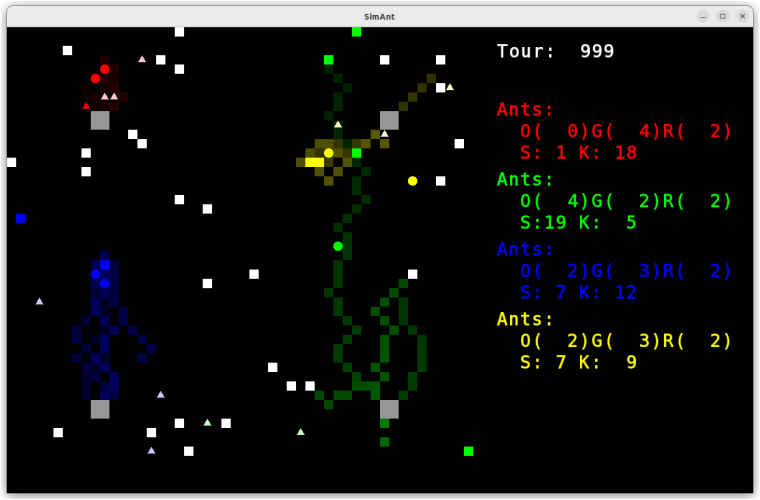
université PARIS-SACLAY

Projet de programmation

Cours de programmation modulaire

—Licence MI/IM/MNSI - Info 3—

Simulation de colonies de fourmis



Dans ce projet, nous allons simuler des *colonies de fourmis* en compétition pour ramener de la nourriture au nid. Pour repérer les sources de nourriture et le nid, les fourmis laissent sur le terrain des substances chimiques appelées *phéromones*. Elles peuvent ensuite suivre ces traces pour retrouver la nourriture.

1 Introduction

On considère une grille de 20 lignes 20 colonnes (mais qui pourra être plus grande). Une case de la grille peut être vide, ou contenir une des trois choses suivantes : une fourmi, du sucre ou un élément de nid. Une case ne peut pas contenir en même temps deux fourmis, ou bien du sucre et une fourmi (sauf si la fourmi porte le sucre), ou bien un élément de nid et une fourmi. Les fourmis doivent donc contourner le sucre et le nid, et se contourner entre elles. Une fourmi peut ramasser du sucre si elle se trouve à côté d'une case en contenant et peut poser du sucre dans le nid si elle en porte et se trouve à côté d'une case contenant un élément de nid.

Au début, les fourmis cherchent du sucre aléatoirement sur la grille. Une fois qu'une fourmi a trouvé du sucre, elle le ramène vers le nid en suivant des «phéromones de nid», qui indiquent le chemin de retour vers le nid. Pendant leur retour du tas de sucre vers le nid, les fourmis marquent leur chemin avec des «phéromones de sucre», qui vont permettre aux autres fourmis de trouver le sucre plus rapidement en suivant cette trace. Les phéromones sont des substances émises par la plupart des animaux et certains végétaux, et qui agissent comme des messagers sur des individus de la même espèce. Extrêmement actives, elles agissent en quantités infinitésimales, si bien qu'elles peuvent être détectées, ou même transportées, à plusieurs kilomètres.

Pour simplifier, nous supposerons que les «phéromones de nid» sont présentes en permanence et installent un gradient dirigé vers le nid. Cela signifie que l'intensité des phéromones augmente au fur et à mesure que l'on se rapproche du nid. Pour se rapprocher du nid, il suffit donc de se déplacer vers

une case de plus forte intensité en phéromones. Déposées lors du passage des fourmis portant du sucre, les «phéromones de sucre» s'évaporent. Ainsi, les fourmis ne s'obstineront pas à suivre un chemin de phéromones de sucre vers un tas de sucre qui a été complètement vidé.

Les fourmis sont réparties dans plusieurs colonies différentes, chacune ayant son propre nid et ses propres phéromones de sucre et de nid. Si deux fourmis vivantes se rencontrent, c'est-à-dire si elles sont dans deux cases adjacentes, et qu'elles proviennent de deux colonies différentes, elles doivent se battre : la fourmi qui perd meurt et disparaît, et son sucre, si elle en porte un, disparaît. Les fourmis peuvent aussi naître : lorsqu'un nid continent assez de sucre, une nouvelle fourmi naît.

Pour un comportement plus réaliste des fourmis, on se propose d'ajouter le mode d'organisation sociale des fourmis appelé "eusocialité" qui divise le groupe d'individus en castes d'individus fertiles et non-fertiles (on dit que les fourmis pratique le "polyéthisme" de caste.). Dans cette version de la simulation, on propose d'ajouter trois castes différentes : les fourmis ouvrières, les fourmis guerrières et les fourmis reproductrices.

1.1 Comportement des fourmis

Pour décrire le comportement des fourmis, nous utiliserons les prédicats (fonctions booléennes) dont on donne une liste pas forcément exhaustive ci-dessous :

— Prédicats sur une **fourmi** f :

`f.chercheSucre()`	f ne porte pas de sucre
`f.rentreNid()`	f porte du sucre
`f.estVivante()`	f est vivante

— Prédicats sur une **place** (une case) p :

`p.contientSucre()`	p contient du sucre
`p.contientNid()`	p contient un élément de nid
`p.contientNid(colonie)`	p contient un élément de nid de la colonie donnée
`p.contientFourmi()`	p contient une fourmi
`p.estVide()`	p ne contient ni sucre, ni fourmi, ni élément de nid
`p.estSurUnePiste(colonie)`	l'intensité des phéromones de sucre d'une colonie donnée est non nulle sur p

— Prédicats sur une place p_1 et une place voisine p_2 :

`plusProcheNid(`p_1`, `p_2`)`	l'intensité des phéromones de nid sur p_1 est plus importante que celle sur p_2
`plusLoinNid(`p_1`, `p_2`)`	l'intensité des phéromones de nid sur p_1 est moins importante que celle sur p_2

Note : Pour une place donnée, plusieurs prédicats peuvent être vrais, par exemple : `estSurUnePiste(`p`)` et `estVide(`p`)`.

Le principe général du comportement des fourmis est le suivant. Elles cherchent du sucre en bougeant aléatoirement. Quand elles trouvent de la phéromone de sucre, elles s'éloignent le plus vite possible du nid, tout en restant sur la piste tracée par la phéromone de sucre. Quand elles sont à côté du sucre, elles chargent une partie du sucre, puis se rapprochent le plus vite possible de leur nid tout en posant des phéromones de sucre propre à leur colonie sur les cases empruntées. Lors des rencontres avec les autres fourmis, elles se battent et peuvent mourir. Les fourmis naissent lorsque le nid contient suffisamment de sucre (par exemple plus de dix unités).

Plus formellement, le comportement d'une fourmi est dicté par les règles énoncées ci-dessous par **ordre de priorité décroissante**, c'est-à-dire que l'on fait uniquement la première action dont les conditions sont vérifiées.

Règles :

Soient f une fourmi, p_1 la case qu'elle occupe et p_2 une case adjacente à p_1.

1. Si `estVivante(`f`)`, `contientFourmi(`p_2`)` et p_2 contient une fourmi g d'une colonie différente, alors f tue g.
2. Si `chercheSucre(`f`)` et `contientSucre(`p_2`)`, alors f charge du sucre et pose une phéromone de sucre de sa colonie sur p_1.
3. Si `rentreNid(`f`)` et `contientNid(`p_2`)`, alors f pose son sucre.
4. Si `rentreNid(`f`)` et `estVide(`p_2`)` et `plusProcheNid(`p_2, p_1`)`, alors f se déplace sur p_2 et pose une phéromone de sucre de sa colonie sur p_2.
5. Si `chercheSucre(`f`)` et `estSurUnePiste(`p_1`)` et `estVide(`p_2`)` et `plusLoinNid(`p_2, p_1`)` et `estSurUnePiste(`p_2`)`, alors f se déplace sur p_2.
6. Si `chercheSucre(`f`)` et `estSurUnePiste(`p_2`)` et `estVide(`p_2`)`, alors f se déplace sur p_2.
7. Si `chercheSucre(`f`)` et `estVide(`p_2`)`, alors f se déplace sur p_2.

1.1.1 polyéthisme de caste et eusocialité

L'ajout du polyéthisme de caste modifie les règles de bases du comportement des fourmis. Elles ne sont plus régit par un seul ensemble de règles générales, mais chaque caste aura son propre ensemble de règles. Une même règle peut être utilisé par plusieurs castes, mais l'ensemble des règles d'une caste permet de donner un comportement différent à chaque caste. Voici les règles des différentes castes que l'on souhaite implémenter.

Fourmis Ouvrières : Les fourmis ouvrières ont l'unique tâche d'approvisionner le nid de la fourmilière en sucre. Le comportement de prédation leurs est enlevés ce qui les empêches de tuer d'autre fourmis. En termes de comportement, les fourmis ouvrières garde le même ensemble de règles que les fourmis de base hormis la règle 1 qui leur est retiré. On garde uniquement les règles 2 à 7.

Fourmis Guerrières : Les fourmis guerrières défendent la colonie de fourmis en attaquant les fourmis des autres nids. Les fourmis guerrières sont les seuls qui suivent les phéromones de sucres des autres fourmis pour tenter d'intercepter les fourmis des autres nids et les tuées. Les règles des fourmis guerrières sont les suivantes :

Règles Guerrières :

Soient f une fourmi, $f_{colonie}$ est la colonie auxquels appartient f,*colonie* désigne une des colonies existantes, p_1 la case qu'elle occupe et p_2 une case adjacente à p_1.

1. Si `estVivante(`f`)`, `contientFourmi(`p_2`)` et p_2 contient une fourmi g d'une colonie différente, alors f tue g.
2. Si p_2`.estSurUnePiste(`$colonie \neq f_{colonie}$`)` et `estVide(`p_2`)`, alors f se déplace sur p_2.

	Si `estVide`(p_2),	
3.	alors f se déplace sur p_2.	règle équivalente à la règle 7, mais sans la nécessité que f doivent porter du sucre

Fourmis Reproductrice : Attention ! : L'ajout de fourmis reproductrices peut être plus ou moins difficile en fonction des choix de conception fait au début du projet. Pensez bien à ajouter votre travail sur git avant de vous lancer dans cet ajout. Vous risqueriez d'avoir à modifier une partie de votre projet qui peut rendre non fonctionnelle une simulation qui l'était précédemment. Ça peut être le bon moment si vous souhaitez apprendre à faire des branches avec git (voir tuto : git-branch-openclassrooms).

L'ajout des fourmis reproductrices permet de rendre compte du mécanisme d'eusocialité dans notre simulation de colonie de fourmis. L'eusocialité se caractérise par des individus fertiles et non-fertiles. Dans notre cas, les fourmis ouvrières et les fourmis guerrières sont non-fertiles et seuls les fourmis reproductrices sont fertiles.

Dans le cas d'une simulation avec les fourmis reproductrices, vous devrez désactiver la naissance automatique de nouvelles fourmis lorsque la quantité de sucre est suffisante dans le nid. À la place, on souhaite mettre en place un mécanisme qui permet l'émergence d'une nouvelle fourmi pour la colonie si et seulement si deux fourmis reproductrices sont dans le nid et que la quantité de sucre est suffisante. Lorsque les fourmis reproductrices ne cherchent pas à engendrer de nouvelles fourmis, elles se comportent comme des fourmis ouvrières classique.

Plus concrètement : la marche à suivre est la suivante :

1. Une fourmi reproductrice se comporte comme une fourmi ouvrière.
2. Si une fourmi reproductrice détecte qu'il y a suffisamment de sucre dans le nid pour engendrer une nouvelle fourmi alors elle rentre dans le nid et attend.
3. Une seconde fourmi reproductrice doit également entrer dans le nid pour donner naissance à une nouvelle fourmi.
4. Si la quantité de sucre est la bonne, une nouvelle fourmi naît.
5. Les deux fourmis reproductrices sorte du nid et reprennent leur travail d'ouvrière jusqu'à ce que la quantité de sucre soit de nouveau atteinte au nid.

Le choix de la caste, lorsqu'une nouvelle fourmi naît, se fera aléatoirement. Attention à prévoir un mécanisme pour qu'il y ait suffisamment de fourmi reproductrice pour ne pas empêcher les naissances dans vos colonies.

1.2 Simulation

La simulation consiste essentiellement à initialiser la grille en plaçant les fourmis, les nids, le sucre et les gradients de phéromones de nid, puis à itérer un certain nombre de fois le déplacement des fourmis. Il faut aussi diminuer périodiquement l'intensité des phéromones de sucre pour tenir compte de leur évaporation.

Initialisation de la grille avec les phéromones de nid

Les phéromones de nid ont une intensité décrite par un réel compris entre 0 et 1. L'intensité vaut 1 sur le nid et diminue linéairement quand on s'éloigne du nid. Pour initialiser la grille avec les phéromones de nid, pour chaque nid, on commence par donner l'intensité 1 aux cases du nid et l'intensité 0 partout ailleurs. Ensuite, si la grille est constituée de t lignes et t colonnes, on itère t fois la règle suivante sur toutes les cases :

$$\text{Si } (p < 1) \text{ alors } p \leftarrow \max\left(m - \frac{1}{t}, 0\right)$$

avec p l'intensité des phéromones de nid sur la case considérée et m la valeur maximale des phéromones de nid sur les cases voisines.

Déplacement des fourmis

Pour chaque fourmi f sur une case p_1, on essaie d'appliquer les règles une par une, de la plus prioritaire à la moins prioritaire. Pour chaque règle, on cherche une case p_2 parmi les cases voisines de p_1 qui vérifient la condition d'application de la règle. Si l'on trouve une telle case, on applique la règle. Sinon, on passe à la règle suivante. Si aucune règle ne s'applique, la fourmi ne se déplace pas.

Remarque sur la règle 7 : si l'on parcourt le voisinage toujours dans le même sens pour trouver une case vide p_2 où déplacer la fourmi, on va beaucoup plus souvent sélectionner p_2 parmi les cases observées en premier. La simulation ne paraitra alors pas très naturelle, les fourmis ne se déplaçant pas vraiment aléatoirement. Par exemple, si l'on cherche une case voisine vide en commençant par regarder la case au dessus, alors les fourmis se dirigeront très souvent vers le haut. Lors de l'application de la règle 7, vous devrez donc choisir la case p_2 aléatoirement parmi les cases voisines vides. Pour un comportement encore plus aléatoire, on peut aussi appliquer un choix aléatoire pour les règles 4, 5 et 6.

Phéromones de sucre

L'intensité des phéromones de sucre est un entier compris entre 0 et 255, qui diminue de 5 à chaque itération de la simulation. Lorsque la fourmi pose des phéromones de sucre, elle pose directement 255.

2 Affichage

Dans un premier temps pour le test, on pourra faire un affichage texte avec, par exemple pour deux colonies de fourmis, `'f'` pour représenter une fourmi de la colonie 1 sans sucre, `'g'` pour représenter une fourmi de la colonie 2 sans sucre, `'F'` et `'G'` une fourmi avec du sucre, `'n'` pour le nid 1, `'m'` pour le nid 2 et `'s'` pour du sucre. Le défaut de cette méthode est qu'il sera difficile de visualiser les différents niveaux de phéromone, mais vous pourriez en afficher l'état séparément.

Dans un deuxième temps, on utilisera un affichage graphique, ce qui permet d'avoir des couleurs. On représente chaque case de la grille par un carré coloré. La couleur dépend du contenu de la case :

- les fourmis de la colonie 1 en vert ;
- les fourmis de la colonie 2 en rouge ;
- les sucres en blanc ;
- les nids en bleu ;
- les phéromones de sucre de chaque colonie d'une couleur proche de celle des fourmis de la colonie.

Si une case contient plusieurs éléments, on affiche en priorité l'élément qui est le plus haut dans la liste ci-dessus. L'amélioration suivante devra être apportée par la suite : les phéromones seront affichées avec un dégradé permettant de rendre compte de leur intensité. Ainsi, les cases seront vert ou rouge vif quand une phéromone de sucre vient d'être déposée et de plus en plus claires quand celle-ci s'évapore.

Pour l'affichage graphique, on propose d'implémenter une solution non-interactive (mais si vous avez le temps de faire une solution interactive, par exemple avec la bibliothèque `SFML`, n'hésitez pas). On va générer une suite d'images au format «ppm» dont les noms seront de la forme `img000.ppm`, `img001.ppm`, `img002.ppm` .... représentant chaque étape de la simulation. On va ensuite utiliser soit `convert` (sous Linux et Mac), soit le logiciel *beneton movie* pour les rassembler en un fichier gif animé :

— Sous Linux ou Mac, les fichiers ppm générés peuvent être rassemblés dans un fichier gif animé par la commande

```
convert -scale 300 -delay 10 img*.ppm movie.gif
```

Le paramètre 10 après delay peut être changé pour varier la vitesse d'animation. Le fichier gif peut ensuite être visionné avec Firefox. L'outil `convert` peut être installé sous Ubuntu avec :

```
sudo apt-get install imagemagick
```

— Sous Windows, le logiciel *beneton movie* est téléchargeable gratuitement à l'adresse `https://beneton-movie-gif.fileplanet.com/`.

3 Implémentation de la simulation

Afin de vous aider à réaliser le projet, nous vous donnons une analyse de la suite d'actions à mettre en œuvre pour implémenter la simulation.

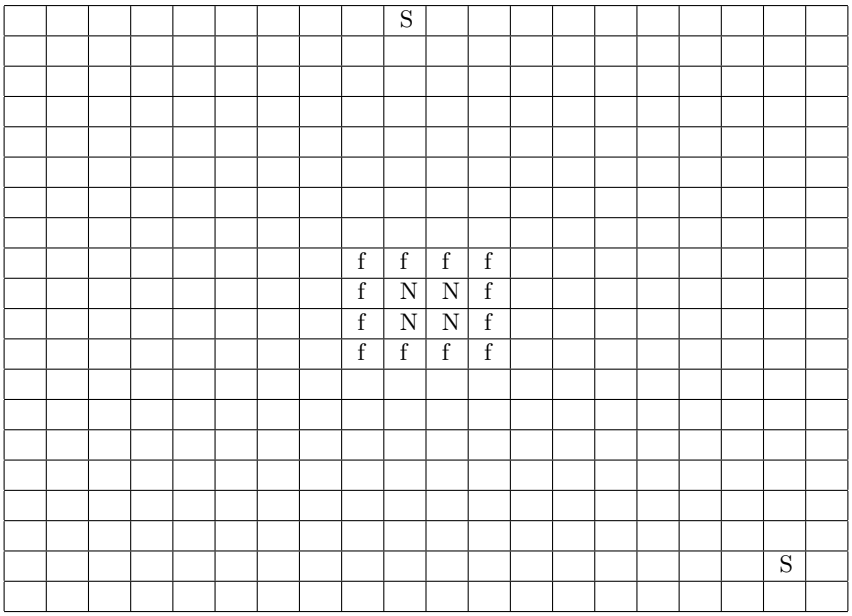
Après l'écriture de chaque fonction, il faut vérifier soigneusement qu'elle fonctionne bien. Quand c'est possible, on écrira un cas de test avec l'infrastructure `doctest`, sinon on fera un affichage. Dans ce second cas, on essayera également de conserver dans une procédure le code qui permet de reproduire ce test, sans le lancer automatiquement. Il faudra bien penser à sauvegarder régulièrement votre code en le soumettant sur le `gitlab`.

Implémenter la solution avec deux colonies est plus complexe, on commencera avec la simulation d'une seule colonie de fourmis. Dans ce cas, le nombre de fourmis sera fixe car elles ne peuvent pas mourir, nous conseillons dans ce premier temps de ne pas faire de type abstrait `Colonie`, mais d'utiliser un simple tableau contenant toutes les fourmis.

3.1 Mise en place et initialisation

Au cours de la simulation, on devra parcourir les fourmis à chaque fois pour les déplacer. Afin d'être plus efficace et ne pas perdre de temps à chercher les fourmis dans la grille, on aura à disposition un tableau des fourmis. Les fourmis ne seront donc pas directement stockées dans la grille : la grille contient le numéro de la fourmi (cela permet donc aussi un gain d'espace mémoire).

1. Spécifier **un type abstrait fourmi**, **un type abstrait place** (pour les cases) et **un type abstrait grille** contenant toutes les informations nécessaires pour la simulation.
2. Décider des types concrets associés.
   Aide 1 : Une fourmi doit être associée à ses coordonnées dans la grille. Pour cela, utiliser le type concret `Coord` défini dans le TP.
   Aide 2 : Comme une case peut contenir ou non une fourmi, s'il y a une fourmi, on identifiera celle-ci en notant son indice dans le tableau des fourmis, et on notera '-1' s'il n'y a pas de fourmi.
3. **Initialiser une grille** 20×20 avec au centre un nid de 2×2, 12 fourmis autour et deux tas de sucre loin du nid. L'intensité des phéromones de nid sera pour l'instant nulle partout. Le schéma suivant est un exemple de situation initiale où f = Fourmi, N= Nid, S= Sucre
4. Afficher le contenu de cette grille dans le terminal pour vérifier que l'initialisation a correctement été effectuée.
5. Écrire une procédure `deplacerFourmi` qui prend en paramètre une fourmi f, sa place p_1, une place p_2, et déplace f de p_1 vers p_2. Tester cette procédure en déplaçant une fourmi de la grille initiale.
6. Implémenter le prédicat `Place::estVide()`. À l'aide de la fonction `choisirCoordAleatoirement` du TP, déplacer chaque fourmi vers une des cases voisines vides et afficher la grille à nouveau.



7. Itérer 30 fois ce comportement et vérifier que ce que vous obtenez semble cohérent (si vous avez mis en place un affichage graphique, vous pouvez faire un premier film de 30 images).
8. Initialiser la grille avec **l'intensité de phéromone de nid** associée à chaque case. Afficher l'intensité en phéromone de nid des cases de la grille dans le terminal pour vérifier que l'initialisation a été correctement effectuée.

3.2 Déplacement des fourmis

9. **Implémenter les conditions de chaque règle** définie à la fin de la section 1.1 comme une fonction retournant *vrai* si la condition de la règle est vérifiée (une fonction par règle).
10. **Implémenter les règles 2, 3, 4 et 7** et déplacer les fourmis en suivant ces règles, sur 30 itérations. On doit observer quelques fourmis qui ramènent du sucre au nid.
11. **Rajouter l'apparition du sucre** sur une case aléatoire (vide) tous les 20 tours (définir une constante pour pouvoir faire modifier ce nombre).
12. **Rajouter les règles 5 et 6** permettant d'accélérer «exponentiellement» le rapatriement du sucre. Faire environ 300 itérations et observer à différents moments (si vous avez mis en place un affichage graphique, faire un film de 300 images).

3.3 Plusieurs colonies

13. **Sauvegardez votre projet**, pour le cas où vous ne réussiriez pas à finir la suite.
14. **Rajouter une classe Colonie** pour implémenter la fonctionnalité «deux colonies». Cette classe va remplacer le tableau de fourmis pour stocker les fourmis de chaque colonie. Dans cette classe, on va aussi stocker le stock de sucre de la colonie, et la position du nid.
15. **Mettre à jour votre code** pour prendre en compte deux colonies de fourmis. Modifiez vos fonctions pour prendre en compte l'existence des deux colonies. Agrandissez votre grille (50x50), rajoutez un nid sur la grille avec un deuxième gradient de phéromones de nid. Pour tuer une fourmi, il faudra passer par la classe Colonie. Il faudra aussi lister des constantes permettant de régler le comportement des fourmis (selon ce que vous choisissez d'implémenter : naissance, nombre maximum de fourmis, nombre de colonies,...)

16. **Rajouter la règle 1** de bataille entre deux fourmis et observer les rencontres de fourmis sur 50 itérations.
17. **Rajouter la naissance de fourmis**. Maintenant qu'elles peuvent mourir, ajoutez la fonctionnalité du nid permettant au nid de faire naître une nouvelle fourmi dès qu'il a au moins 10 unités de sucre. Les nids ont une taille de 2x2 cases, et les fourmis naîtront en sortant de la première case du nid (celle du haut à gauche). Observez l'évolution de vos deux colonies sur 50 itérations. Pour faire naître une fourmi, il faudra passer par la classe Colonie.

3.4 Polyéthisme de castes et eusocialité

18. **Sauvegardez votre projet**, pour le cas où vous ne réussiriez pas à finir la suite.
19. **Mettez à jour la classe Fourmi** pour permettre trois castes différentes de fourmis : Ouvrière, Guerrière et Reproductrice. N'implémentez pas encore les différents comportement.
20. **Modifiez l'affichage** pour distinguer les trois castes de fourmis.
21. **Implémentez** le comportement des fourmis ouvrières. Cela consiste principalement à enlever la possibilité de tuer d'autre fourmis pour cette caste.
22. **Implémentez** le comportement des fourmis guerrières. Commencez simplement par une marche aléatoire avec la possibilité de tuer des fourmis des nids adverses. Ajouter ensuite le suivi des pistes de phéromones de sucres des nids adverse.
23. **Sauvegardez votre projet**, pour le cas où vous ne réussiriez pas à finir la suite.
24. **Désactiver** la naissance des fourmis. A ce stade, vous devriez constater que plus aucunes nouvelles fourmis apparaît quelle que soit la quantité de sucre dans le nid.
25. **Implémentez l'entré** des fourmis Reproductrice dans le nid si la quantité de sucre requise est atteinte pour la naissance d'une fourmi.
26. **Faite naitre** une nouvelle fourmi si deux fourmis reproductrices sont dans le nid et si la quantité de sucre est suffisante.
27. **Faite sortir** les fourmis reproductrices du nid.

Si vous avez réussi à implémenter toutes les castes telles qu'énoncer ci-dessus, bravo ! ! Si le temps et l'envie vous animent, soyez créatif, vous pouvez ajouter de nouvelles castes ou comportements aux castes déjà existantes.

Dans l'état actuel, les colonies ne sont pas très réalistes. Voici quelques pistes d'amélioration :

— Les guerrières ont tendance à exterminer à elle toute seule les autres pauvres colonies de fourmis. Ajouter un mécanisme qui limite les capacités de destruction des fourmis guerrières.

— Les reproductrices reviennent directement dans le nid après avoir donné naissance ce qui les empêche d'accomplir pendant un temps leurs devoirs d'ouvrière. Ajouter un mécanisme de délai pour les fourmis reproductrices pour éviter cela.

— Les fourmis ne transporte qu'une unité de sucre. Vous pourriez faire en sorte que les fourmis ouvrières transportent plus d'unité de sucre que les fourmis reproductrices.

— Une fourmi victime d'une fourmi guerrière voit son sucre disparaître si elle en transportait. Faite déposer au sol la quantité de sucre qu'elle transportait pour que d'autres fourmis puissent en bénéficier.

3.5 Affichage graphique

L'affichage graphique est optionel et peut être fait à tout moment après la mise en place de la grille. Pour plus de détails, se reporter à la section 2 consacrée à l'affichage.

4 Organisation et évaluation du projet

Deux séances de cours et deux séances de TP porteront sur le projet. N'hésitez pas à poser des questions à vos encadrants de TP pour régler les problèmes que vous ne parvenez pas à surmonter. Il est indispensable de commencer à travailler sur le projet dès maintenant.

L'évaluation du projet se fera au travers d'une soutenance orale. L'organisation de la soutenance est la suivante :

— vous vous installez sur une machine, chargez votre programme et vérifiez rapidement que tout fonctionne comme la dernière fois que vous l'avez testé ;

— les deux membres du binôme ont 5 min pour présenter ensemble le résultat de leur travail (organisation du code, visualisation des résultats, discussion sur les problèmes rencontrés...) ;

— ensuite l'examinateur a 10 min pour juger de votre maîtrise de la programmation. À partir de ce moment, vous serez considérés individuellement, et un effort important sera fait pour mettre en avant clairement votre implication dans le projet. Un étudiant ne peut prétendre être noté sur un code qu'il ne réussit pas à expliquer.

— Un étudiant absent à la soutenance aura zéro sur vingt.

Un critère d'évaluation est le nombre d'actions fonctionnant correctement. Nous prendrons également en compte :

— Qualité de présentation...

— Types abstraits (tous les attributs sont privés) : pertinence, choix alternatifs, utilisation dans le code...

— Qualité du code : modularité, lisibilité...

— Originalité : gestion de problèmes rencontrés, affichage d'informations intermédiaires...

Attention ! Les critères ci-dessus ne seront pas évalués en regardant seulement le code, mais également par la manière dont vous répondez aux questions lors de la soutenance. Il est tout à fait autorisé de se faire aider, mais il faut avoir *écrit soit même, compris et être capable d'expliquer le code que l'on présente.* La note des deux membres du binôme est individuelle et dépend de leurs réponses aux questions et de leur implication dans le projet.

5 Annexe

Un fichier `ppm` (voir `https://fr.wikipedia.org/wiki/Portable_pixmap`) contient une image décrite par la suite de caractères suivante :

1. Un "magic number" identifiant le type.
   Pour un fichier ppm il s'agit des deux caractères "P3".
2. Des espaces (blancs, TABs, CRs, LFs).
3. Une largeur L, formatée avec des caractères ASCII en décimal.
4. Des espaces.
5. Une hauteur H, également en ASCII en décimal.
6. Des espaces.
7. La valeur de couleur maximale, en ASCII en décimal. Comprise entre 1 et 65536.
8. Un retour à la ligne ou d'autres caractères espace.

9. Un tableau de H rangées, allant de haut en bas, chaque rangée contient L pixels de gauche à droite. Chaque pixel est un triplet des composantes RGB (c'est-à-dire rouge, vert, bleu) de la couleur, dans cet ordre. Tous les nombres doivent être précédés et suivis d'un espace. Les lignes ne doivent pas dépasser 70 caractères.

Voici l'exemple d'un fichier img000.ppm contenant une image de 16 pixels (4 par 4), représentant une diagonale rouge sur fond vert (les composantes RGB de la couleur rouge sont (255,0,0) et celles de la couleur verte sont (0,255,0)).

```
P3
4 4
255
255 0 0    0 255 0    0 255 0    0 255 0
0 255 0    255 0 0    0 255 0    0 255 0
0 255 0    0 255 0    255 0 0    0 255 0
0 255 0    0 255 0    0 255 0    255 0 0
```

Ce fichier ppm a été généré par le programme suivant :

```
#include <iostream>     // pour cout
#include <iomanip>      // pour setfill, setw
#include <sstream>      // pour ostringstream
#include <fstream>      // pour ofstream
#include <string>

using namespace std;
// variable globale permettant de creer des noms de fichiers differents
int compteurFichier = 0;
// action dessinant un damier
void dessinerDamier(){
  int i,j;
  int r,g,b;
  ostringstream filename;
  // creation d'un nouveau nom de fichier de la forme img347.ppm
  filename << "img" << setfill('0') << setw(3) << compteurFichier << ".ppm";
  compteurFichier++;
  cout << "Ecriture dans le fichier : " << filename.str() << endl;
  // ouverture du fichier
  ofstream fic(filename.str(), ios::out | ios::trunc);
  // ecriture de l'entete
  fic << "P3" << endl
      << 4 << " " << 4 << " " << endl
      << 255 << " " << endl;
  // ecriture des pixels
  for (i = 0; i < 4; i++){
      for (j = 0; j < 4; j++){
        // calcul de la couleur
        if (i == j) { r = 255; g = 0; b = 0; }
        else { r = 0; g = 255; b = 0; }
        // ecriture de la couleur dans le fichier
        fic << r << " " << g << " " << b << "    ";
      }
    // fin de ligne dans l'image
    fic << endl;
  }
  // fermeture du fichier
  fic.close();
}
int main (){
  dessinerDamier();
  return 0;
}
```