BATTLESHIP

Rapport de fin de projet de programmation orientée objet.
Cours POO 3ème année

POLYTECH MONTPELLIER

Goncalves Lucas IG3 2017 / 2018

SOMMAIRE

Si ce document est en version numérisé, cliquez sur les titres pour être redirigé à chaque section correspondante.

1. Introduction

Ce document détaille l'ensemble de l'architecture et des décisions qui ont été faites pour mener à bien le projet proposé dans le cours de programmation orientée objet. L'ensemble du programme est disponible sur un dépôt GitHub ouvert et accessible au téléchargement. L'ensemble des instructions de récupération et d'utilisation sont disponibles dans la section ci-dessous.

2. Instruction de compilation et d'utilisation

Tout d'abord, si vous n'avez pas téléchargé le projet, cliquez sur le lien suivant : **GitHub du projet**.
Une fois téléchargé, rendez-vous dans le dossier où vous avez enregistré le projet ou dans le répertoire de votre git clone si vous avez utilisé git).

Une fois dans votre répertoire, assurez vous de voir les deux dossiers « fr » et « goncalves » a la base de votre arborescence. Procédez aux commandes suivantes depuis votre terminal :

Compilation du programme de test des IA :

```
1  javac fr/battleship/TestIA.java
```

Compilation du programme de jeu Battleship :

```
2  javac goncalves/lucas/*.java
```

*** Il est important de compiler les deux packages pour qu'ils fonctionnent.**

Lancement du programme de jeu Battleship :

```
3  java goncalves.lucas.Battleship
```

Lancement du programme de test des IA :

```
4  java fr.battleship.TestIA
```

Une fois le programme TestIA exécuté, il créé un fichier proof_IA.csv qui vous permet de voir le résultat du combat de chaque IA sur 100 parties.

3. Architecture

3.1. Les classes principales :

3.1.1. Les Coordonnées (Coord) :

Les coordonnées (classe Coord) représentent le système d'enregistrement des informations. En effet, les coordonnées de bateau ainsi que les tirs sont composés de coordonnées. La coordonnée correspond à une position (x et y) mais aussi à son état (touché ou non). Ainsi, on peut stocker l'ensemble des informations nécessaires au fonctionnement de la partie et les évènements comme « touché » et « coulé » peuvent être calculés à partir des méthodes proposées par cette classe.

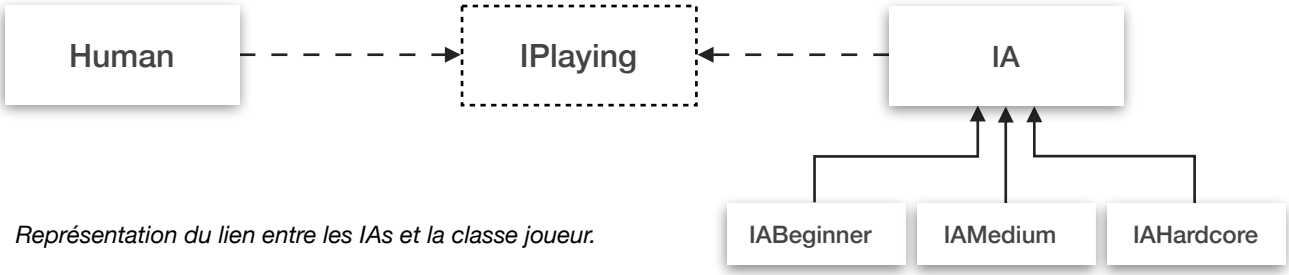
Coord
x : int **y : int** **value : String** **hit : boolean**

3.1.2. Les bateaux (Ship) :

Ship
position[Coord] **destroyed : boolean**

Le bateau est un objet composé de coordonnées (position) et d'un état (détruit). Il implémente un ensemble de fonctions qui permettent de savoir si il a été touché, si il est touché par un tir et/ou si il est détruit.

3.1.3. Les joueurs : (Human & IA)



Représentation du lien entre les IAs et la classe joueur.

Afin de créer l'architecture du programme, l'objectif était de séparer les différents types de joueurs. En effet, chaque joueur (que ce soit Human ou IA) utilise le même système de stockage des informations (même si ce dernier est modifiable pour chaque type de joueur). Pour ce faire, chaque joueur possède un ensemble de bateaux ainsi qu'un ensemble de « Coord ». Chaque Coord représente alors un tir effectué par le joueur. Même si dans cette architecture l'implémentation des IA et des humains est identique, il est tout à fait possible de modifier uniquement l'architecture de l'IA sans affecter le fonctionnement de joueur.

Human & IA
capacity [int] **shot <Coord>** **army <Ship>** **name : String**

Légende :

Néanmoins, afin d'assurer un bon fonctionnement dans le cadre de notre système de partie, chaque entité (Humain & IA) est associée à l'interface IPlaying qui permet de s'assurer que : quelque soit le type de partie jouée, chaque entité aura la possibilité de tirer, créer sa flotte, etc.
C'est à cet instant que la notion de « *separation of concern* » est la plus importante.

Par exemple, si l'on souhaite savoir si le tir d'un joueur J1 a touché un bateau du joueur J2, le fonctionnement sera le suivant : On appelle la fonction isHit() du joueur ***J2.isHit(Coord)***, cette fonction va alors demander à chaque bateau s'il a été touché à cette coordonnée ***Ship.isHit(Coord)*** . Le bateau va alors s'assurer que la coordonnée transmise en paramètre est égale à une des coordonnées qui le compose ***Coord.equals(Coord).*** Seule la classe Coord pourra dire si la coordonnée du bateau est bien égale à celle entrée en paramètre.

3.2. Les avantages de cette architecture :

Cette architecture permet de placer dans un endroit spécifique l'ensemble des fonctions proposées. L'objectif étant d'avoir la possibilité de modifier entièrement la structure de la classe tout en conservant les méthodes qui sont associées sans affecter le reste du programme. Par exemple, les bateaux et les tirs étaient initialement stockés, pour le joueur, dans des matrices. Le passage à un stockage sous forme de liste n'a affecté que l'implémentation des méthodes au sein de la classe, et non pas l'ensemble du programme qui fait appel aux bateaux et aux tirs du joueur.

Human et IA implémentent IPlaying, l'utilisation d'interface permet de dissocier ce qu'une classe « est » de ce qu'une classe « peut faire ». On peut alors utiliser cette interface pour faire jouer n'importe quel type de joueur.

3.3. Les inconvénients de cette architecture :

Bien que cette architecture permette de séparer clairement les accès aux données de chaque classe, elle possède plusieurs inconvénients :

• Tout d'abord les règles. Les capacités sont écrites « en dur » pour chaque type de joueur et doivent être corrigées à la main. Dans l'ancienne version (v1.1.0 du git), la mise en place d'une classe de règle plus flexible était possible, ce qui permettait de choisir le nombre de bateaux et la taille de la grille. (Cette implémentation a par la suite été supprimée, car elle utilisait un ensemble de caractéristiques qui nécessitait de rester privées.)

• L'ensemble des IAs utilisent la même structure. En effet dans le cadre de ce projet, les trois IAs héritent toutes de la classe abstraite IA. Par conséquent, si l'on souhaite modifier la structure d'une des IAs, il faut intégrer l'ensemble des attributs au niveau inférieur.

4. Les intelligences artificielles

Dans cette section, nous aborderons les différentes caractéristiques apportées aux IAs du jeu. Nous verrons en quoi diffèrent leurs niveaux de « compétence » de tir et comment leur stratégie d'attaque permet de battre leur adversaire (IA) de niveau inférieur.

4.1. Le placement des IAs :

Le placement est un fonctionnement commun à toutes les IAs (dans cette version de Battleship). Les trois IAs utilisent un placement aléatoire qui définit : une première coordonnée (Lettre + chiffre), un sens (Vertical ou Horizontal) et une seconde coordonnée (Lettre + chiffre). La seconde coordonnée est calculée en fonction des tailles de bateaux qu'il est encore possible de placer et du sens définit. L'IA va alors parcourir la liste des tailles de bateaux disponibles et en fonction, définir la seconde coordonnée. Une fois la seconde coordonnée créée, elle va parcourir l'ensemble des coordonnées des bateaux déjà placés pour vérifier qu'elle ne chevauche aucun bateau. Une fois la vérification terminée, elle ajoute le bateau à sa flotte.

4.2. La puissance des IAs :

Les trois IAs utilisent un fonctionnement de tir différent selon leur niveau, c'est pourquoi chacune d'elle peut vaincre les IAs de niveau inférieur. L'IA dite Beginner (niveau le plus faible) utilise une fonction de génération de tir aléatoire (lettre + chiffre) et renvoi la coordonnée de ce tir au jeu. Elle ne procède pas à une vérification et peu donc tirer plusieurs fois au même endroit.

L'IA dite Medium (niveau intermédiaire) utilise un système de tir similaire à l'IA Beginner mais elle implémente en plus une vérification de tir. Une fois que la fonction de tir a généré une coordonnée, l'IA vérifie que le tir qu'elle vient de créer ne fait pas partie d'un ensemble de tirs déjà effectués. Cette vérification permet d'éviter à l'IA de tirer deux fois au même endroit et par conséquent de vaincre facilement l'IA Beginner de niveau inférieur. Néanmoins, d'après l'article posté sur le site DataGenetics, ce fonctionnement nécessite 96 coups en moyenne pour finir la partie. Cela permet tout de même de vaincre l'iA de niveau 0 qui peut nécessiter plus de 100 coups pour gagner.

L'IA Hardcore (niveau supérieur) utilise le principe de « Walkthrough » ou « parcours pas-à-pas ». L'objectif est d'effectuer le système de tir de l'IA Medium en proposant des tirs aléatoires uniques et de commencer un parcours de tir dès lors qu'un bateau est touché. Pour ce faire, le parcours de découverte de bateau s'effectue de la façon suivante ; si le tir est « touché », l'IA bloque la position du tir et essaye de tirer à la position se situant sur la droite. Elle parcourt alors dans cette direction jusqu'à avoir « coulé » le bateau, être arrivée au bout de la grille ou avoir déjà tiré à la prochaine position. Si le bateau n'est pas « coulé », elle revient au premier tir « touché » et repart dans la direction opposée. Une fois que le bateau adverse est « touché », elle « déverrouille » son blocage de parcours et reprend la saisie de tir aléatoire.

Cet algorithme de parcours permet de vaincre l'IA Medium, car son fonctionnement utilise le verrouillage de position pour détruire un bateau, ce qui permet d'éviter un nombre considérable de coups pour gagner.

5. Post Mortem

5.1. Avantages du projet :

Ce projet d'initiation au langage objet développé en Java m'a permis de découvrir l'utilisation d'un modèle de conception de programmation objet basée sur des classes. Le Java est un langage de « typage fort » vraiment très interessant. Ce projet permet également de se poser les « bonnes » questions et de remettre en considération ses choix d'architecture et de conception. S'interroger sur ces choix d'implémentation permet de prendre du recul sur le travail réalisé et d'adapter sa vision au vrai problème que l'on essaye de résoudre.

5.2. Les inconvénients du projet

L'utilisation de classes et de programmation objet force à mettre en évidence la séparation entre ce que la classe doit partager et rendre accessible et ce qu'elle doit garder pour elle. Cette distinction des rôles de chaque classe est plutôt complexe à mettre en place dans le cadre d'un premier projet de POO et a demandé plusieurs réajustements. La plus grande difficulté n'est pas liée au projet en lui-même, mais à l'espacement des cours qui ne permet pas de concevoir le projet de façon « optimisée », car de nouvelles notions sont à implémenter au fur et à mesure (Interfaces et Exceptions). En effet, même si l'utilisation d'interface est nécessaire dans le bon fonctionnement du projet, la mise en place d'Exception aurait permis une meilleure implémentation de certains points du projet qui ont été ajustés pour avoir un programme robuste par un ensemble de vérifications diverses.

5.3. Leçons assimilées

L'importance de la notion de « separation of concern » : Les classes doivent utiliser leurs attributs et permettre aux autres classes (quand c'est nécessaire) d'avoir accès aux informations utiles sans indiquer la façon dont elles sont implémentés.

L'héritage n'est pas un outil de factorisation du code : Utiliser l'héritage ne doit pas permettre de factoriser des fonctions mais de définir les caractéristiques d'un objet. Pour cela il faut utiliser les interfaces pour définir des contrats avec les classes concernées. En effet, bien que l'Humain et les IA semblent être des joueurs, l'IA et l'humain implémentent le comportement d'un joueur, ils n'en héritent pas.

Il est important de prévoir l'ensemble des spécificités du projet avant de commencer à travailler. On ne peut pas écrire de programme si l'on a pas de vision sur l'intégralité de ce que l'on veut faire. Une fois notre idée d'architecture pensée, on peut concevoir le programme.