

RESUME

Ce document comporte une explication ainsi qu'un résumé du fonctionnement du projet de simulation OSCAR.

Martin Devreese

Programmation 2

OSCAR

Documentation

Introduction à l'OSCAR

OSCAR (Outil de Simulation Comportemental par Attraction-Répulsion) est un outil de simulation multi-agent (SMA) basé sur la communication d'automates exécutant des règles simples.

La simulation consiste en une grille dans laquelle sont représentés par des couleurs les différents agents. Les règles auxquelles sont soumis les automates sont très limitées mais permettent tout de même d'obtenir une certaine complexité de simulation.

Le programme réalisé va pour chaque agent de la grille parcourir les règles associées à cet agent et déterminer s'il est capable de se déplacer, de donner naissance à un autre agent ou de changer d'état. Le terme « autour d'un agent » fera référence au 8 cases adjacentes à cet agent.

Un des exemples les plus intéressants parmi ceux existant ou cités dans notre cours est la configuration de simulation nommée « Wireworld ».

Elle vise à simuler le déplacement d'un électron dans un circuit et est basée sur 3 agents différents :









- ✚ L'agent **tail** : agit comme la queue de l'électron, son unique règle est de devenir un agent **conductor**.
- ✚ L'agent **head** : agit comme la tête de l'électron, son unique règle est de devenir un agent **tail**.
- ✚ L'agent **conductor** : agissant comme le fil de cuivre, il agit selon les règles suivantes :
 - S'il y a 1 ou 2 agents **head** autour de lui, alors il devient **head**.
 - Sinon, il reste un agent **conductor**.

Cette configuration permet par exemple de reproduire le fonctionnement de portes logiques et même de créer un afficheur digital (<https://www.quinapalus.com/wi-index.html>).

Fonctionnalités

Mis à part la prise en charge de l'évolution des agents dans la grille qui n'est pas totalement fonctionnelle, le programme comporte plusieurs fonctionnalités utiles à la visualisation et à l'édition de la grille.

L'utilisateur peut :

-  Mettre sur pause et reprendre l'évolution des agents
-  Recharger la configuration d'origine
-  Avancer pas à pas dans la simulation
-  Activer un système permettant de déterminer si la simulation est terminée ou non (si les 2 dernières itérations sont identiques – ne convient pas aux simulations comportant des agents de type « animal » ou « végétal »)
-  Définir la vitesse d'itération de la simulation
-  Afficher uniquement un certain type d'agent
-  Charger un nouveau fichier de configuration
-  Modifier directement les agents sur la grille

Choix algorithmiques

Au niveau de l'implémentation sous Python, certains choix ont été fait concernant l'architecture de l'algorithme afin de minimiser le temps de calcul d'une itération. Malheureusement le programme n'est toujours pas assez performant et ne fonctionne pas comme demandé.

Fonctionnement des algorithmes :

Dès l'initialisation de la configuration de la grille, les différents agents présents dedans sont indexés dans un dictionnaire ce qui va permettre d'éviter le parcours complet de la grille avant le début des itérations. A chaque fin d'itération, les nouveaux placements des agents son réindexés et ainsi de suite...

La réception d'un champs « field » se fait, pour chaque agent, au travers d'une liste de tous les agents qui émettent ce champ. Pour chacun de ces agents émetteurs, la distance en cases (valeur absolue du maximum entre distance en abscisses et distance en ordonnées) entre l'agent récepteur et celui émetteur est déterminée et permet ainsi de calculer l'impact du champ émis sur l'agent récepteur. La somme de ces impacts est alors comptabilisée. S'en suit le changement d'état correspondant en fonction de cette valeur.

L'algorithme ne détermine donc pas pour chaque case de la grille la valeur des champs présents dedans. Il serait intéressant de comparer ces deux méthodes en terme d'impact sur les performances.

Le déplacement d'un agent ou sa naissance se fait sur la case adjacente à celle de l'agent d'origine dont la valeur de la somme des champs perçus par ce dernier agent est maximale. Pour cela, l'algorithme va discriminer 2 cas, le premier étant le cas où l'agent ne perçoit aucun champ, auquel cas la nouvelle case sera déterminée aléatoirement parmi les cases adjacentes libres. Le second cas correspond alors au cas où l'agent perçoit un ou plusieurs champs. (C'est dans ce cas-là qu'une indexation des champs dans chaque case de la grille s'avère intéressante).

Pour déterminer la meilleure case adjacente à un agent, l'algorithme va sommer les valeurs des champs perçus par cet agent en chacune des cases adjacentes libres et indexer les valeurs dans un dictionnaire. Parmi les cases dont la valeur est la plus grande (s'il y en a plusieurs), un tirage aléatoire renverra la case à peupler.

Limitations

Etant donné la mauvaise optimisation du programme, il n'est pas en mesure de faire tourner facilement des simulations sur une grille de taille allant au-delà de 50x50. Cela vient aussi en partie de l'utilitaire graphique (Tkinter) au niveau du rafraîchissement de la grille (Canvas).

De plus le programme ne traite pas encore de la bonne manière les agents de types « animal » et « vegetal » ce qui a pour conséquence d'induire la simulation en erreur en surchargeant la grille et donc l'exécution du programme.

Une manière d'observer la vitesse-même de calcul du programme est d'utiliser le petit utilitaire sous Kivy fonctionnant à l'aide de traitement de textures très rapides.

Bilan

Encore un gros travail d'optimisation reste à faire concernant le traitement des champs émis par les agents ainsi que par rapport à l'implémentation des agents minéraux et végétaux. Ceci est globalement dû à une mauvaise compréhension du sujet, dont notamment une mauvaise compréhension du changement d'état d'un agent minéral ou végétal (cf. `Agent.merge()`).

Les attributs TRACE et END n'ont pas non plus été implémentés par manque de temps et de compréhension du sujet.

Une finalisation de l'application sous Kivy sera aussi de mise une fois s'être assuré du bon fonctionnement du noyau du programme.