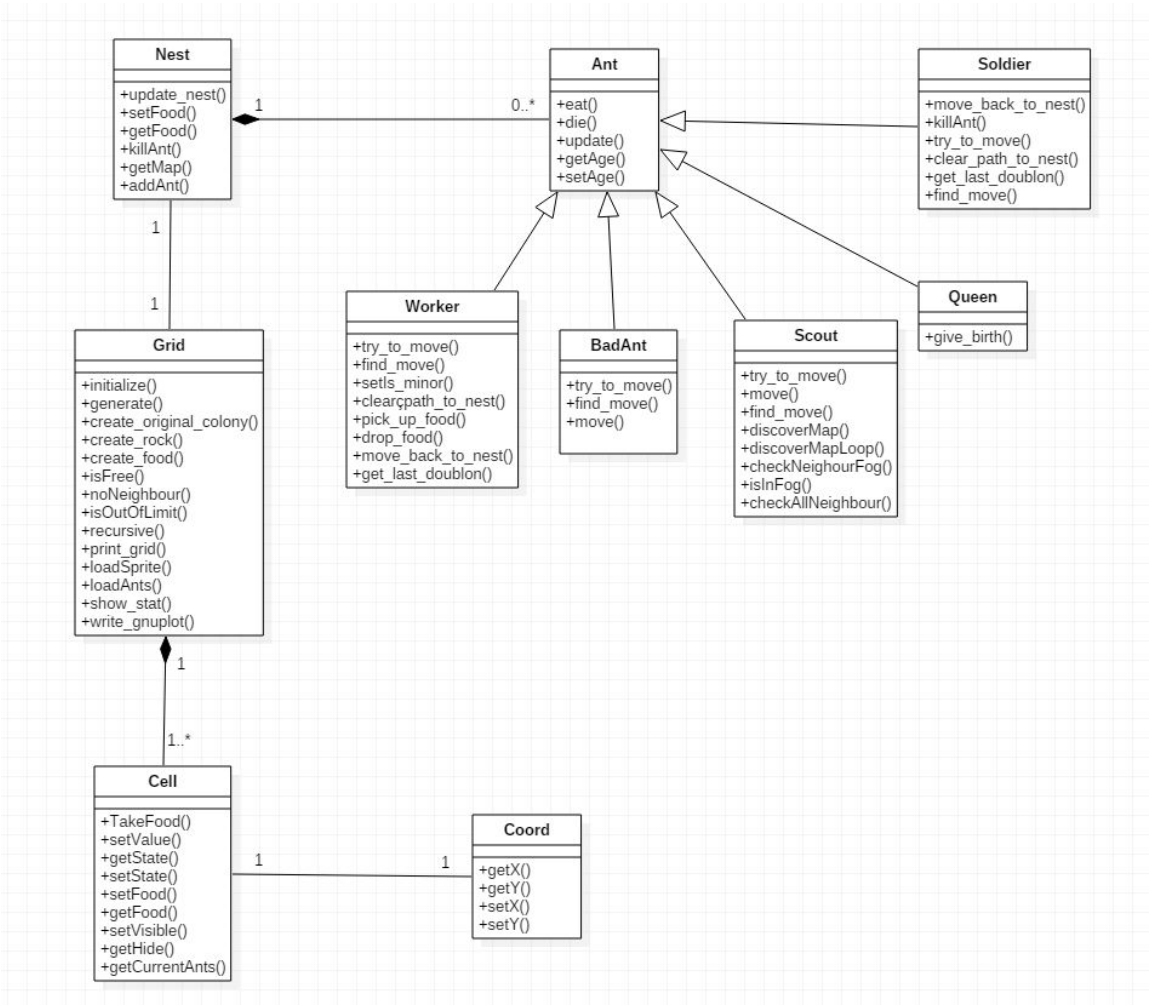
Rapport projet C++

MORVAN Jérome, THOMAS Edgar

Choix de conception

Nos choix de conception sont assez “basique” pour un projet orienté objet.
Voici notre diagramme de classe :



Nous avons une classe abstraite Ant avec plusieurs méthodes virtuelles à redéfinir dans toutes les classes filles.
Il y a 5 classes filles : Queen, Worker, Scout, Soldier and BadAnt.
Toutes ces classes ré-implémentent les fonctions virtuels de Ant selon leur comportement, mais ont aussi d’autre fonction propre à eux.
Nous avons une classe Nest, qui représente la colonie, et qui contient plusieurs fourmis.
La classe grid représente le terrain, il est composé de cellules, elles même composées de coordonnées.

Toutes nos fourmis implémentent donc une fonction update, qui correspond à la vie de la fourmi pour une journée.
Tous les jours, nous allons parcourir toutes les fourmis et appeler leur fonction update.

Difficultés rencontrés

Lorsque les fourmis travailleuses trouvent de la nourriture, il faut qu'elles reviennent à la colonie pour la déposer. Pour qu'une fourmi à un endroit x, trouve le chemin vers la colonie, il y a plusieurs méthodes plus ou moins efficaces en terme de temps de voyage pour la fourmi.
La méthode la plus efficace consisterait à implémenter un algorithme de Djikstra pour trouver le plus court chemin, mais cela aurait été long à implémenter et assez coûteux en performance. Nous avions donc simplement décidé de doter chaque fourmi d'une stack de coordonnées, et d'ajouter une coordonnées à cette stack à chaque fois que la fourmi bouge, ainsi, en dépilant tout simplement la stack, la fourmi sera capable de retrouver son chemin vers la fourmilière. Malheureusement, cela n'était pas très judicieux car la fourmi pouvait faire de nombreux détours inutiles au retour. Nous avons donc décidé de "nettoyer" la stack avant de retourner à la colonie, c'est à dire de supprimer tous les "détours" dans le chemin du retour. Malheureusement pour faire cela, une stack ne suffisait pas car il nous fallait accéder aux élément de la stack par indice, ce qui n'est pas possible. Nous avons donc dû passer par un vecteur intermédiaire.

La mise en place de SFML nous a également posé beaucoup de problèmes et nous a fait perdre énormément de temps.

Fonctionnalités non réalisées:

- vol d'oeuf de fourmis par les esclavagiste dans la colonie
- élimination des esclavagistes par les soldats
- application des phéromones
- expansion des cases dites de la colonie pour chaque centaine de fourmis présente.

Ces fonctionnalités n'ont pas été réalisées principalement par manque de temps, même si nous avons régulièrement travaillé hors des séances de TP, cela n'a pas suffit.

Répartition des tâches

L'interface graphique et tout ce qui touche à SFML ainsi que les fonctions relatives aux terrains et à la génération ont entièrement été réalisé par Jérome.

Les classes fourmis ont été implémenté dans par Edgar, mais les esclavagistes (BadAnt) ont été faites par Jérôme.

Les types de fourmis:

- Scout: Les fourmis éclaireuses iront en priorité sur les case non découvertes.

- Esclavagiste: ils apparaissent sur les bords de la grille et sont visible par dessus le brouillard pour que l'utilisateur puisse voir leur déplacement lorsque ceux-ci sont éloigné de la colonie. Leur déplacement est aléatoire.

- Soldat: ils apparaissent sur la colonie et ont un déplacement aléatoire. Il stock bien évidemment tout leur trajet depuis la colonie et reviennent régulièrement sur la colonie, en optimisant leur chemin de retour.

- Ouvrière: elles récoltent de la nourriture qu'ils apportent à la colonie grâce au chemin qu'ils stockent. A noter que lors du retour, le chemin est nettoyé puisqu'il est possible que ces dernières réalise des boucles avant de trouver de la nourriture. Ainsi, s'il y a dans le chemin stocké une coordonné apparaissant 2 fois, les coordonnées se trouvant entre les doublons seront supprimées.

Génération de la grille et graphisme

Fonctionnalités réalisées:

- génération d'obstacles de différentes tailles
- génération de blocs de nourritures (bloc seul et bloc de 5)
- mise en place d'un brouillard sur les cases
- ouverture d'une fenêtre avec SFML
- affichage de sprite pour les types de cases et les fourmis
- ouverture d'une fenêtre GNUPLOT
- déplacement de la caméra au dessus de la simulation et possibilité de zoomer
- affichage de statistiques en direct sur la simulation

- Possibilité d'accélérer ou ralentir la vitesse de la simulation avec les flèches gauche/droite

Choix de conception le plus important lors de la génération:
Utilisation d'un algorithme de "back-tracking" récursif pour générer des blocs d'éléments (par exemple 4 ou 5 obstacle, 5 bloc de réserve de nourriture) .
En effet, nous avons pris la décision d'utiliser la récursivité qui nous a semblé plus performante qu'un autre algorithme basique. Il faut noter que c'est lorsque nous sommes sommes parvenus à trouver x-bloc voisins disponible pour un obstacle ou de la nourriture et que la récursivité se "return" que nous changeons l'état des cases en obstacle ou nourriture. Ainsi, si nous avons à la fin de la récursivité l'impossibilité de placer les derniers block, nous n'avons pas à retourner en arrière puisque les obstacles ne sont toujours pas réellement généré et nous pouvons donc choisir une position aléatoire comme début de l'obstacle.
Lors de cette même génération nous laissons tout de même une trace sur la case choisis avant de passer à la suivante. En effet, nous nous sommes rendu compte qu'il était possible que la génération tourne en rond sur elle même et essaie de choisir comme nouvelle case du bloc une case qui pouvait être choisis par une itération antérieur. Nous effectuons donc une vérification afin de s'assurer que la case n'a pas déjà été choisis en observant qu'il n'a y a pas l'état "GHOST_ROCK" ou "GHOST_FOOD".

Aides extérieures

Nous avons beaucoup utilisé la documentation de SFML pour l'interface graphique.

Comme nous n'avions jamais fait de C++ avant, il nous arrivait souvent d'aller sur www.cplusplus.com/doc/ pour des questions de syntaxe ou de fonctionnement basique du langage. Naturellement, nous nous sommes aussi beaucoup retrouvé sur le forum StackOverflow.

Le seul code que nous avons importé à notre projet et la gestion du déplacement et du zoom de la caméra, que nous avons bien entendu modifier pour satisfaire nos besoins.

Concept	Mis en œuvre
Surcharge des fonctions	☒
Entrées / Sorties	☒
Références	☒
Constance	☒
Opérateur de résolution de portée	☒
Espaces de noms	☒
Fichiers d'en-têtes et espaces de noms	☒
Portée	☒
Gestion mémoire : Allocation	☒
Gestion mémoire : Libération	☒
Classes et objets	☒
Membres & partage	☒
Accès aux Membres & Portée	☒
Membres & Protection	☒
Fonction membre : accès & pointeur `this`	☒
Membres amis : `friend`	☐
Constructeur	☒
Destructeur	☒
Surcharge d'opérateurs	☐
Conversion utilisateur	☐
Fonctions amies *versus* fonctions membres	☐
Fonctions *inline*	☐
Constructeur, initialisation, affectation	☒
Constructeur, initialisation, affectation et allocation dynamique	☒
Classes dérivées – Héritage simple	☒
Héritage et constructeur	☒
Conversion classes de base <-> classes dérivées	☐
Dérivation et protection	☐
Vers le polymorphisme	☐
Liaison retardée, fonctions virtuelles	☒
Classes abstraites et fonctions virtuelles pures	☒
Type des objets et conversion explicite	☐
Héritage multiple	☐
Généricité : Patrons de fonction	☐
Généricité : Patrons de classe	☐
Flots, Entrées /Sorties, Fichiers	☒
La *Standard Template Library* (STL)	☒
Programmation défensive – Assertions	☐
Programmation défensive – Exceptions	☐

Captures d’écran de l'exécution du programme et du résultat gnuplot.