Rapport de projet - Fourmis V1

1. Explication du diagram de conception

classe **ControlFourmis :**

- Permet de faire le lien entre l'interface "AntFacadeController" et les autres classes.
- ControlFourmis est l'**expert en information** *(GRASP)*
- *ControlFourmis est le* ***créateur*** *(GRASP),* car il vas instancié : la reine, les soldats, le graphe, la nourriture

classe **Fourmis :**

- Une fourmi peut être une reine ou un soldat (pour l'instant)

- Une fourmi est sur un noeud et ne peut se déplacer que sur les voisins de ce noeud là (noeud.getFreeVoisins())

- On peut obtenir la positions d'une fourmi et établir sa positions (getPositions et setPositions)

- La reine ne peut se déplacer (Override de la méthode move(), elle peut néanmoins créé des soldats (createSoldiers)

classe **Noeud & Graphe :**

- La position des obstacles passe par la classe Graphe.

- Un nœud possède une liste de ses voisins. (getVoisins()), mais aussi une liste des ses voisins accessible, c'est à dire où getNoeudState() renvoie STATE.FREE.

- Un graphe possède une liste de noeuds

- Un nœud est occupé lorsque une reine (STATE.ANTHILL) ou un obstacle (STATE.OBSTACLE) est sur celle-ci.

- Un graphe peut créer une colonie (donc place une Reine sur une case)

- Si il n'y a plus de graphe , il n'y a plus de nœuds. Donc la relation Graphe vers Noeud est "complète", c'est donc une relation d'agrégation.

2. Principe GRASP & SOLID

2.a PRINCIPE GRASP

Patron Créateur & **Contrôleur** : La classe *ControlFourmis* est nécessaire afin de pouvoir créer les instances de classe à l'aide de l'interface *AntFacadeController*, c'est-à-dire que *ControlFourmis* est le créateur.

Patron Expert en information : La classe *Graphe* a l'information sur l'emplacement des noeuds,

Les *Noeuds* ont l'information : s'il possède un obstacle ou non, leurs nœuds voisins.

Les fourmis ont l'information sur quel noeud ils sont.

Patron Polymorphisme : la classe Fourmis contiendra la méthode move() qui permettra de déplacer les fourmis, mais elle sera définie dans les sous classes de Fourmis.(Par ex : le déplacement de la reine sera mis à null car elle ne peut pas se déplacer). Cela nous permet d'anticiper l'ajout de nouveaux types de fourmis sans pour autant ajouter un switch dans le code qui deviendra conséquent, ce qui implique alors un **faible couplage**.

Patron invention pure : utilisation future pour la V2 (création du fichier des itérations des fourmis).

2.b PRINCIPE SOLID

Single Responsibility

- Le graphe a pour responsabilité de s'occuper des nœuds
  .
- ControlFourmi ne répond pas à ce principe, mais cela est dû au fait que cette classe est la classe créatrice (Principe **GRASP**).

- La classe nœud a pour responsabilité de permettre l'emplacement d'une fourmis, d'un obstacle ou d'une fourmilière à l'aide de l'énumération STATE.

Open-Closed

- Ouvert à l'extension : On peut ajouter de nouvelle type de fourmis (le type ouvrière sera ajouté par la suite grâce à ce principe), de même pour l'énumération STATE, nous pourrons ajouter un nouveau statut comme par exemple, le fait de savoir si il y a une fourmis ouvrière ou non.

Liskov Substitution Principle

- On est pas impacté par ce principe, donc pas d'utilisation de ce principe

Interface Segregation Principle

- Pas trop utilisé car compliquerait pour rien la conception (sur-ingénierie)

Dependency Inversion

- Un graphe est composé de Noeud, peu importe leur composition ou leur nombre. Pas spécialement de tableau 2d (rectangulaire)

- Nous dépendons aussi de l'interface *AntFacadeController* (+Interface Segregation Principle)

3. Choix de conception

3.0. Convention anglais/français

Nous somme partie de base en français, mais nous avons étais conseiller d'écrire les variables/fonctions en anglais, alors nous avons du faire du franglais.

3.1. Fourmi sur un nœud...

Nous avons décidé que c'est la fourmi qui connait le nœud où elle se trouve et **non** le nœud qui connaît les fourmis sur elle.
La fourmi a un attribut -position de type Noeud.

3.2. Collection Java et voisins..

Nous avons décidé de partir sur une ArrayList (tableau 1d avec des index et sans limite. Cela nous permet de ne pas se restreindre à un tableau 2d et ainsi avoir une structure qui puissent changer. Ainsi chaque nœud (sommet) aura une liste de voisins. Les listes de voisins sera instanciée dans le constructeur du Graphe, après avoir instancié tous les nœuds dont nous avions besoin.

3.3. Statut des noeuds..

Les noeuds ont 3 statut différents :

- Libre (FREE)
- Obstacle (OBSTACLE)
- Fourmilière (ANTHILL)

Ce qui permet de différencier les nœuds et nous permet de par exemple ne pas permettre au soldat de passer par un nœud avec un obstacle ou de ne pas placer un obstacle sur une fourmilière.