Programmation Orientée Objet

PROJET 2015-2016 : ANTS Vs BEES

Etudiants:

CRAPANZANO Claire

MEA Cédric

SOMMAIRE

Introduction :

Durant ce projet, nous avions à compléter le programme Ants Vs Bees, un jeu de Tower Defence dont la base était fournie, en suivant dans un premier temps les instructions pour les différentes modifications à effectuer, et dans un second temps, en le complétant librement nous-mêmes.

Passons au détail de ce que nous avons programmé.

I. Le Sujet imposé

1. Les Fourmis

Nous ne détaillerons pas à chaque fois le fait que nous avons changé le super et le foodcost dans le constructeur des fourmis pour respecter le sujet.

HARVESTER ANT : Immédiate avec la méthode *colony.increaseFood()* et le *super*.

THROWER ANT : Nous veillons à ce que la fourmi ne fasse une action que si elle est en vie.

HUNGRY ANT : Pour simuler le *cooldown*, nous avons rajouté un paramètre *time*, que nous avons utilisé dans *action* pour veiller, à l'aide de *if* et d'incrémentation de *time*, à ce que la fourmi ne fasse son action qu'une fois sur trois.

SCUBA ANT : par rapport à *Thrower Ant*,Nous avons juste ajouté un attribut *watersafe* qui est vrai pour la *Scuba* et faux pour la première.

FIRE ANT : Nous utilisons un tableau pour stocker toutes les abeilles qui sont sur la case de la fourmi, puis nous réduisons leur armure dès que l'armure de la fourmi est nulle.

WALL ANT : Nous modifions le *super* pour mettre beaucoup d'armure, et nous redéfinissons le *action* pour qu'elle ne fasse rien.

NINJA ANT : Nous mettons la variable *BloqueChemin* à *false* afin que les abeilles ne soient pas bloquées et qu'elles n'attaquent donc pas la fourmi ninja. *BloqueChemin* est un paramètre rajouté dans *Ant* et qui sert dans *Bee*, qu'on utilise comme condition d'attaque dans l'*action* de ces dernières et comme condition pour laisser ou non passer l'abeille.
Nous utilisons aussi un tableau comme dans Fire Ant.

BODYGUARD ANT : Nous implémentons Containing dans cette fourmi, et créons un paramètre fourmiContenue, qui indiquera sur quelle fourmi on place le Bodyguard, ainsi que des méthodes permettant de savoir si cette fourmi a ou non une fourmi contenue; ces méthodes seront utilisées dans Place, pour gérer l'ajout de fourmis.
Son action sera celle de la fourmi contenue; elle ne fera rien.

QUEEN ANT :
On implémente une interface *QueenPlace* qui représente la place de la reine au fond des tunnels et celle qu'on ajoute

nous-mêmes dans le jeu. Elle servira pour le aussi pour *GameOver*
Nous initialisons le compteur de reine à -1 car une reine est automatiquement créée à la sortie des tunnels.
Vu que la reine n'est pas supprimable, nous lui rajoutons un paramètre qui l'explicite.
Pour pouvoir supprimer les autres reines, on rajoute un compteur *nbreInstance* qui s'incrémente à chaque création de reine. Pour *nbreInstance* supérieur à 1 on supprime immédiatement la reine créée.
Pour galvaniser les fourmis voisines, nous créons une liste stockant les fourmis à l'entrée et à la sortie de la reine.
Nous avions une exception quand nous mettions la reine tout au bout des tunnels; cette erreur venait du fait que la reine ne pouvait appliquer son bonus à gauche/à droite d'elle. Nous avons donc rajouté des conditions sur les cases adjacentes (avec des *getExit/Entrance() != null*) pour savoir si nos ajoutons ou non des fourmis à la liste des fourmis encouragées.

2. Les Interfaces

WATER : On implémente une interface *Water* définissant une méthode pour placer l'eau. Nous utilisons cette interface dans *Place*, dans *AddInsect*, où on vérifie que la fourmi est *watersafe* si la case est une *instanceof Water*
Nous créons *watersafe* dans *Insect* et l'initialisons à *false*, puis on le modifie dans les fourmis qui résistent à l'eau.

CONTAINING : Nous implémentons cette interface et nous l'utilisons dans la méthode addInsect de Place, de la façon suivante :
Si la fourmi qu'on place implémente Containing, on lui rajoute une fourmi si elle n'en n'a pas déjà.
Si la fourmi sur laquelle on veut placer notre fourmi implémente Containing, on ajoute notre fourmi à la la Containing déjà placée, puis on place le tout à la même place.
Si la place à laquelle on veut placer une fourmi est déjà occupée par une instance de Containing déjà occupée par une fourmi, on renvoie un message d'erreur.
Nous l'utilisons aussi dans removeInsect, où, si sur une place on a deux fourmi dont une Containing la Containing est alors supprimée en première

UNDELETABLE : Nous implémentons une interface sans méthode, qui ne nous servira que pour la reine. Dans *Ant*, nous mettons le paramètre que nous rajoutons *isDeletable* à *true* par défaut, puisque toutes les abeilles exceptée la reine sont supprimables, et nous le mettons à *false* seulement si la fourmi est une instance de *Deletable*.

II. <u>Les bonus</u>

1. Les bonus proposés

SHORT & LONG : Nous modifions juste la portée par rapport à *Thrower.*
SLOW & STUN : Nous avons commencé par programmer des méthodes permettant d'arrêter les abeilles lorsqu' (grâce à *movingTo(place)* pour qu'elle reste sur place) dans *la methode Action de Bee* . Cependant, ces méthodes s'arrêtaient quand les fourmis étaient tuées; nous avons donc du définir des paramètres *slow* et *stun* dans *Bee*, puis créer les boucles permettant de passer son tour dans la méthode *action* de *Bee.*

2. Nos propres ajouts

:

- GlassCannonBee (en belge) : Des abeilles faisant beaucoup de dommage, mais mourant facilement.
- NotStunBee (en rouge noir) : Des abeilles qui font peur aux Stun et Slow, qui ne sont donc ni étourdies ni ralenties.

AUTRES MODIFICATIONS : Pour que le jeu soit plus lisible, nous avons rajouté les noms des fourmis et changé les messages en français. Nous avons aussi refait une partie des *sprites*.

III. Bilan

Tableau du nombre d'heures

	Cédric	Claire
Recherches	5h	5h
Programmation	15h	11h
Tests	3h	3h
Bonus	8h	6h
Rapport	2h	3h
Total	**33h**	**28h**

Conclusion

Durant ce projet, nous avons appris plusieurs choses. Nous nous sommes familiarisés avec les *Array<List>* et les différentes manières de déboguer un programme, ainsi qu'avec l'usage de l'héritage, des packages et des interfaces. Il nous aura aussi permis d'apprendre à utiliser GitHub et les différentes possibilités d'Eclipse.

Ce qui nous a posé le plus de problème dans ce projet ont été l'interface *Containing*, la *Queen* et les *Stun* et *Slow Ants*. En effet, la première fut longue et nous avons passé beaucoup de temps au niveau des if et des méthodes à utiliser. La seconde nous a levé quelques exceptions, à cause du compteur, de la suppression et du problème détaillé plus bas/haut pour les bonus. Les dernières nous ont causé du soucis au niveau de la façon dont on devait les implémenter; avant de trouver la méthode *moveTo*, nous avions essayé beaucoup de choses infructueuses, comme modifier le *framerate* du jeu ou encore essayer de stopper ou relancer l'animation de l'abeille.