1) Utiliser la commande cd « votre emplacement du fichier run » pour travailler dans le dossier correspondant
2) Pour exécuter le modèle, utilisez la commande « *mesa runserver* ». Quelques instants plus tard une page s'ouvrira dans votre navigateur web avec http://127.0.0.1:8521/ comme adresse.



```
C:\Users\spiry>cd C:\Users\spiry\Desktop\wolf_sheep-main

C:\Users\spiry\Desktop\wolf_sheep-main>mesa runserver
```

Fichiers

Vous pouvez retrouver l'ensemble de nos fichiers notre GitHub accessible ici : https://github.com/Oscar-B2001/wolf_sheep.git

- wolf_sheep/random_walk.py : Ce script définit la classe RandomWalker, qui implémente le comportement de déplacement aléatoire pour les agents. Les classes Wolf et Sheep hériteront de cette classe pour effectuer leurs déplacements sur la grille.
- wolf_sheep/agents.py : Définit les classes d'agents Loup et Mouton.
- wolf_sheep/scheduler.py : Ce script définit une variante personnalisée du planificateur RandomActivationByType. Il permet de gérer la programmation des agents dans le modèle et d'ajouter des filtres pour la fonction get_type_count, ce qui peut être utile pour compter et gérer différents types d'agents de manière plus flexible.
- wolf_sheep/model.py : Ce fichier contient la définition du modèle de prédation Loups-Moutons. Il intègre tous les agents, gère leurs interactions (comme la reproduction et la prédation. C'est le cœur de la simulation, où la logique principale du modèle est implémentée.
- wolf_sheep/server.py : Ce script configure le serveur de visualisation interactif qui permet aux utilisateurs de voir le modèle en action dans leur navigateur. Il gère la communication entre le modèle et l'interface utilisateur, permettant ainsi aux utilisateurs d'interagir avec la simulation.
- run.py : Lance un serveur de visualisation de modèle.

Lectures complémentaires

Ce modèle est basé sur le modèle de prédation Loups-Moutons de NetLogo :

Wilensky, U. (1997). Modèle de prédation Loups-Moutons dans NetLogo. http://ccl.northwestern.edu/netlogo/models/WolfSheepPredation. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.

Voir également les équations de Lotka–Volterra pour un exemple de modèle d'équation différentielle présentant des dynamiques similaires.

Vidéo

Observable sur Youtube : Notre simulation en temps réel