Hartmann Florian - 269 726
Rimlinger Matthieu - 269 775

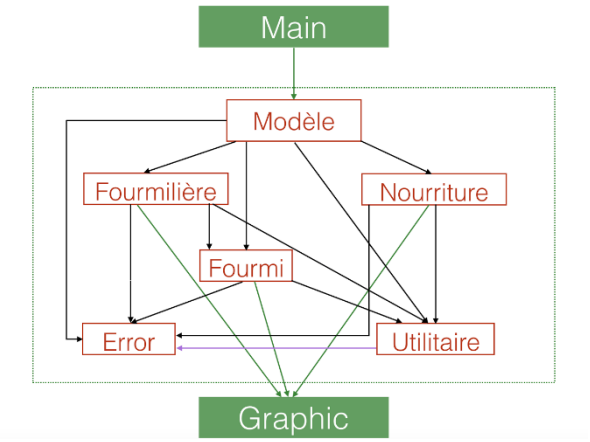
Programmation II - Rapport du rendu final :

I - Architecture logicielle et implémentation :

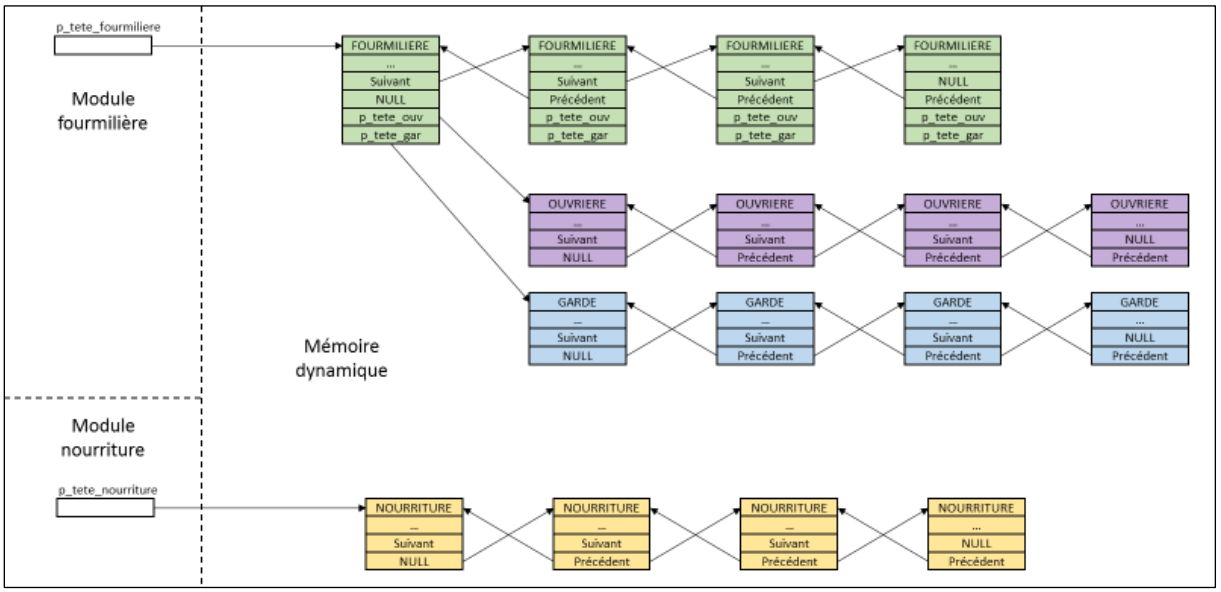
Nous avons réalisé le rendu final à partir de notre propre version du rendu 2, elle-même issue de notre rendu 1. Il n'y a pas eu d'importantes modifications des fonctionnalités existantes de notre programme entre le rendu 2 et celui-ci.

Notre programme conserve une architecture logicielle identique à celle du précédent rendu. Cette architecture diffère de la figure 8b par les points suivants :
- suppression du lien entre **modele** et **graphic**.
- suppression du lien entre **utilitaire** et **graphic**.
- ajout d'un lien entre **utilitaire** et **error**.

Ci-contre l'architecture logicielle finale de notre programme, très similaire à celle de la figure 8b.



Le stockage des données en mémoire se fait à partir de listes de structures doublement chainées allouées dynamiquement. Nous avons créé quatre types de structure de données différents : **FOURMILIERE**, **OUVRIERE**, **GARDE**, **NOURRITURE** (un type **POINT** existe également mais est d'ordre utilitaire).
Les pointeurs de tête de liste de fourmilières et de nourritures sont déclarés en variables globales statiques dans leur module. Dans le cas des listes de fourmis, il existe une liste d'ouvrières et de gardes propre à chaque fourmilière. Les pointeurs de tête de ces listes font donc partie des structures **FOURMILIERE**.



Par ailleurs, pour réaliser chaque mise à jour de la simulation, certaines données sont récupérées sur ces listes et stockées temporairement dans des tableaux pour être envoyées à d'autres modules (respect du type opaque). L'usage de listes chainées implique un **coût de mémoire linéaire**.

Hartmann Florian - 269 726
Rimlinger Matthieu - 269 775

Voici le **cout calcul** d'une mise à jour de la simulation, exprimé en fonction de :

- nbF - le nombre de fourmilières
- nbO - le nombre d'ouvrières d'une fourmilière, que l'on considère identique pour toutes les fourmilières
- nbG - le nombre de gardes d'une fourmilière, que l'on considère identique pour toutes les fourmilières
- nbN - le nombre d'éléments de nourriture présents sur le sol

Le coût calcul peut différer selon les simplifications faites sur la présence d'ouvrières intruses dans une fourmilière. En considérant le pire des cas, on note ce nombre NBF*(NBO-1) soit toutes les ouvrières de toutes les autres fourmilières présentes dans la fourmilière. La complexité est alors :

$$nbF^2*nbO*nbN + nbF^2*nbO^2 + nbF^3*nbO*nbG + nbN^2$$

(terme dominant en **nbF cube**)

Mais dans un cas plus probable, ce nombre d'intrus n'excède pas NBO. La complexité est alors :

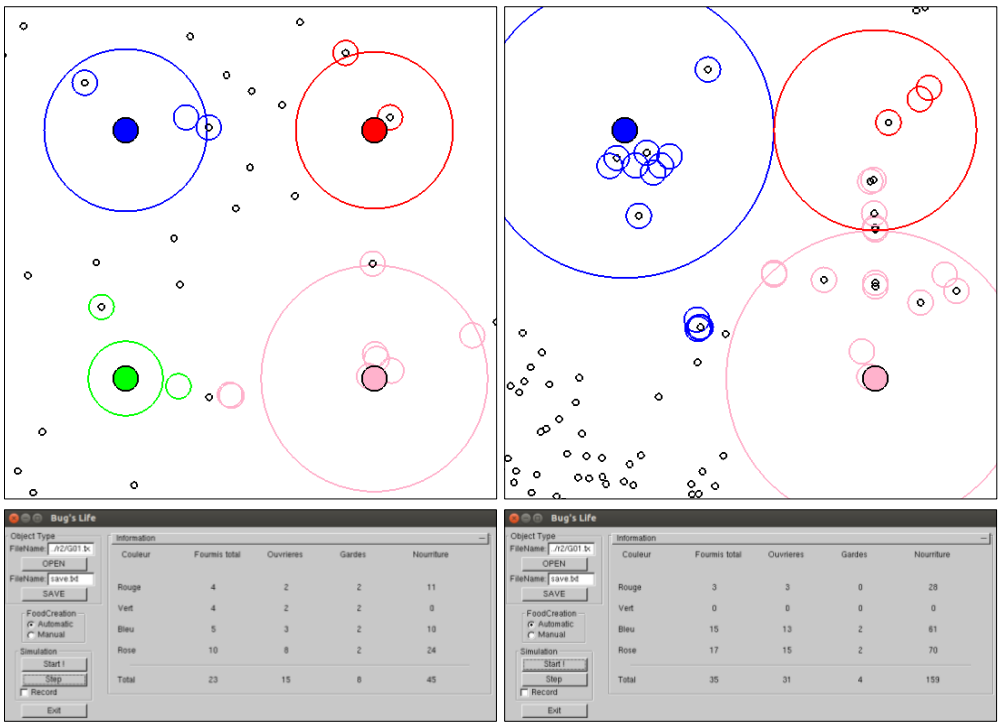
$$nbF^2*nbO*nbN + nbF^2*nbO^2 + nbF^2*nbO*nbG + nbN^2$$

(terme dominant en **nbF carré**)

Il apparait que les parties du programme les plus complexes sont celles gérant les tests de superposition entre fourmis pour les morts prématurées, et la mise à jour des positions des fourmis (bon choix de nourriture, test de trajectoire).

II - Illustrations :

Voici 4 images illustrant le déroulement de notre simulation sur le fichier G01.txt



Hartmann Florian - 269 726
Rimlinger Matthieu - 269 775

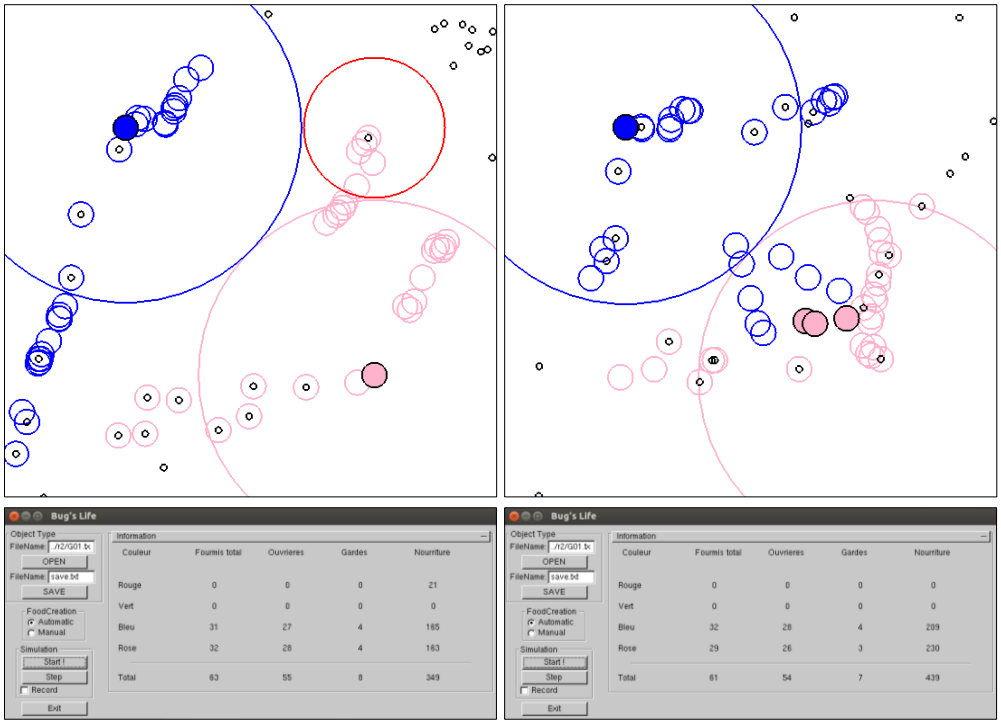


Fig. 1 : quatre fourmilières présentes, les ouvrières acquièrent de la nourriture.
Fig. 2 : la fourmilière verte a disparu, les trois autres sont tangentes (on observe le retour d'un raid des rouges sur les roses).
Fig. 3 : au fil des raids, la fourmiliere rouge diminue en taille.
Fig. 4 : la fourmilière rouge a disparu, un lutte acharnée s'engage entre bleus et roses (on observe la défense des gardes).

III - Méthodologie et organisation :

La majorité du temps, nous avons travaillé au même endroit, afin que chacun consacre le même volume horaire sur le projet. Notre « plan » de fonctionnement général fut le suivant :
Pour implémenter une fonctionnalité particulière, nous discutions de la façon dont nous voulions le faire (quelles informations sont nécessaires, quelles fonctions doit-on écrire). Puis chacun réalisait une partie du travail, avec la possibilité de se concerter en cours de route.

Il n'y a donc pas eu de personne en charge d'un module particulier, mais chaque membre a réalisé des fonctions individuellement au sein d'un même module. Nous sommes conscients que cette approche n'est pas optimale lors de la réalisation de « gros projets » impliquant beaucoup de personnes. Il serait préférable dans un tel cas d'avoir un responsable de module, connaissant les informations disponibles en entrée et devant travailler avec pour retourner un résultat précis.
Toutefois, notre mode de fonctionnement nous a permis de comprendre le détail du fonctionnement de tout le programme en vue de l'oral.

Pour le deuxième rendu, la séparation des tâches nous a parue assez évidente entre la réalisation de l'interface graphique (Florian) et la gestion dynamique des données (Matthieu).

Hartmann Florian - 269 726
Rimlinger Matthieu - 269 775

Pour le troisième rendu, nous avons effectués les tâches dans l'ordre chronologique ci-dessous :

- création aléatoire et manuelle de nourriture
- fonction de record, mise à jour du tableau GLUI (recherche d'informations)
- mise à jour des fourmilières (rayon, nourriture, destruction)
- mise à jour des gardes, chasse des éléments intrus
- mise à jour des ouvrières, algorithme du bon choix (nourriture la plus proche, trajectoires)
- récupération de nourriture, dépôt, vol et retour
- morts prématurées des fourmis
- attaques entre fourmilières

Nous pensons que cet ordre fût adéquat afin d'avancer efficacement et sans avoir à remettre en cause les fonctionnalités déjà implémentées. En particulier le fait d'avoir d'abord traité la création de nourriture fût utile pour ensuite vérifier le bon comportement de nos fourmis.

Les problèmes les plus fréquents que nous avons rencontrés concernaient le comportement des fourmis. Pour les résoudre, nous avons souvent eu recours à des scénarios de test faits par nos soins, afin de comparer le comportement de la démo et celui de notre programme. Pour résoudre les problèmes de boucles infinies ou de segmentation fault, l'usage des printf avec le nom des fonctions fut utile.

CONCLUSION :

Pour dresser un « bilan » de notre travail, nous pouvons commenter les points suivants :
- **Taille du code** : total de 3750 lignes. Cela nous semble raisonnable ; nous avons veillé au principe de réutilisation afin d'obtenir un code relativement léger.
- **Complexité** : le coût calcul décrit plus haut nous semble raisonnable, d'autant plus que la vitesse d'exécution est proche de celle du programme de démo.

Bien évidemment, même si ces deux aspects nous semblent convenables, ils pourraient être améliorés.

Quant au fonctionnement de notre groupe, nous pensons avoir été suffisamment efficaces et bien coordonnés. Ce projet nous a permis de nous familiariser avec ce type de travail. Nous pourrons donc réinvestir ces nouvelles compétences dans le futur.
Concernant l'environnement et les ressources mis à notre disposition, nous jugeons cela tout à fait convenable. Nous avons eu suffisamment d'occasion pour poser des questions, et bien que nous n'ayons pas utilisé les rapports publics, ceux-ci semblent d'un intérêt certain.