Projet 2015-2016

POO : Programmation Orientée Objet

Première année

★ Modalités

Ce projet est à réaliser en binôme. Vous indiquerez la composition de votre binôme dans un fichier dénommé `AUTHORS` (placé à la racine du répertoire de votre projet) qui doit contenir les noms et logins UNIX des membres du groupe, un par ligne.

Travail à rendre : Vous devez rendre un mini-rapport de projet (5 pages maximum, format pdf) en plus de vos fichiers sources. Vous y détaillerez les difficultés auxquelles vous avez été confronté, et comment vous les avez résolues. Vous indiquerez également le nombre d'heures passées sur les différentes étapes de ce projet (conception, codage, tests, rédaction du rapport) par chaque membre du groupe.

Comment rendre votre projet : Vous déposerez sur ARCHE une archive (.zip ou .tar.gz) contenant le code source (toutes les classes, avec documentation) de votre projet ainsi que votre rapport.

`http://arche.univ-lorraine.fr/mod/assign/view.php?id=176315`

Avant le 25 mars 2015, à 23h55.

(aucune soumission en retard ne sera acceptée)

Évaluation : Un projet ne compilant pas sera sanctionné par une note adéquate. Aucune soumission par email ne sera acceptée. Des soutenances individuelles de projet seront organisées à une date ultérieure. Vous serez jugé sur la qualité de votre programme et de sa documentation, celle de votre rapport et votre capacité à expliquer son fonctionnement.

★ Travail personnel et honnêteté

Ne trichez pas ! Ne copiez pas ! Si vous le faites, vous serez lourdement sanctionnés. Nous ne ferons pas de distinction entre copieur et copié. Vous n'avez pas de (bonne) raison de copier. En cas de problème, nous sommes prêt à vous aider. Encore une fois : en cas de doute, envoyez un courriel à vos enseignants, ça ne les dérange pas.

Par tricher, nous entendons notamment :

- Rendre le travail d'un collègue avec votre nom dessus ;
- Obtenir une réponse par Google™ ou autre et mettre votre nom dessus ;
- Récupérer du code et ne changer que les noms de variables et fonctions ou leur ordre avant de mettre votre nom dessus (*"moving chunks of code around is like moving food around on your plate to diguise the fact that you havn't eated all your brussel sprouts"*) ;
- Permettre à un collègue de *s'inspirer* de votre travail. Assurez vous que votre répertoire de travail n'est lisible que par vous même.

Il est plus que très probable que nous détections les tricheries. Chacun a son propre style de programmation, et personne ne code la même chose de la même manière. De plus, il existe des programmes très efficaces pour détecter les similarités douteuses entre copies (MOSS, `http://theory.stanford.edu/~aiken/moss/`).

En revanche, il est possible (voire conseillé) de discuter du projet et d'échanger des idées avec vos collègues. Mais vous ne pouvez rendre que du code écrit par vous-même. Vous indiquerez dans votre rapport toutes vos sources d'inspiration (comme les sites internet de vulgarisation de l'informatique que vous auriez consulté).

★ Bibliographie

Le problème présenté a été initialement proposé par John DeNero, Tom Magrino, and Eric Tzeng en 2014 en tant que *Nifty Assignment* au groupe d'intérêt *ACM Special Interest Group on Computer Science Education (SIGCSE)*. Le sujet original peut être consulté à l'adresse suivante :

`http://nifty.stanford.edu/2014/denero-ants-vs-somebees/`

Il a été adapté pour le langage Java par Joel Ross :

`http://faculty.washington.edu/joelross/courses/archive/f14/cs261/hwk/2/`

La documentation de l'API Java : `http://docs.oracle.com/javase/8/docs/api/`.

★ Présentation du problème : Fourmis contre Abeilles

Dans ce projet, vous allez créer un jeu de type « tower defense », appelé « Fourmis contre Abeilles », inspiré du jeu « Plants vs. Zombies ». Dans ce jeu, vous pouvez contrôler la reine des fourmis, et équiper votre colonie avec les fourmis les plus courageuses que vous pouvez trouver. Vos fourmis devront protéger leur reine des abeilles malveillantes qui attaquent votre territoire. En jetant des feuilles aux abeilles, les fourmis pourront énerver les abeilles et les faire fuir. Si vous n'arrivez pas à protéger votre reine, elle sera attaquée par les abeilles et risque de mourir.

Pour implémenter ce jeux, on profitera des avantages de la programmation orientée objet, en utilisant l'héritage et le polymorphisme pour simplifier la programmation et éviter la duplication de code. De plus, ce projet vous permettra de vous entraîner à travailler avec un large projet existant, notamment à le lire et comprendre.

★ Objectifs

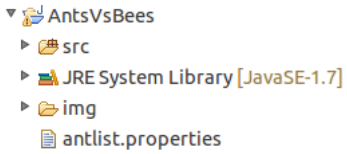
- S'entrainer à la programmation objet avec héritage et polymorphisme
- S'entrainer à la lecture, compréhension et modification de code existant

★ Fichiers nécessaires

Vous trouverez sur `ARCHE` (`http://arche.univ-lorraine.fr/course/view.php?id=7973`), un fichier .zip contenant les images et squelettes du programme.

Ce fichier contient un grand nombre de fichiers Java, qui implémentent le cœur du jeu. Il vous faudra importer ces fichiers dans Eclipse.

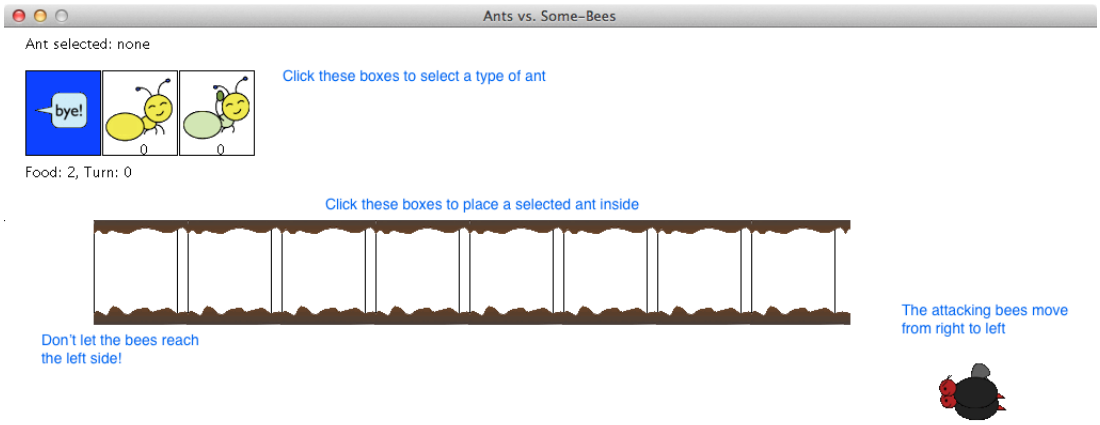
- Le fichier .zip contient d'autres fichiers additionnels. Le répertoire `img` contient des images et icônes pour les fourmis, et le fichier `antlist.properties` contient de l'information sur quel image va avec quel type de fourmis (ceci est un bon exemple de l'utilisation d'un fichier externe pour stocker de l'information, ce qui permet de réutiliser et modifier le programme sans recompiler). Il vous faudra mettre ces fichiers dans le **répertoire principal** de votre projet Eclipse, pour obtenir ceci :

  (Attention : le répertoire `img` et le fichier `antlist.properties` se trouvent à l'intérieur du répertoire `projet`, et pas dans `src` !)

★ Le jeu

Vous pouvez lancer le jeu en exécutant la méthode principale de la classe `AntsVsSomeBees.java`. Vous devriez voir une interface ressemblant à ceci (les commentaires en bleu décrivent comment jouer) :



Le jeu est prêt à être joué !

Vous pouvez placer des fourmis et essayer de défendre votre reine contre les abeilles. (Dans le code initial, les fourmis ne consomment pas de nourriture.)

Ce jeu est composé de plusieurs composantes :

— **Colony :** Les fourmis vivent dans une colonie, c'est-à-dire un ensemble de tunnels, qui sont représentés par des objets de type `Place`. La colonie a un stock de nourriture, nécessaire à la reproduction des fourmis. Seulement une fourmi peut occuper une `Place` (un endroit dans le tunnel), mais il n'y a pas de limite sur le nombre d'abeilles !

— **Ants :** Créer une fourmi supplémentaire consomme de la nourriture ; la quantité de nourriture nécessaire varie en fonction du type de fourmi à créer. Chaque type de fourmi peut faire des actions différentes. Les deux types les plus simples sont la `HarvesterAnt`, qui produit une unité de nourriture par tour, et la `ThrowerAnt`, qui jette une feuille vers les abeilles chaque tour.

— **Bees :** Les abeilles vont voler à travers les tunnels vers la reine, qui se situe tout à gauche. Si une fourmi bloque le trajet d'une abeille, elle va s'arrêter et piquer la fourmi.

Un jeu de « Fourmis contre Abeilles » se joue en une série de tours. Dans chaque tour, des abeilles peuvent entrer dans la colonie. (Vous pouvez rajouter des fourmis à tout instant.) Après, tous les insectes (fourmis, puis abeilles) exécutent leurs actions : les abeilles piquent les fourmis, et les fourmis jettent des feuilles vers les abeilles. Le jeu se termine dans deux cas : soit la reine a été atteinte par une abeille (et vous avez perdu), soit toutes les abeilles ont été éliminées par vos fourmis (et vous avez gagné).

★ Le code

L'implémentation de ce jeu utilise plusieurs classes – comme pour tout programme orienté objet bien conçu, chaque concept dans le jeu est implémenté par sa propre classe.

— La classe `AntColony` modélise une colonie de fourmis. Elle contient les endroits dans la colonie, avec une interface publique qui restreint l'accès à ces endroits. Les endroits sont représentés par la classe `Place`, qui modélise un seul placement dans un tunnel, et les insectes qui y résident. La classe `Hive` est une sous-classe de `Place`, elle contient des champs et méthodes supplémentaires pour tracer l'invasion des abeilles.

— Les insectes eux-mêmes sont modélisés par la classe `Insect`, avec une sous-classe pour les fourmis et une sous-classe pour les abeilles. Tout insecte a un emplacement (`Place`), une valeur de ARMOR, et une méthode `action()` qui exécute les actions de l'insecte à chaque tour. La classe `Ant` a d'autres sous-classes pour les différents types de fourmis, avec des variables et actions propres. Les deux classes de base, `HarvesterAnt` et `ThrowerAnt`, sont déjà fournies pour que vous puissiez tester le jeu.

— Le jeu lui-même est géré par la classe `AntGame`. Cette large classe est un `JPanel` sur lequel les fourmis et abeilles sont affichées. De plus, il y a un `Timer` qui affiche de façon récurrente de nouvelles instances du jeu pour visualiser son évolution.

— Vous pouvez lancer le jeu avec la classe `AntsVsSomeBees`, qui contient la méthode principale. Cette méthode instancie un objet `AntColony` (vous pouvez regarder son JavaDoc pour plus de détails) et un objet `Hive`, puis démarre le jeu. Quelques options qui pourront vous servir pour jouer :

- `AntColony colony = new AntColony(1, 8, 0, 2); //is the default testing layout (one tunnel)`
- `AntColony colony = new AntColony(1, 8, 0, 10); //testing layout with 10 food`
- `AntColony colony = new AntColony(3, 8, 0, 2); //makes a full tunnel layout`
- `AntColony colony = new AntColony(3, 8, 3, 2); //makes a full tunnel layout with water (see below)`
- `Hive hive = Hive.makeTestHive(); //makes a simple hive with just a couple bees`
- `Hive hive = Hive.makeFullHive(); //makes a hive full of bees`
- `Hive hive = Hive.makeInsaneHive(); //makes a challenging hive!`

Vous pouvez modifier et mélanger ces constructeurs afin de jouer à des configurations de partie différentes.

— Le code est organisé dans deux **paquetages**. Le paquetage `core` contient la majorité des classes (le coeur du jeu). Le paquetage `ants` contiendra tous les différents types de fourmis que vous allez créer !

Vous serez amenés à modifier certaines de ces classes fournies, comme expliqué dans la suite. Il est conseillé de lire le code pour avoir une idée de son fonctionnement. Pour cela, vous pouvez commencer avec la classe principale.

Sachez que le code contient des concepts avancés que vous n'avez peut-être pas encore vus, comme des `Maps`, des listes chaînées ou de la réflexion.

★ Travail à réaliser

Il est recommandé de lire une première fois l'ensemble des consignes avant de commencer ! Le projet est composé de nombreuses (mais généralement petites) tâches que vous devez compléter.

Les sections marquées d'une étoile (*) sont plus conséquentes et plus complexes donc pensez à organiser votre temps en fonction.

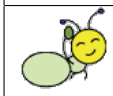
Nourriture et moissonneuses

Pour le moment, il n'y a pas de coût de déploiement pour aucun type de fourmi, donc il n'y aucun défi dans le jeu. Il vous faut ajouter des coûts en nourriture.

— Remarquez que les fourmis ont un coût en nourriture de 0. Vous pouvez redéfinir cela dans les sous-classes (`ThrowerAnt` et `HarvesterAnt`). Étant donné que la variable de coût de nourriture (`foodCost`) est protégée (`protected`), il est facile de le faire.

— Si vous allez plus loin dans les consignes, vous verrez que toutes les fourmis auront des coûts en nourriture et des valeurs d'armure différentes. Comme ces valeurs seront toutes les mêmes au départ, elles seront spécifiées dans le constructeur pour chaque type de fourmi. Étant donné que nous devons tout de même appeler le constructeur de la classe parente de tous les types de fourmis, il serait intéréssant d'avoir un constructeur de la classe parente qui inclut ces deux valeurs pour ainsi les initialiser plus facilement. Ajoutez ce type de constructeur.

— Spécifiez le coût en nourriture pour les classes de fourmis `HarvesterAnt` et `ThrowerAnt` (voir tableau ci-dessous), et vérifiez que vous ne pouvez pas déployer une fourmi si vous n'avez pas assez de nourriture !

Classe	Nourriture	Armure
 HarvesterAnt	2	1
 ThrowerAnt	4	1

Vous utilisez maintenant de la nourriture, mais il n'y a aucun moyen d'en produire plus ! Pour remédier à cela, finissez d'implémenter la classe `HarvesterAnt`. En terme d'action (définie dans la méthode `action()`), les fourmis moissonneuses doivent produire 1 de nourriture pour la colonie à chaque tour.

— Vous pouvez appeler la méthode appropriée sur le paramètre `AntColony` pour ajouter cette nourriture.

Essayez de lancer le jeu à nouveau. Quand vous aurez placé une `HarvesterAnt`, vous devriez à présent accumuler de la nourriture à chaque tour. Vaincre les abeilles est à nouveau possible avec la configuration par défaut du jeu !

Fourmis murailles

Maintenant vous allez créer votre première nouvelle fourmi. Vous allez ajouter de la protection à votre glorieuse colonie en implémentant la classe `WallAnt` qui est une fourmi qui ne fait rien à chaque tour mais qui possède une valeur d'armure forte.

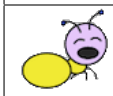
Classe	Nourriture	Armure
 WallAnt	4	4

— N'oubliez pas de créer la classe `WallAnt` dans le package `ants`. Faites aussi attention de bien nommer vos fourmis correctement (la mécanique du jeu s'appuie sur les noms spécifiques des fourmis pour fonctionner).

— La classe `WallAnt` aura besoin d'inclure la méthode `action()` (pourquoi ?) mais elle n'a pas besoin de faire quoi que ce soit.

Fourmis affamées

Implémentez une nouvelle unité offensive appelée la fourmi affamée (`HungryAnt`) qui dévorera une abeille au hasard à la même place, tuant instantanément l'abeille ! Cependant, après avoir avalé une abeille, la fourmi doit passer 3 tours à la digérer avant de pouvoir en manger de nouveau.

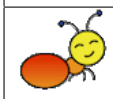
Classe	Nourriture	Armure
 HungryAnt	4	1

— Vous pouvez implémenter cette fonctionnalité en déterminant la méthode `action()` que la fourmi soit en train de digérer ou non. Vous pouvez utiliser des variables d'instance pour garder une trace du temps que la fourmi a passé à digérer.

— Vous pouvez déterminer combien de dégats doit être fait à une abeille pour la tuer en utilisant la méthode `getArmor()`.

Fourmis de feu

Implémentez à présent la fourmi de feu (`FireAnt`). Une fourmi de feu ne réalise aucune action à chaque tour. Cependant, elles possèdent une compétence spéciale : quand la valeur d'armure d'une fourmi de feu atteint zero (ou moins), les valeurs d'armure de toutes les abeilles au même endroit dans le tunnel (`Place`) que la fourmi de feu sont réduites du montant de l'attribut de dégat (`damage`) de la fourmi (3 par défaut).

Classe	Nourriture	Armure
 FireAnt	4	1

- Pour implémenter cela, vous devez rédéfinir la méthode `reduceArmor()` de la fourmi pour y ajouter cette fonctionnalité. Assurez-vous d'appeler la version de la méthode de la classe parent pour conserver le fonctionnement de base.
- Assurez-vous de donner à votre fourmi de feu une variable de dégat (`damage`) pour pouvoir la modifier facilement ; ne codez pas en dur la valeur des dégats dans la méthode `reduceArmor()` !
- Regardez la classe `Place` pour voir les méthodes qui vous permettront d'accéder à la liste des abeilles à blesser.

Une fois la classe `FireAnt` implémentée, assurez-vous que celle-ci fonctionne et testez votre programme en jouant une partie ou deux ! Une fourmi de feu devrait détruire toutes les abeilles au même endroit du tunnel que votre fourmi lorsque celle-ci meurt. Vous pouvez démarrer une partie avec 10 nourritures en spécifiant la valeur appropriée dans la méthode principale de la classe `AntsVsSomeBees`.

Eau*

Maintenant que vous avez ajouté le feu, nous allons ajouter l'eau ! Pour rendre les choses plus intéressantes, vous allez ajouter un nouveau type de `Place`, représentant un tunnel rempli d'eau dans la colonie. Notez que cela sera une des tâches les plus complexes de votre travail, car vous serez ammenés à modifier certaines parties du programme coeur (`core`).

- Créer une nouvelle classe `Water` qui sera une sous-classe de `Place` (assurez-vous de créer cette classe dans le paquetage `core`).
- Vous aurez besoin de modifier `AntColony` afin qu'il inclut `Water` parmi les endroits/places possibles. Dans le constructeur, modifiez les boucles afin qu'au lieu de créer un nouvel objet `Place`, cela crée périodiquement un nouvel objet `Water` en fonction du paramètre `moatFrequency`.
  - Astuce : ajoutez une simple instruction `if` pour déterminer si le tunnel doit être une `Place` ou de l'eau (`Water`). Vous devrez être capable d'instancier de nouveaux objets `Water` en utilisant les mêmes types de paramètres que le code actuel pour créer une nouvelle `Place`. Assignez le nouvel objet `Water` à la variable `curr` et tout doit fonctionner.
  - Vous devrez créer un objet `Water` tout les `moatFrequency` places. Vous pourrez faire cela en vérifiant si chaque variable step est divisible par la valeur cible. Par exemple, si vous voulez inclure `Water` toutes les troisièmes places, vous pouvez placer `Water` en 0, 3, 6, etc. (Mais réellement, vous devrez ajouter 1 pour que ce soit les places 1, 4, 7, etc).
  - Cela peut être délicat – rappelez vous de l'opérateur modulo `%` !
  - Rappelez vous aussi de ne pas inclure d'eau si `moatFrequency` vaut 0 !
- Ensuite, vous aurez besoin de modifier la classe `AntGame` afin que l'eau soit visible. Cela peut être difficile, car beaucoup de choses sont gérées dans `AntGame` !
  - Regardez la méthode qui dessine une colonie. Dans cette méthode, vous devriez voir que la "bordure" de la place est dessinée comme un simple rectangle. Vous pouvez dessiner l'eau (`Water`) en remplissant simplement un rectangle bleu avant de dessiner cette bordure !
  - Vous aurez besoin de déterminer si une `Place` correspond à de l'eau (rappelez-vous que le polymorphisme signifie que `Water` est également une `Place`). C'est le bon endroit pour utiliser l'opérateur `instanceof`.

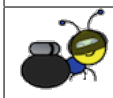
— Testez si l'eau apparaît effectivement dans le jeu !

Une fois l'eau ajoutée, vous devrez vous assurer que les insectes y réagissent. Seuls certains insectes (`watersafe`) peuvent être placés sur une place `Water`.

— Ajoutez un nouvel attribut `watersafe` à la classe `Insect` pour conserver cette propriété. Cet attribut doit être à faux par défaut.
— Vous aurez probablement besoin d'une méthode d'accès (un getter) pour cet attribut.
— Comme les abeilles peuvent voler, leur attribut `watersafe` sera vrai, redéfinissez la valeur par défaut dans ce cas.
— Une fois que cela fonctionne, vous ne devriez plus pouvoir placer des fourmis sur des points d'eau !

Fourmis sous-marines

Créez un nouveau type de fourmis qui peuvent être déployées sur des points d'eau ! Ajoutez une fourmi `ScubaThrowerAnt` qui est un type de `ThrowerAnt` qui ne craint pas l'eau (`watersafe`), mais qui est identique à la classe parent sinon.

Class	Food	Armor
 ScubaThrowerAnt	5	1

Fourmis ninja

Ajoutez une fourmi `NinjaAnt`, qui endommagent toutes les abeilles (`Bees`) qu'elle rencontre, mais qui n'est jamais vue par celles-ci (donc permettant les abeilles de la traverser).

Class	Food	Armor
 NinjaAnt	6	1

— Une fourmi `NinjaAnt` ne peut pas être attaquée par une abeille car elle est cachée, ni ne peut bloquer le chemin d'une abeille car elle la survole. Pour implanter ce comportement, ajoutez un nouvel attribut à la classe `Ant` qui indique si une fourmi bloque le chemin d'une abeille (ce qui sera faux pour une fourmi `NinjaAnt`).
— Modifiez la méthode `isBlocked()` de la class `Bee` afin qu'une abeille ne soit pas bloquée si une fourmi ne la bloque pas. Maintenant, les abeilles devraient pouvoir survoler correctement les fourmis ninja.
— Enfin, faites en sorte que `NinjaAnt` endommage toutes les abeilles qui la survolent. La méthode `action()` de NinjaAnt doit endommager toutes les abeilles qui sont sur la même place au travers de la valeur damage (qui doit valoir 1 par défaut).

Comme challenge, essayez de gagner un jeu par défaut en utilisant uniquement des fourmis `HarversterAnt` et `NinjaAnt` !

Fourmis boucliers*

Pour l'instant, vos fourmis sont fragiles. Nous aimerions leur donner un moyen de tenir plus longtemps sous l'assaut des abeilles. C'est là qu'entre en scène la fourmi bouclier (`BodyguardAnt`).

Class	Food	Armor
 BodyguardAnt	5	2

La fourmi bouclier diffère d'une fourmi normale car elle peut occuper la même `Place` qu'une autre fourmi. Lorsq'une fourmi bouclier est ajoutée sur l'emplacement d'une autre fourmi, elle la protège des dégats. Les attaques doivent impacter la fourmi bouclier en premier, puis l'autre fourmi une fois que la fourmi bouclier a péri.

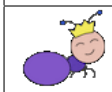
L'implantation de cette fourmi se fait en plusieurs étapes :

- Normalement, seule une fourmi peut occuper une `Place` à un moment donné. La première chose à faire est de changer cela. Il faut pouvoir définir un nouveau type de fourmi qui est capable de "contenir" une autre fourmi. Il est envisageable d'avoir besoin un jour de plusieurs types de "fourmis contenantes" (par exemple des sous-classes d'autres types de fourmis). Ce comportement doit être spécifié en utilisant une interface nommée `Containing`.
  - L'interface `Containing` doit supporter trois méthodes : la capacité d'ajouter l'insecte contenu (qui doit indiquer si l'insecte a bien été ajouté ou non), la capacité de supprimer l'insecte contenu (qui doit indiquer si l'insecte a bien été supprimé ou non), et la capacité d'obtenir l'insecte contenu.
- Maintenant,vous avez besoin de modifier la classe `Place` afin qu'elle puisse recevoir une fourmi `Containing` qui peut elle-même contenir une autre fourmi. Vous devez modifier la méthode d'ajout d'une fourmi de la manière suivante :
  (a) Si la fourmi occupant déjà la `Place` est une `Containing`, ajouter la nouvelle fourmi à celle `Containing`.
  (b) Si la fourmi que vous essayer d'ajouter est une `Containing`, prendre la fourmi déjà en place, ajouter celle-ci à la `Containing`, et ajouter enfin la fourmi `Containing` à la place de la première.
  (c) Si la `Containing` ne peut pas contenir la fourmi spécifiée (`addContenantInsect()` retourne faux), afficher la même erreur que précédemment.
  (d) Si aucune des fourmis n'est une `Containing`, afficher la même erreur que précédemment.
- Vous aurez également besoin de modifier la méthode `removeInsect(Ant)` de telle sorte que, si la fourmi à supprimer est à l'intérieur d'une `Containing`, elle soit supprimée depuis celle-ci. De même, si la fourmi à supprimer est une `Containing`, celle-ci est supprimée et si elle contient une autre fourmi, cette dernière prend sa place.
- Vous aurez également besoin de mettre à jour la méthode `getAllAnts()` de `AntColony` de telle sorte qu'à la fois la fourmi `Containing` et la fourmi contenue soient incluses dans la réponse.
- Vous devrez aussi vous assurer que les deux fourmis apparaissent dans le jeu. Pour ce faire, vous devrez modifier la méthode `drawColony()` de `AntGame` une nouvelle fois. Après avoir vérifié si une `Place` a une fourmi, vérifiez si cette fourmi est une `Containing` et, si tel est le cas, dessinez la fourmi contenue en plus de la `Containing`.
  - Vous pouvez utiliser les mêmes paramètres de positionnement pour la méthode `g2d.drawImage()`. Dessinez simplement la fourmi contenue en premier (pour qu'elle apparaisse à l'arrière), et ensuite la `Containing`.
- Une fois tout préparé, vous pouvez créer la classe `BodyguardAnt` ! Cette classe doit bien sûr implanter l'interface `Containing` !
  - Pour avoir accès à la fourmi contenue dans une `BodyguardAnt`, utilisez une variable d'instance. Elle sera initialisée à null pour indiquer qu'aucune fourmi n'est actuellement protégée. Veillez à faire une encapsulation propre (variable privée !).
  - Vous devrez vous assurer que la fourmie contenue continue d'effectuer ses propres actions. Surchargez la méthode `action()` du `BodyguardAnt` en fonction.

Soyez rigoureux lors de l'élaboration de cette fourmi bouclier. Elle touche en effet de nombreuses parties du code qui peuvent produire de nombreuses erreurs.

La Reine*

Pour finir, vous rendrez possible le positionnement de la reine elle-même dans une galerie.

Classe	Nourriture	Armure
 QueenAnt	6	1

La reine (`QueenAnt`) est résistante à l'eau (`waterproof`) et attaque à distance (comme `ScubaAnt`, que vous devriez utiliser en tant que classe parente). La reine galvanise les fourmis à proximité. En effet, elle double les dommages causés par les fourmis voisines (Celles de la place d'entrée et de sortie).

- Pour vous assurer que vous ne doubliez pas la même fourmi deux fois, vous pouvez conserver une liste des fourmis qui ont déjà été galvanisées.
- Vous pouvez avoir besoin d'ajuster l'héritage de l'attribut `damage` afin d'interagir avec n'importe quelle fourmi, quelque soit son type. Vous pouvez également écrire une interface `Damaging`, avec des accesseurs et modificateurs appropriés.

Un grand pouvoir implique de grandes responsabilités... La reine est soumise à trois règles spéciales :

(a) Si une abeille entre sur la place occupée par la reine, alors les abeilles gagnent immédiatement la partie. Le jeu termine, même si la reine est protégée par une fourmi bouclier (`BodyGuardAnt`). Les abeilles gagent également dans le cas où l'une d'entre elles atteint la fin d'une galerie (Là où réside, en temps normal, la reine).
   - La méthode `queenHasBees()` de la classe `AntColony` indique si l'une des abeilles a atteint la reine. Ceci est déterminé en vérifiant si `this.queenPlace.getBees().length > 0`. Actuellement, la variable `queenPlace` est une place ordinaire. Implémentez `QueenPlace`, une sous-classe de `Place`. Cette dernière représente concrètement deux places : Où est actuellement la reine, et la place `queenPlace` qui termine toutes galeries.
     - Indice : Ce devrait être une place qui en contient une seconde ; cette seconde place fait référence à la position actuelle de la reine.
     - Vous aurez besoin de redéfinir l'accesseur `getBees()` afin qu'il renvoie les abeilles de la première et de la seconde place.
   - Dans la méthode `action()` de `QueenAnt`, mettez à jour l'attribut `queenPlace` de la colonie, avec une nouvelle instance de `QueenPlace` comprenant la place courante de la reine.
     - Il est nécessaire d'ajouter un accesseur et un modificateur pour l'attribut `queenPlace`.

(b) Il ne peut y avoir qu'une vraie reine en jeu. Toutes les autres reines sont des imposteurs et devraient mourir instantanément (armure réduite à 0) avant d'engager une action (Elle n'attaque pas et ne galvanise pas les fourmis voisines). De plus, elles ne devraient pas affecter l'attribut `queenPlace` de la colonie.
   - Pour ce faire, vous pouvez utiliser une variante du patron de conception Singleton [1] (Vous l'aborderez certainement en génie logiciel).
   - Son implémentation se résume à conserver le nombre de fois que la classe `QueenAnt` a été instanciée. Vous devrez utiliser une variable de classe (statique). Chaque fois qu'une reine est créée, incrémentez cette variable de un. Vous ne devriez pas avoir besoin de parcourir la colonie de place en place pour trouver les imposteurs.

(c) La vraie reine ne devrait pas pouvoir être supprimée ; toutes tentatives de suppression, ne devrait avoir aucun effet.
   - Idée : Vous pourriez écrire une interface que la classe `QueenAnt` et d'autres pourrait implémenter et utiliser un test de type pour déterminer si l'insecte est supprimable.

Vous êtes arrivé·e·s à la fin ! Alors ? Parvenez-vous à battre ces abeilles dans le mode fou (insane-mode) ?

1. Voir `https://fr.wikipedia.org/wiki/Singleton_(patron_de_conception)`

★ Bonus

Il y a de nombreuses manières d'enrichir ce jeu !

Nous vous avons fourni des images (et leur support dans l'interface graphique) pour de nouveaux types de fourmi. Vous pouvez également concevoir vos propres types de fourmi !

Classe	Nourriture	Armure	Description
 SlowThrowerAnt	4	1	Applique un effet de ralentissement de trois tours. Une abeille ralentie effectue une action après en avoir passé 2.
 StunThrowerAnt	6	1	Applique un effet d'étourdissement pour 1 tour. Une abeille étourdie passe son tour.
 ShortThrowerAnt	3	1	Une ThrowerAnt qui lance des feuilles aux abeilles situées au plus à 2 places de la fourmi.
 LongThrowerAnt	3	1	Une ThrowerAnt qui lance des feuilles aux abeilles situées au moins à 4 places de la fourmi.

Vous pourriez aussi créer de nouveaux types d'abeilles (Cette extension peut demander un grand nombre de modifications et d'ajouts).

Les extensions peuvent vous rapporter jusqu'à 2 points de bonus. Assurez-vous toutefois que les fonctionnalités rudimentaires sont implémentées et fonctionnent correctement.