RAPPORT FINAL – ARE DYNAMICS
Anthill, création et développement d'une fourmilière

Ariana CARNIELLI, Cécile GIANG, Corneille KALUNGA

15 avril 2017

Résumé

Le projet que nous présentons ci-dessous s'est effectué dans le cadre de l'UE « ARE Dynamics ». Ce projet devant s'appuyer sur le modèle des automates cellulaires de Schelling que nous avions déjà étudié en TD, nous avons choisi de nous intéresser à la création et à l'expansion d'une fourmilière virtuelle. L'étude de ce système dynamique se rapproche beaucoup de ce modèle, la construction de tunnels et le déplacement de fourmis dépendant en grande partie de son voisinage. Les codes constituant notre programme devant être écrits en langage Python et postés chaque semaine sur un blog, nous avons choisi d'utiliser Github pour publier notre projet : `https://github.com/Sawken/Anthill`. Nous avons en particulier réussi à créer un programme qui simule la création d'une fourmilière et l'expansion de ses tunnels sur un modèle réaliste, ainsi que le comportement des fourmis qui tendent à choisir le chemin le plus court vers une source de nourriture.

Abstract

The following project was completed as part of a group project with the aim of validating the « ARE Dynamics » course unit. Since this team work should be based on the traditional model of Schelling cellular automata we studied beforehand, we chose to work on the creation and expansion of a virtual anthill. The deep relation between construction of tunnels, travel of an ant and its neighbourhood, makes the study of this dynamic system closely linked to the former model. Since we had to encode our computer program in Python language and post our work each week, we decided to use the website Github in order to publish our project : `https://github.com/Sawken/Anthill`. We eventually managed to create a Python program that is able to simulate the creation of a virtual anthill as well as its growth through the expansion of tunnels based on a realistic model. We also succeeded in modelling the behaviour of ants which seek to choose the shortest way to food sources.

1 Introduction

Écrit par : Cécile et Corneille. Relu par : Ariana.

Le groupe de travail était originalement constitué de quatre élèves : Ariana CARNIELLI, Cécile GIANG, Sofiane HASNAOUI et Corneille KALUNGA. Notre projet informatique ANTHILL avait initialement pour objectif d'étudier l'expansion d'une fourmilière en fonction de quelques paramètres environnementaux (proximité de nourriture, humidité, température...) et comportementaux ou biologiques (taux de natalité, taux de mortalité, temps de passage d'une phase de vie à une autre...) pouvant être modifiés par l'utilisateur. Pour y procéder, nous avons cherché à coder un programme capable de générer un environnement de vie pour les fourmis, ainsi que de faire évoluer de manière aléatoire la fourmilière en permettant la création et la destruction de tunnels en fonction de l'évolution de la population de fourmis.

Nous avions alors l'intention de passer à la partie « Choix du chemin le plus court », mais avons été largement ralentis du fait de l'absence d'un membre de notre groupe, Sofiane HASNAOUI, durant les trois dernières séances. Il nous a enfin informé qu'il ne viendrait plus en cours et ne participerait plus aux séances d'ARE. Ce départ a affecté le rythme de travail de notre groupe, qui codait de manière moins rapide ; et nous avons été contraints à abandonner certaines idées de codes que nous comptions

mettre en place, telles que la dépendance entre l'évolution des populations de fourmis avec le climat extérieur.

Notre rapport final présente le travail fourni par notre groupe durant les six semaines de travail, séances ainsi qu'à la maison. Ce projet s'est découpé en deux grandes parties : l'expansion de la fourmilière via des tunnels en fonction de l'évolution de la population de fourmis, ainsi que le choix du chemin le plus court vers une source de nourriture par une colonie de fourmis ouvrières.

Ce rapport est organisé de la façon suivante. Dans la Section 2 nous présentons quelques caractéristiques d'une fourmilière typique que nous avons cherché à reproduire dans ce projet. La Section 3 donne les idées générales utilisées dans notre code avec des commentaires sur les questions les plus intéressantes auxquelles nous avons fait face. Nous présentons ensuite dans la Section 4 les résultats obtenus avec nos simulations et analysons leur dépendance par rapport à différents paramètres de la simulation. Une synthèse des conclusions obtenues est présentée dans la Section 5.

2 Présentation de la thématique

Écrit par : Cécile et Corneille. Relu par : Ariana.

Le sujet auquel nous avons décidé de nous intéresser est celui de la fourmilière, et plus précisément la croissance de la population de fourmis, l'évolution spatiale de la fourmilière, ainsi que le déplacement des fourmis à son extérieur pour chercher de la nourriture. Ce sujet est intéressant à étudier pour plusieurs raisons : tout d'abord il s'inscrit bien dans le thème des automates cellulaires, puisque prenant en compte le voisinage de chaque individu (chaque fourmi) pour l'évolution de la fourmilière. De plus, nous nous sommes basés tout au long de notre projet sur des phénomènes et des chiffres réels, afin de nous rapprocher au mieux de la réalité.

Pour bien simuler une fourmilière nous avons d'abord recherché des informations sur les fourmis et leur comportement. Cette recherche a été utilisée comme base pour un ensemble de caractéristiques que nous voulons retrouver dans notre simulation.

2.1 Description d'une fourmilière

La vie et le comportement des fourmis a été le sujet de nombreuses recherches [2, 3]. Les fourmis se sont adaptées à presque tous les milieux terrestres et sous-terrains (on en a trouvé jusqu'au fond d'une grotte de 22km de long en Asie du Sud-Est). Elles n'existent cependant pas dans les milieux aquatiques et dans les zones polaires ou glacières permanentes. Une fourmilière se structure autour d'une reine qui est responsable pour la ponte des œufs. À la création de la fourmilière, c'est elle qui s'occupe des larves, mais ce rôle est ensuite délégué aux fourmis ouvrières. Une reine pond entre 18 et 25 œufs par jours tout au long de sa vie. Une colonie très conséquente peut tolérer plus d'une reine, qui habitent assez éloignées pour ne pas se croiser.

En général la reine décide de donner naissance à une portée d'œufs féconds, l'individu est alors diploïde et donnera une femelle, sinon il sera haploïde et donnera un mâle. Chaque individu passe par 4 étapes de développement : œuf, larve, nymphe et adulte, prenant environ 2 semaines pour se développer d'une étape à la suivante. La plupart des fourmis femelles grandissent pour devenir des femelles aptères (i.e. non-ailées) et stériles, dites ouvrières. Les autres (femelles non-stériles et mâles) sont ailés. Périodiquement, des essaims de fourmis mâles et femelles ailés quittent la colonie afin de se reproduire. On appelle ce phénomène le « vol nuptial ». Suite à cela, les mâles meurent tandis que les reines fécondes partent fonder de nouvelles colonies.

La longévité d'une fourmi dépend de son type : Les mâles ont une vie très brève, ils meurent juste après s'être reproduits. La fourmi ouvrière peut vivre entre 3 semaines et 1 an. La reine vit en général entre 1 et 3 ans. Chez certaines espèces, sa longévité est extrêmement longue et peut aller jusqu'à une quinzaine à une vingtaine d'années.

Les premiers jours de sa vie, une fourmi ouvrière s'occupe de la reine et de ses petits. Ensuite elle participe à la construction et au maintien du nid, puis à son approvisionnement et enfin à sa défense. Les ouvrières se regroupent selon l'activité commune qu'elles ont à un stade de leur vie.

Pour la recherche de nourriture, une fourmi dite « éclaireuse », qui découvre une source de nourriture, rentre dans sa colonie en laissant sur son passage une trace chimique. Cette trace stimule d'autres fourmis ouvrières, les attirant à l'extérieur de la colonie afin de la suivre et de trouver à leur tour la source de nourriture. Elles rentrent ensuite chez elles, laissant sur leur passage une trace chimique qui renforce la précédente, et le même processus se reproduit. Il est intéressant de noter que ce comportement de recherche de nourriture par les fourmis a inspiré des algorithmes de recherche de chemin plus court dans un graphe afin de donner une réponse au « problème du voyageur du commerce », dont le but est de trouver le chemin le plus court permettant de relier un ensemble de villes [1].

3 Modélisation et implémentation

Écrit par : Ariana et Cécile. Relu par : Corneille.

3.1 Caractéristiques retenues pour notre simulation

D'après les éléments précédents nous avons décidé de simuler la croissance de la population de fourmis, l'évolution temporelle des tunnels d'une fourmilière et la recherche de nourriture par une fourmi.

Pour la croissance de la population, notre modèle prendra en compte les caractéristiques suivantes :
— La fourmilière commence avec une reine qui pond des œufs.
— À chaque jour la reine pond entre 18 et 25 œufs.
— Une fourmi prend 15 jours pour passer d'une étape de son développement à la suivante.
— Les ouvrières meurent après au maximum 1 an.
— La reine ne meurt pas (car elle peut vivre plusieurs années, au-delà de l'horizon de temps choisi pour nos simulations).
— Puisque une jeune fourmi ouvrière s'occupe de la reine et des ses petits, nous considérons que leur quantité influence le taux de naissance.

Pour l'évolution temporelle des tunnels de la fourmilière, nous prenons en compte :
— La quantité totale de fourmis ouvrières, puisque celles-ci participent à la construction et au maintien du nid.
— L'emplacement initial de la reine, c'est-à-dire, le lieu où la fourmilière a commencé.

Pour la recherche de nourriture par une fourmi, nous considérons :
— Le passage d'une fourmi ayant trouvé de la nourriture laisse une trace chimique.
— Cette trace chimique attire les fourmis suivantes.

3.2 Croissance de la population de fourmis

Implémentation : Cette partie a été commencée par Ariana et Sofiane. Après le départ de Sofiane, Cécile et Corneille ont aidé.

Nous avons d'abord élaboré un programme capable de simuler l'évolution temporelle de la population d'une fourmilière ; pour cela, nous avons mis en place des fonctions simulant séparément la naissance et la mortalité d'une fourmi. Nous avons crée quatre vecteurs, `oeuf`, `larve`, `nymphe` et `ouvriere`, contenant chacun le nombre total de fourmis dans l'étape respective de développement. L'élément `i` d'un vecteur contient le nombre de fourmis qui sont dans l'étape correspondante depuis `i` jours. Par exemple, `ouvriere[5]` contient le nombre de fourmis qui sont adultes depuis 5 jours. Nous avons créé des vecteurs (`numpy.array`) et pas de listes pour simplifier les opérations algébriques (somme élément par élément).

Notre programme contient 3 fonctions : Une fonction `naissance` qui crée un nombre aléatoire d'œufs entre 18 et 25. Une fonction `mort` qui, à chaque fois, retourne une liste avec une quantité aléatoire de fourmis mortes selon leur âge et enfin une fonction `croissance_fourmis` qui fait le vieillissement d'un jour de chaque fourmi et appelle les deux dernières. Nous avons choisi d'augmenter exponentiellement la probabilité de mort avec l'âge de la fourmi pour prendre en considération non seulement le vieillissement des fourmis mais aussi le fait que leur rôle change en fonction de leur âge.

3.3 Expansion de la fourmilière en tunnels

Implémentation : Cette partie a été commencée par les 4 membres de l'équipe, puis après avoir eu l'idée du « Chemin le plus court », elle a été confiée à Ariana et Sofiane. Après le départ de Sofiane, les trois derniers membres (Cécile, Ariana et Corneille) ont tous contribué.

Nous sommes ensuite passés à la programmation de l'évolution spatiale de la fourmilière à proprement parler. Dans ce programme-ci, notre environnement spatial sera représenté par une matrice `map`, initialisée avec des 1 (représentant la terre pas encore creusée). Notre simulation appelle d'abord la fonction `set_queen`, qui génère une reine placée aléatoirement sur la matrice, représentée par l'entier 2, et crée un espace autour d'elle rempli de 0, qui représentent des tunnels. Nous créons aussi une matrice `map_dist`, de même taille de `map`, initialisée avec des zéros, et l'appel de la fonction `set_queen` la remplit avec les distances euclidiennes de chaque élément par rapport a la reine. Nous voulons utiliser cette matrice pour choisir dans quel endroit creuser un tunnel.

L'étape suivante est la création d'une fonction de création de tunnels que nous avons appelée `croissance_tun`. Cette fonction parcourt la matrice et, pour chaque tunnel trouvé, elle décide de façon aléatoire si un nouveau tunnel sera crée à partir de celui-ci ou non. Ce choix aléatoire prend en compte la distance de la reine, la taille de la fourmilière (calculée à l'aide d'une autre fonction) et la proportion de tunnels déjà crées dans un voisinage, dont la taille est définie par une variable globale `voisinage_exte`. Un tunnel plus loin de la reine et avec une proportion de tunnels voisins petite a plus de chances de générer un nouveau tunnel à coté de lui. Cela évite de tomber dans une configuration où toutes les cases finissent par loger un tunnel et n'aboutir qu'à un agrandissement circulaire de la fourmilière. Lorsque le programme décide de creuser un tunnel la direction de celui-ci est aussi choisi de façon aléatoire en prenant en compte la distance de la reine : nous avons plus de chance de creuser un tunnel que s'éloigne de la reine. Nous avons alors bien une création de tunnels longs et fins.

Cette fonction `croissance_tun` est appelée a l'intérieur d'une fonction `dynamics` dont le rôle est d'implémenter le développement temporelle de la fourmilière. Nous avons défini que chaque tour de boucle correspond à un jour, temps suffisante pour augmenter de façon visible les tunnels. Dans chaque tour de boucle, la fonction `dynamics` fait l'évolution de la population de fourmis décrite dans la Section 3.2 et, en prenant en considération le nombre de ouvrières et la taille actuelle de la fourmilière, décide si il y aura une création ou destruction de tunnels. Ce point est intéressant car si la population de fourmis croît, plus d'espace sera nécessaire et donc plus de tunnels. Mais si, au contraire, la population de fourmis diminue, l'espace à disposition sera trop étendu et donc une partie de celui-ci sera abandonné par les fourmis et s'effondrera.

Pour implémenter la décroissance de tunnels nous avons crée une nouvelle fonction `decroissance_tun` qui prend en argument le nombre de tunnels en excès `quant_tun` calculé dans la fonction `dynamics`. Dans un premier temps nous avons décidé de détruire les `quant_tun` tunnels les plus éloignés de la reine, ce qui a eu pour effet que, au bout d'un grand nombre d'itérations, nous arrivons à un état stable où les tunnels forment toujours une image circulaire. Ce caractère déterministe est en fait dû à la prise en compte uniquement de la distance euclidienne de chaque tunnel.

En ajoutant un facteur aléatoire dans la destruction de tunnels nous avons réussi à surmonter ce problème. Ce facteur aléatoire consiste à établir la liste des `quant_tun` tunnels les plus éloignés de la reine et en choisir un. La fonction détruit alors un carré autour de celui-ci contenant au moins `quant_tun` tunnels. Par contre, cette solution peut entraîner la création d'une fourmilière non connexe, ce qui n'arrive pas dans le monde réel.

Pour garder la connexité de la fourmilière nous avons décidé de détruire également tous les tunnels non connectés a la reine à la fin d'un jour. Il nous fallait donc identifier les tunnels connectés à la reine. Cela a été fait à l'aide de la théorie des graphes. Nous avons considéré notre matrice comme un graphe dont les nœuds sont les tunnels et 2 nœuds sont reliés par une arête si les tunnels sont voisins. Nous avons alors utilisé un algorithme classique de parcours de graphe, le *depth first search*, implémenté de façon récursive, pour construire l'ensemble de tunnels connectés à la reine. Le tunnels ne se trouvant pas dans cet ensemble sont alors détruits.

Avec ces implémentations, nous avons un code fonctionnel et suffisamment proche de la réalité, comme on peut voir dans les simulations de la Section 4.2.

3.4 Choix du chemin le plus court

Implémentation : Cette partie a été commencée par Cécile et Corneille. Après avoir fini l'implémentation de la Section 3.3, Ariana nous a également rejoints.

Dans cette dernière partie, nous nous intéressons à un phénomène scientifique depuis longtemps observé et assez remarquable : on sait que les fourmis, lorsqu'elles vivent en colonie, sont capables de trouver le chemin le plus court pour accéder à une source de nourriture. En réalité, une première fourmi dite « éclaireuse » trouve la source, puis revient à la fourmilière en laissant sur son passage une trace chimique qui attire les autres fourmis. On comprendra ainsi que ces dernières se déplacent elles-mêmes jusque-là en laissant une trace à leur tour, et plus le chemin emprunté est court, plus elles sont de retour rapidement et plus les traces chimiques du chemin court sont renforcés. Ce phénomène est traduit en informatique sous le nom de « Ant Algorithm », et répondant au problème du voyageur de commerce [1].

Avec nos moyens, nous avons essayé de modéliser ce phénomène à notre tour, en nous inspirant de [1]. Nous avons tout d'abord créé un environnement de vie dans lequel se situeront la fourmilière et la source de nourriture, appelé `espace`, qui est une matrice nulle carrée de dimension `dim`. À l'aide de fonctions que nous avons créées, nous pouvons générer une sortie de la fourmilière notée -1 (réduite à la case de coordonnées `(dim,dim)`) ainsi qu'une source de nourriture dont la valeur est très grande, afin d'attirer les fourmis. Cette source se situe dans un endroit aléatoire de l'espace, situé dans la partie supérieure gauche de la matrice. Une première fourmi, l'éclaireuse, est générée à un voisinage immédiat et aléatoire de la fourmilière.

La première chose à faire était alors de créer un programme qui la fasse se déplacer jusqu'à ce qu'elle trouve la source de nourriture. Nous avons donc d'abord naïvement donné à la fourmi des probabilités de se déplacer dans chaque case voisine à peu près identique, avec plus de chance de se déplacer vers le haut (la fourmilière se situant dans la partie inférieure de l'espace). Nous avons remarqué très vite que ce choix n'était pas convenable car elle revenait souvent sur ses pas et ne trouvait presque jamais la source de nourriture.

Nous avons compris qu'il fallait que la fourmi « sente » la proximité d'une source de nourriture, et avons donc créé un environnement attractif autour de cette source. Ici, il s'agissait de mettre de valeurs (nous avons choisi 3) autour de la source, et faire en sorte que la probabilité de déplacement d'une fourmi vers une case soit proportionnelle à la valeur contenue dans cette case (phénomène d'attraction). De plus, nous avons créé des fonctions telles que la fourmi enregistre chaque position qu'elle a prise, et ne puisse pas revenir sur ses pas.

Ainsi, soit elle finit par rester bloquée car prise en piège dans des positions déjà empruntées et disparaît, soit le nombre d'itérations passé en paramètre n'est pas suffisant pour qu'elle trouve cette source et elle finit par mourir, soit elle trouve la source. Lorsqu'elle trouve la source, nous considérons qu'elle reprend le même chemin pour rentrer à la fourmilière et laisse une trace chimique qui attirera les autres fourmis et que nous représentons par une augmentation des valeurs de la matrice dans ce chemin.

Il s'agit ensuite pour les autres fourmis de suivre à leur tour la trace chimique jusqu'à la source de nourriture. Nous avons rencontré un autre problème : malgré la différence déjà significative de valeurs entre les positions empruntées et celles qui ne le sont pas, les fourmis ont parfois tendance à prendre des chemins non empruntés. Nous avons compris que malgré la différence de valeurs, la probabilité de chaque position ne différait que très peu et nous avons alors pensé à incrémenter la probabilité de manière exponentielle. Avec cette implémentation, nous avons un code fonctionnel, comme on peut voir dans les simulations de la Section 4.3.

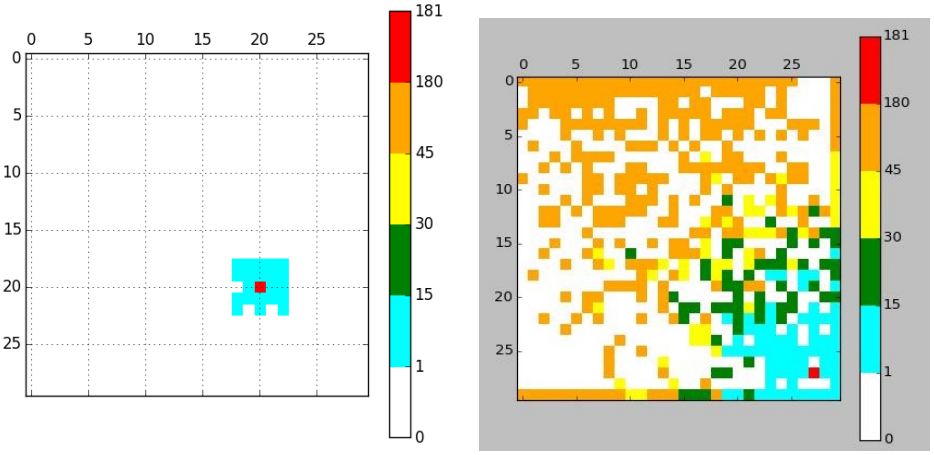
4 Résultats et analyse

Écrit par : Ariana. Relu par : Cécile et Corneille.

4.1 Croissance de la population de fourmis

Implémentation : Cette partie a été faite par Cécile et Ariana.

Avant de commencer de coder notre programme pour l'évolution spatiale de la fourmilière à proprement parler, nous avons tout d'abord codé un petit programme intermédiaire, permettant de visualiser la création et l'expansion d'une population de fourmis à partir d'une fourmi reine. Ce programme n'aura pas de grande importance par la suite, mais nous l'avons quand même écrit afin de nous échauffer avant la « vraie programmation ». De plus, certaines fonctions que nous avons créées dans cette partie nous seront utiles pour la suite car elles sont réutilisables. Dans la figure obtenue grâce à la bibliothèque `Matplotplib`, chaque couleur correspondra à une étape de la vie de la fourmi grâce à l'utilisation de la fonction `colorlab`. Nous basant sur les chiffres réels, chaque fourmi se voit attribuer une valeur correspondant à son âge en jours. Une fourmi vit au plus 180 jours (pour simplifier la simulation) ; et seule la reine (en rouge) peut vivre plus longtemps. Les résultats de cette simulation sont présentés sur la Figure 1.



(a) (b)

FIGURE 1 – (a) Reine (en rouge) et première génération de fourmis sous forme d'œufs (cyan), (b) État de la population au bout de 180 jours.

Nous avons également fait une autre visualisation, plus simple, de la distribution des fourmis selon leur âge, représenté sur la Figure 2. On y voit l'effet de l'augmentation exponentielle de la probabilité de mort avec l'âge des fourmis.

Pour vérifier comment l'évolution de la population fonctionne, nous avons crée 2 autres simulations. Dans la première, nous avons tracé l'évolution de la quantité totale de fourmis au cours du temps, son résultat étant donné dans la Figure 3(a). On peut y voir que, au début, nous avons une croissance presque linéaire du nombre de fourmis, mais, lorsque la population vieillit, le taux de mort de fourmis les plus âgées entraîne une diminution du taux de croissance de la population jusqu'à ce qu'il se stabilise autour d'un valeur. Dans la deuxième simulation nous nous intéressons à l'effet que peut avoir une augmentation de la probabilité de mort des fourmis sur l'âge de la fourmi la plus âgée au bout d'un grand nombre de jours. Nous rappelons que cette probabilité dépend de l'âge de la fourmi et nous l'avons choisie de façon à croître exponentiellement avec l'âge. Le paramètre que nous changeons est la valeur de cette probabilité lorsqu'une fourmi a 365 jours. Le résultat de cette simulation est montré dans la Figure 3(b). On peut y voir que, comme on s'y attend, l'âge maximale diminue avec l'augmentation de la probabilité de mort, mais reste néanmoins assez aléatoire (comme nous pouvons voir par les oscillations dans la figure).

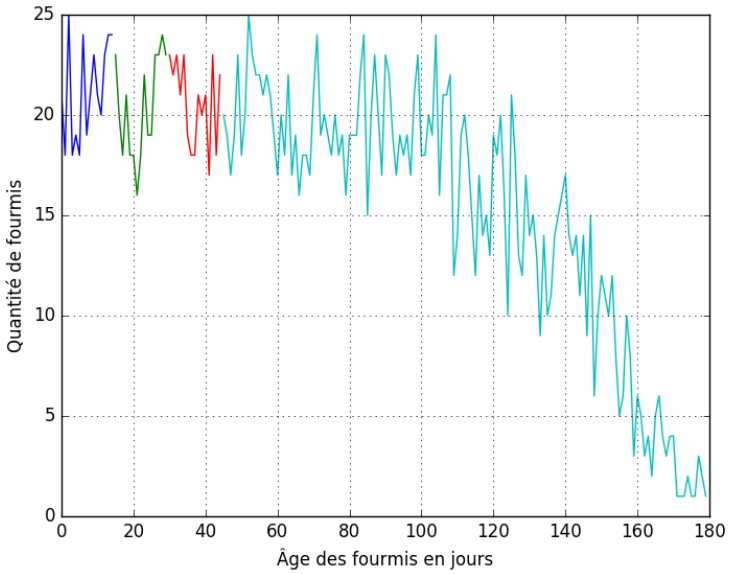
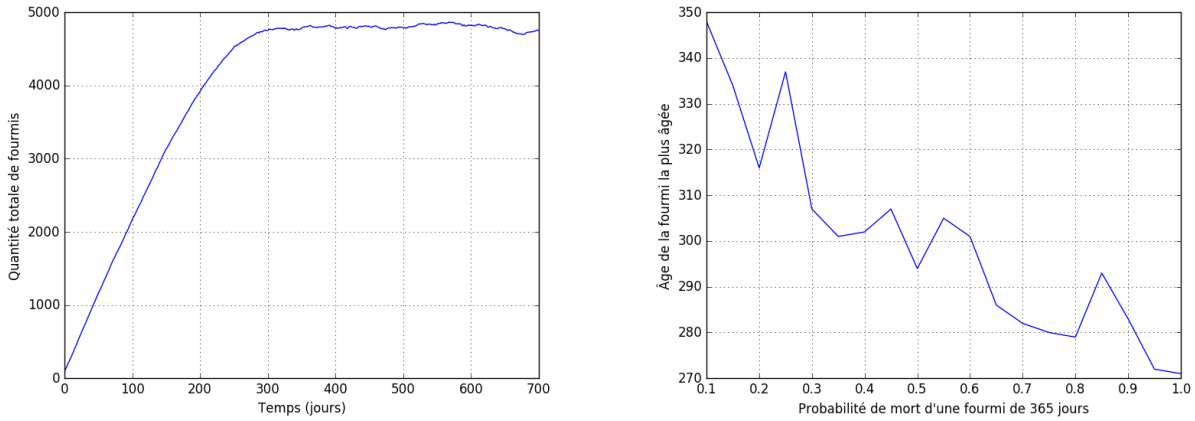


FIGURE 2 – Quantité de fourmis selon leur âge en jours. Les quatre couleurs représentent les quatre étapes de développement.



(a) (b)

FIGURE 3 – (a) Quantité totale de fourmis en fonction du temps, (b) Âge de la fourmi la plus âgée au bout de 500 jours.

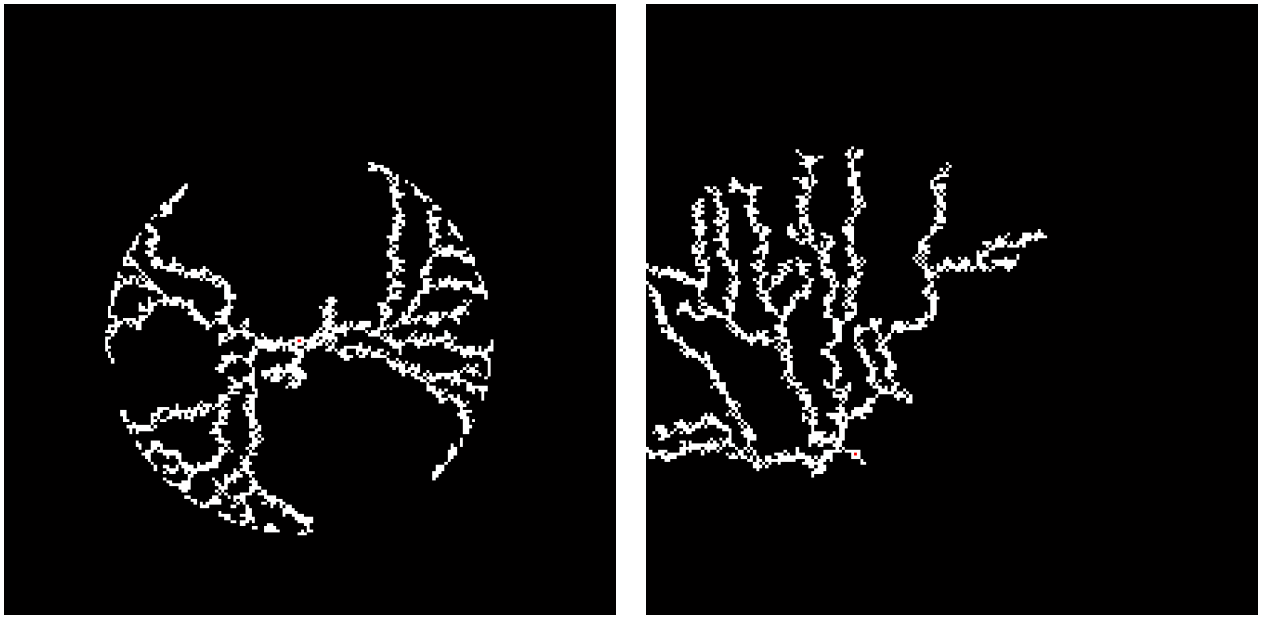
4.2 Expansion de la fourmilière en tunnels

Implémentation : Cette partie a été faite par les 3 membres du groupe.

Nous avons implémenté le code décrit à la Section 3.3. Avec la version préliminaire de la fonction `decroissance_tun`, qui élimine les tunnels les plus éloignés de la reine dans l'ordre, nous obtenons, au bout de 500 jours (i.e. 500 tours de boucle) l'image représentée sur la Figure 4(a). Nous y voyons le contour circulaire de la fourmilière qui provient du choix des tunnels à écraser comme expliqué précédemment. La Figure 4(b) montre le résultat des modifications apportées à la fonction `decroissance_tun`, où nous pouvons voir que, comme attendu, le choix aléatoire des tunnels à écraser rend la forme de la fourmilière plus naturelle et la suppression des tunnels non connectés à la reine garde bien la connexité de la fourmilière.

Pour comprendre la croissance de la fourmilière, nous avons également simulé la croissance, au cours du temps, de son rayon, c'est-à-dire de la distance du tunnel le plus éloigné de la reine. Nous avons fait une moyenne de ce rayon sur 10 simulations de 500 jours chacune, le résultat étant montré dans la

Figure 5. Puisque nous avons pris en compte le nombre de fourmis pour déterminer la dynamique de création/destruction des tunnels de la fourmilière, il est naturel de s'attendre à ce que la croissance du rayon ressemble celle de la population des fourmis, ce que l'on peut bien voir en comparant les Figures 3(a) et 5. Nous pouvons néanmoins remarquer quelques subtilités dans le graphe du rayon : en effet, il commence a croître mais se stabilise autour d'une valeur petite, ce que l'on peut expliquer par le fait que seulement les ouvrières contribuent à creuser de tunnels et il faut attendre 45 jours pour que les œufs pondus par la reine deviennent des ouvrières. Pendant cette période ce sont uniquement les ouvrières qui ont fondé la fourmilière avec la reine qui creusent de tunnels. Ensuite, nous observons la croissance linéaire de la fourmilière, jusqu'à la stabilisation du nombre de fourmis, qui a lieu autour de 300 jours.



(a) (b)

FIGURE 4 – Fourmilière au bout de 500 tours de boucle avec (a) La version préliminaire de la fonction `decroissance_tun` (b) La version finale de `decroissance_tun`.

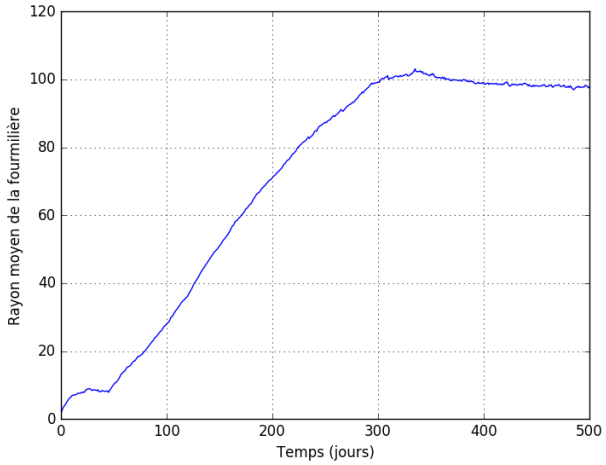
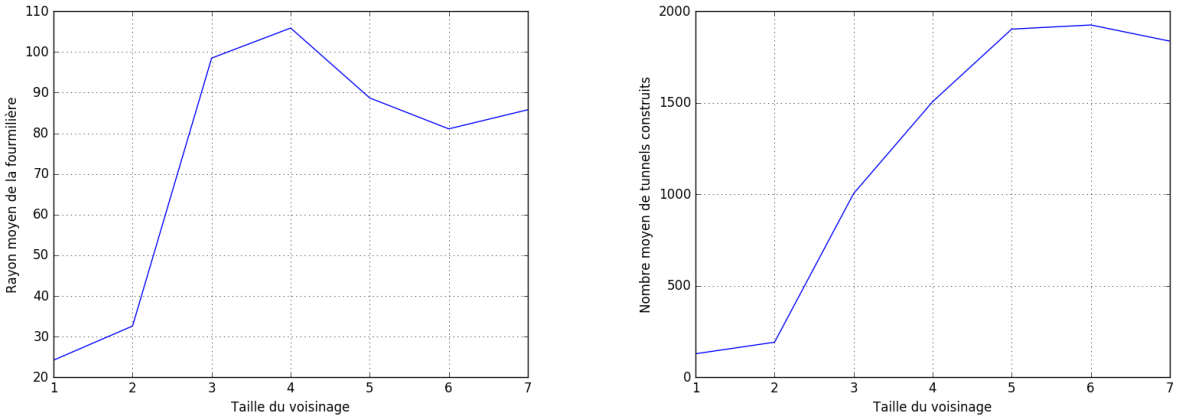


FIGURE 5 – Rayon moyen de la fourmilière par rapport au temps.

Nous avons aussi étudié comment varie le rayon moyen et le nombre totale de tunnels construits en fonction de la taille du voisinage considéré pour la création des tunnels, `voisinage_exte`. Pour cela, nous avons fait varier la taille du voisinage de 1 à 7, en pas de 1, et pour chaque taille de voisinage

nous avons fait 10 simulations regardant le rayon et le nombre de tunnels creusés au bout de 500 jours. La moyenne de ces résultats sur les 10 simulations est représenté dans la Figure 6.

On peut y voir que pour des tailles de voisinage petites ces deux quantités reste petites, c'est-à-dire, la fourmilière ne croit pas beaucoup. À partir d'un voisinage de taille 3, la fourmilière croit beaucoup plus. Il est intéressant à noter que le nombre de tunnels creusés augmente avec la taille du voisinage mais le rayon de la fourmilière décroit pour de voisinages plus grands que 4. En effet, pour de grands voisinages, les tunnels construits sont plus larges, et comme le nombre total de tunnels est limité par le nombre de fourmis, cela conduit à une fourmilière plus dense et avec un rayon plus petit. Pour les voisinages plus grands que 7 notre code présente une limitation : chaque tunnel possède plusieurs voisins qui sont aussi de tunnels et, comme notre fonction d'identification de connexité est récursive, nous avons un nombre excessivement grand d'appels à cette fonction et un dépassement de pile se produit.

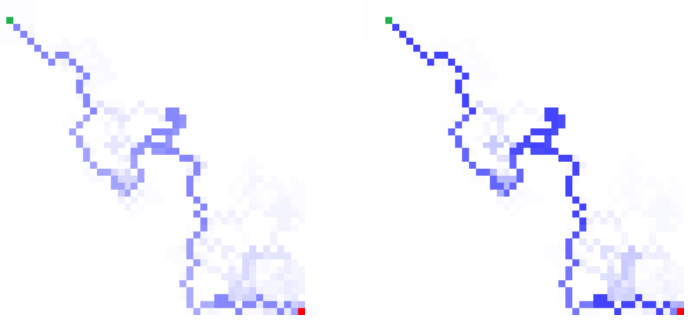


(a) (b)

FIGURE 6 – (a) Rayon moyen de la fourmilière et (b) quantité de tunnels, en fonction de la taille du voisinage.

4.3 Choix du chemin le plus court

Implémentation : Cette partie a été faite par Cécile et Corneille. La visualisation en image a été fait par Ariana.



(a) (b)

FIGURE 7 – Trace chimique laisse par les fourmis au bout du passage de (a) 25 fourmis (b) 200 fourmis. Le point rouge indique la sortie de la fourmilière et le point vert la nourriture.

Nous avons simulé notre code pour le choix du chemin le plus court jusqu'à une source de nourriture. Bien que notre code n'implémente pas toutes les idées que nous avons eues, nous avons déjà un code suffisamment fonctionnel que nous permet de voir le chemin emprunté par les fourmis pour arriver à la source de nourriture. Un exemple typique de résultat obtenu est donné dans la Figure 7. Nous

pouvons y voir que le fourmis arrivent bien à trouver la source de la nourriture et la plupart d'entre elles emprunte le même chemin. La trace de ce chemin est renforcé lorsque le nombre de fourmis y ayant passé augmente. Le chemin choisi n'est pas forcement le plus court car dans notre code, par manque de temps, nous n'avons pas pu implémenter la décroissance des traces chimiques au cours du temps, ce qui serait nécessaire pour qu'un algorithme du type « Ant system » trouve bien le meilleur chemin.

5 Conclusion

Écrit par : Cécile. Relu par : Ariana.

Dans ce projet nous avons réussi à simuler plusieurs aspects d'une fourmilière : l'évolution de sa population, le creusement et effondrement des tunnels et, dans une moindre mesure, la recherche de la nourriture. Nous avons trouvé nos résultats assez satisfaisants, bien que nous n'ayons pas eu le temps de finir la partie sur la recherche de la nourriture, comme le suppose le départ d'un membre de notre groupe. Il a été intéressant de modéliser et nous rapprocher, pourtant avec le déterminisme que suppose une programmation informatique, d'un phénomène naturel et difficilement explicable. Le fait de nous concentrer sur une approche la plus proche possible de la réalité a attiré notre intérêt à des paramètres dont nous n'avions pas mesuré l'enjeu au départ, et qui jouent portant un rôle fondamental dans l'exécution du phénomène. Ainsi, de par nos erreurs ou négligences, nous avons appris à prendre en compte voire parfois découvrir des facteurs intéressants dans l'évolution d'un système dynamique, comme dans notre cas la prise en compte du voisinage le plus éloigné de la reine pour la construction de tunnels, ou encore des positions déjà prises par une fourmi qui cherche de la nourriture.

Il y a beaucoup de choses que nous voulions faire, et n'avons pas pu faute de temps. Tout d'abord pour le programme sur l'évolution spatiale de la fourmilière, nous aurions voulu faire dépendre cette expansion d'autres paramètres que la position de la reine et l'évolution de la population de fourmis. Il aurait été par exemple intéressant de prendre en compte le milieu de vie des fourmis (humide, aride, champs, etc.). Dans le programme de la recherche de la nourriture, nous aurions aimé ne pas limiter la sortie d'une seule fourmi par cycle de déplacement. Il aurait été intéressant de voir ce phénomène avec un nombre élevé de fourmis, simultanément. Il aurait également été intéressant de mettre en place la décroissance de trace chimique pour que les fourmis puissent vraiment trouver le meilleur chemin.

Nous avions également l'intention au départ de mettre en lien ces deux programmes : il s'agirait par exemple de prend en compte la quantité de nourriture trouvée dans les taux de croissance et mort de fourmis, ce qui aurait également un effet sur l'évolution spatiale de la fourmilière. Ainsi si au début les premières fourmis empruntent des chemins assez longs, les tunnels se seraient peu développé, voire détruits. Mais au bout d'un grand nombre d'itérations, les fourmis se seraient nourris plus rapidement et nous aurions pu observer une croissance rapide des tunnels, suivi éventuellement d'un retour à la destruction de tunnels lorsque la source de nourriture serait épuisée.

Ce projet d'ARE a été très enrichissant et divertissant à mener. Nous avons pu mettre en pratique notre savoir jusque-là très théorique du langage Python, et cette mise en pratique a été d'autant plus passionnante que le sujet en lui-même était un phénomène scientifique, et un projet de groupe. Toute ceci nous a permis d'entrevoir modestement comment de vrais scientifiques mettent tout en œuvre pour mener à bien leurs recherches et leurs projets, et comment l'organisation, la confiance et le dialogue en équipe sont primordiaux au bon fonctionnement et à la bonne entente du groupe.

Références

[1] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system : optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1) :29–41, 1996.

[2] L. Keller and E. Gordon. *La vie des fourmis.* Odile Jacob, 2006.

[3] W. M. Wheeler. *Ants : Their Structure, Development and Behavior.* Columbia University biological series. Columbia University Press, 1960.