01.05.2022 Rayan Gauderon, Maxime Luyet

Rapport rendu 2

**Hiérarchie de classe des fourmis :**

Dans notre module fourmis, nous avons décidé de définir une superclasse Fourmi. Les Defensor, Collector, Predator et Generator sont des fourmis et par conséquent des sous-classes de Fourmi. Notre hiérarchie de classe se fait donc sur 2 niveaux uniquement.

La superclasse Fourmi dispose d’un attribut de type *Carre*, une structure permettant de stocker les coordonnées x, y et la longueur du côté de la fourmi. La classe Generator dispose elle d’un attribut de type unsigned int pour le stockage du nombre de nourriture ainsi qu’un getter associé. Les autres classes disposent aussi d’un attribut de type unsigned int pour leur âge ce que n’a pas la fourmi Generator car cette dernière meurt en même temps que la fourmilière. La classe Collector possède un attribut booléen permettant de savoir si la fourmi transporte une nourriture ou pas.

Concernant les méthodes, la superclasse Fourmi gère l’initialisation des fourmis sur la grille de booléen (cf description des types mis en œuvre dans squarecell). Elle dispose aussi d’une méthode *fourmis\_in\_house* de type booléen renvoyant true si les fourmis ne sont pas dans leur fourmilière. Bien évidemment, cette méthode concerne les fourmis de type Generator et Defensor et va donc être appelée uniquement pour ces deux sous- classes. Ensuite, toutes les classes possèdent une méthode *superposition\_fourmi* de type booléen permettant de savoir s’il existe une superposition avec les autres fourmis ou les nourritures disposées sur la grille elle-même de type booléen (cf description des types mis en œuvre dans squarecell). Tout d’abord nous voulions placer cette méthode dans la superclasse mais étant donné que les messages d’erreur sont propres à chaque type de fourmis, nous avons dû nous raviser et la placer dans les classes dérivées. Finalement, les classes Generator et Defensor possèdent en plus une méthode qui va appeler la méthode *fourmis\_in\_house* mise à disposition dans la classe Fourmi et afficher, en cas d’erreur, leur message attribué.

Pour la partie polymorphique, nous avons deux méthodes virtuelles pures et une virtuelle dans la superclasse Fourmi :

1. *test\_chaque\_fourmi*
2. *draw\_fourmis*
3. *ecriture\_frmi*

La première est une méthode qui comme son nom l’indique, sert à effectuer le test de validation pour chaque fourmi. Elle s’occupe d’appeler les méthodes *test\_validation\_carre* mises à disposition dans squarecell ainsi que *superposition\_fourmi* et *fourmis\_in\_house* pour le cas des Generators et Defensors. Il est nécessaire de la rendre pure car les tests sont propres à chaque fourmi. De même pour la deuxième méthode, la représentation de la fourmi va dépendre de son type. Par exemple, une fourmi de type Collector va être dessinée avec un motif en diagonale alors que la Defensor, un motif en forme de grille. Donc il est aussi nécessaire de rendre cette méthode pure. Par contre pour la troisième méthode, qui sert à sauvegarder la simulation courante dans un fichier à part, les fourmis de type Defensor et Predator possédant uniquement des coordonnées peuvent être directement sauvegarder depuis la classe Fourmi. Il est donc judicieux de rendre cette classe virtuelle mais pas pure. Ainsi, on peut voir que le polymorphisme est important pour le bon déroulement de notre programme.

**Structuration des données des autres entités du Modèle :**

La classe Simulation possède deux attributs principaux, l’ensemble de fourmilières ainsi que l’ensemble de nourriture, l’un étant un vecteur de fourmilière et l’autre, de nourriture. Nous avons décidé de faire ainsi car il nous semblait que c’était la manière la plus simple et efficace pour écrire ce programme.

Concernant la classe Fourmilière, elle possède cinq attributs et tout comme les fourmis un attribut de type *Carre* pour y définir sa taille ainsi que ses coordonnées x, y. On y trouve aussi le nombre de Collectors, de Defensors et de Predators qui appartiennent à la fourmilière. De plus, cette classe est responsable de ses fourmis, un attribut de type *Ensemble\_fourmiliere*, lui-même étant un vecteur de unique\_ptr (pour bénéficier du polymorphisme).

Finalement, la classe Food possède aussi un attribut de type *Carre* auquel se rajoute un attribut représentant sa valeur nutritive.

Comme mentionné auparavant, la classe Simulation est responsable de l’ensemble de nourriture et de l’ensemble de fourmilières. Chaque fourmilière est alors responsable de l’ensemble de fourmis qu’elle contient. Ainsi cette structuration permet d’avoir un code clair et simple.

**Description des types mis en œuvre dans squarecell** :

Dans ce module, nous avons défini une structure *Carre*, composée de deux champs, un de type unsigned int pour sa longueur et l’autre de type *Point*, une autre structure regroupant ses coordonnées x, y. De plus, on y trouve une grille de booléen initialisée à false et cachée dans l’implémentation de squarecell ce qui veut dire concrètement que seul squarecell a accès à cette grille, tous les autres modules n’en sont pas autorisés. Cette grille est importante car c’est à cet endroit que l’on va effectuer tous les tests nécessaires pour que notre simulation n’ait pas d’erreurs.