Université de Caen Normandie
UFR des Sciences
Département Informatique

2ème année de licence d'informatique

Bataille navale

Complément de POO

Rapport de projet

Antonin Boyon
Thomas Lalong
Quentin Legot
Arthur Page

Table des matières

1 Comment lancer le jeu

Pour utiliser notre jeu, il est nécessaire de fournir des arguments de lancement au programme, vous pouvez le faire directement depuis le terminal en faisant
java -jar battleship.jar <joueur 1> <joueur 2> [nogui].

<**argument**> correspond à un argument obligatoire.

[argument] correspond à un argument facultatif
[nogui] permet de lancer le jeu sans interface graphique, le jeu se jouera donc à l'aide du clavier en console.

joueur 1 et *joueur 2* correspondent à l'instance du joueur que vous voulez utiliser, les valeurs acceptées sont :

— **Human** pour un joueur humain qui contrôlera le jeu au clavier et à la souris.

— **Random** qui placera ses navires et jouera ses coups aléatoirement.

Vous pouvez aussi lancer le programme avec Apache Ant en faisant *ant run*, cela lancera le programme avec en joueur 1 un humain et en joueur 2, un joueur aléatoire, ainsi qu'une interface console.

Suivez ensuite les indications à l'écran.

Si vous avez lancé la vue terminal, consultez la section 3.1, si vous avez lancé avec la fenêtre graphique, consultez la section 3.2

2 Modèle MVC

2.1 Structure du projet

Notre projet suit une structure de données Modèle-Vue-Contrôleur, le modèle est par conséquent quasiment indépendant du reste du projet. Lors d'une entrée clavier ou souris, le modèle demande à la vue de faire appel au contrôleur car la méthode d'accès aux entrées diffère en fonction de la vue utilisée.

Les noms des paquets ont été choisis pour mettre en avant l'utilisation du modèle MVC, cependant ils auraient pu être renommés avec des noms plus communs : *model* aurait pu être renommé en **game** et *control* en **listener**.

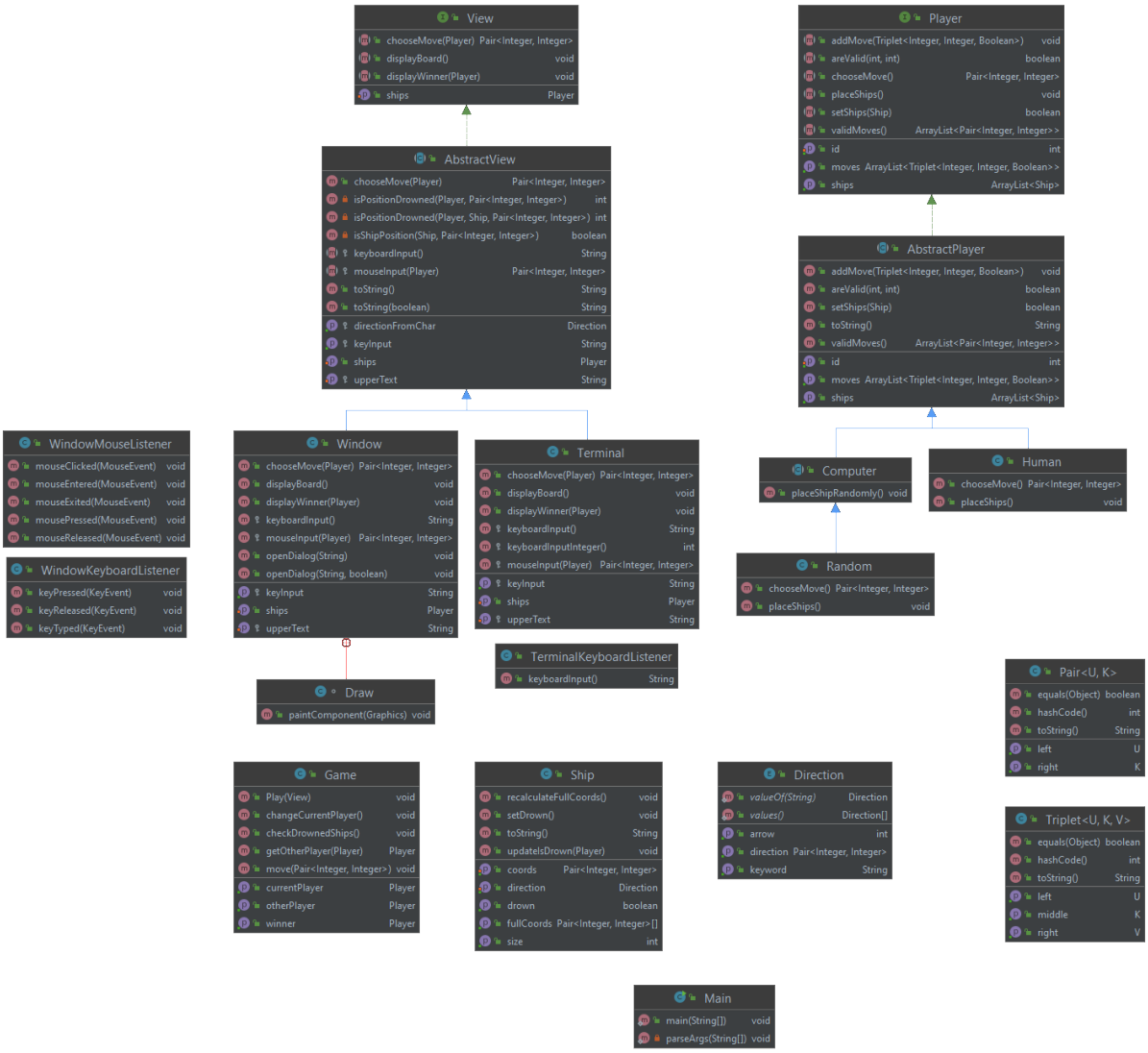

FIGURE 2.1 – Diagramme de classe

Légende du Diagramme :
Un trait rouge représente une classe intérieure (inner class), elle définit dans la classe indiquée par le (+).
Un trait bleu indique un héritage, ces classes héritent des propriétés de leur classe mère.
Un trait vert indique l'implémentation d'une interface dans une classe.

2.2 L'organisation en API de notre projet

La vue et le modèle des joueurs de notre projet sont organisés de cette manière :

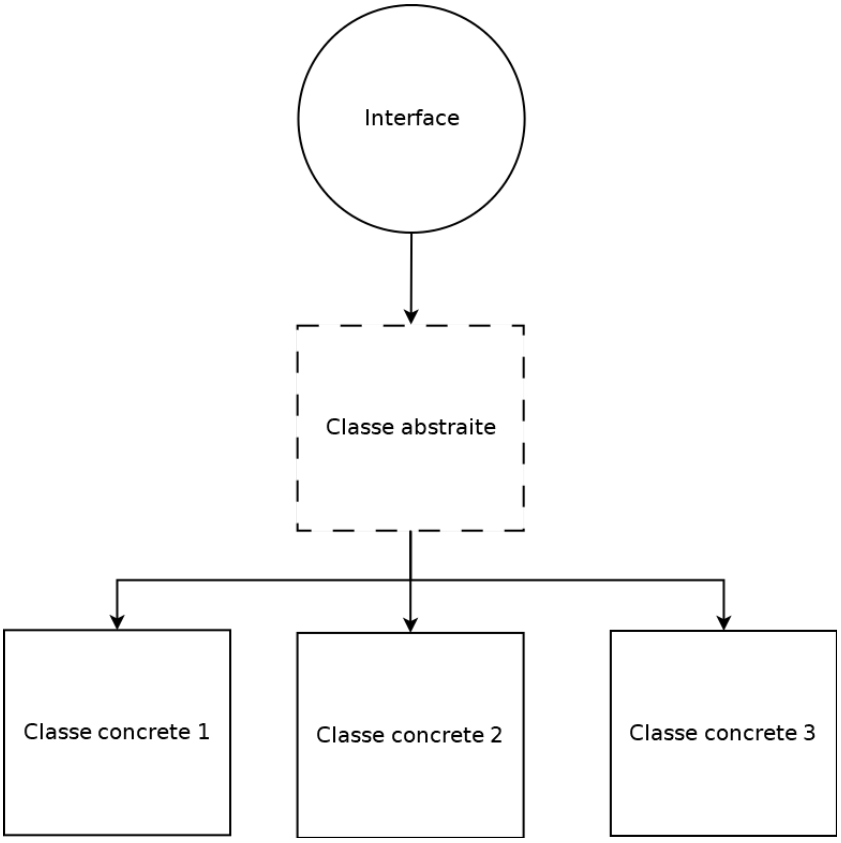

FIGURE 2.2 – Modèle d'API utilisé

Dans la logique suivie par le modèle MVC, nous voulions un code qui tire parti des outils offerts par Java afin d'éviter le *hard code*[1].
Nous voulions qu'il soit modulable afin de pouvoir, par exemple, rendre possible l'implémentation d'un algorithme de résolution par une personne qui serait extérieur au projet. Il lui suffit pour cela d'implémenter une nouvelle classe qui héritera de la classe **Computer** et d'y ajouter ses algorithmes.
S'il veut retravailler les vues, il peut implémenter une nouvelle classe qui implémentera l'interface **View** ou qui héritera de **AbstractView**.
Il n'aura ensuite plus qu'à ajouter son nouvel objet Joueur dans une liste située dans *Main* s'il a créé un nouveau joueur.
L'ajout d'une nouvelle vue lui demandera cependant quelques modifications supplémentaires de la méthode *Main#parseArgs(String[])*.

1. Le hard code est un code non modifiable/modulable.

3 Interactions avec les différentes vues

3.1 Terminal

L'utilisation du mode "Terminal" se fait directement via la console. Lorsqu'il y a un utilisateur (qui n'est pas une IA) ou plus, le programme demandera de placer les navires en choisissant leurs coordonnées (en donnant un chiffre entre 0 et 9 au clavier pour chaque axe) puis leur direction. Si le ou les utilisateurs essaient de placer un navire qui ne rentre pas entièrement dans le plateau ou qui se superpose avec un autre, le programme demandera de recommencer le placement de celui-ci.

Lorsqu'il y a deux utilisateurs, après que le joueur 1 ait fini de placer ses navires, son plateau est bien sur caché laissant la place au joueur 2 afin qu'il place les siens sur son propre plateau.

Une fois tous les navires posés, les plateaux sont masqués et vient le choix du tir. Celui-ci s'effectue d'une manière similaire à celle du placement des navires, c'est-à-dire en entrant les coordonnées au clavier allant de 0 à 9 pour les axes X (abscisses) et Y (ordonnées).

Le terminal a une syntaxe bien précise :

- X indique un navire coulé
- _ indique l'eau ou un navire qui est masqué
- ? indique un coup dans l'eau
- ! indique un coup qui a touché un navire
- . indique vos navires

Le programme se ferme dès qu'un joueur a gagné.

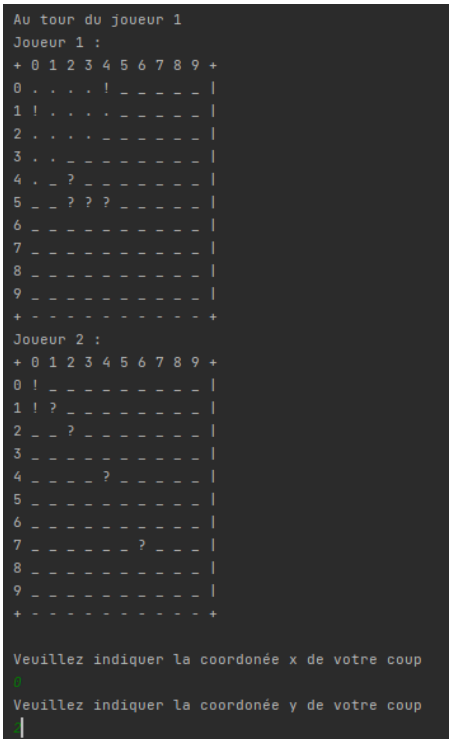

FIGURE 3.1 – Vue terminal

3.2 Fenêtre

L'utilisation du mode "Fenêtre" se fait, quant à lui, via une interface composée de deux grilles (Une pour chaque joueur). Lorsqu'il y a un utilisateur (qui n'est pas une IA) ou plus, le programme demandera de placer les navires en choisissant leurs coordonnées (en cliquant directement sur une des cases de la grille) puis en indiquant leur direction par le biais du clavier (H, B, D et G pour Haut, Bas, Droite, Gauche). Tout comme pour le mode "Terminal", si le ou les utilisateurs essaient de placer un navire qui ne rentre pas entièrement dans sa grille ou qui se superpose avec un autre, le programme demandera de recommencer le placement de celui-ci.

Lorsqu'il y a deux utilisateurs, après que le joueur 1 ait fini de placer ses navires, sa grille est bien sûr masquée, laissant la place au joueur 2 afin qu'il mette en place ses bateaux sur sa propre grille.

Une fois tous les navires posés, la grille du joueur adverse est masquée. Il est ensuite temps de choisir les coups. Ce choix s'effectue d'une manière similaire à celle du placement des navires, c'est-à-dire en cliquant sur une case de la grille, une croix rouge s'affichant si un navire est touché, une croix bleue sinon.

Les navires ont plusieurs couleurs :

— un navire jaune indique que le navire a été placé dans la direction HAUT

— en bleu-vert dans la direction BAS

— en vert dans la direction DROITE

— en bleu foncé dans la direction GAUCHE

Un navire coulé reste affiché à l'écran.

Le programme se ferme dès qu'un joueur a gagné.

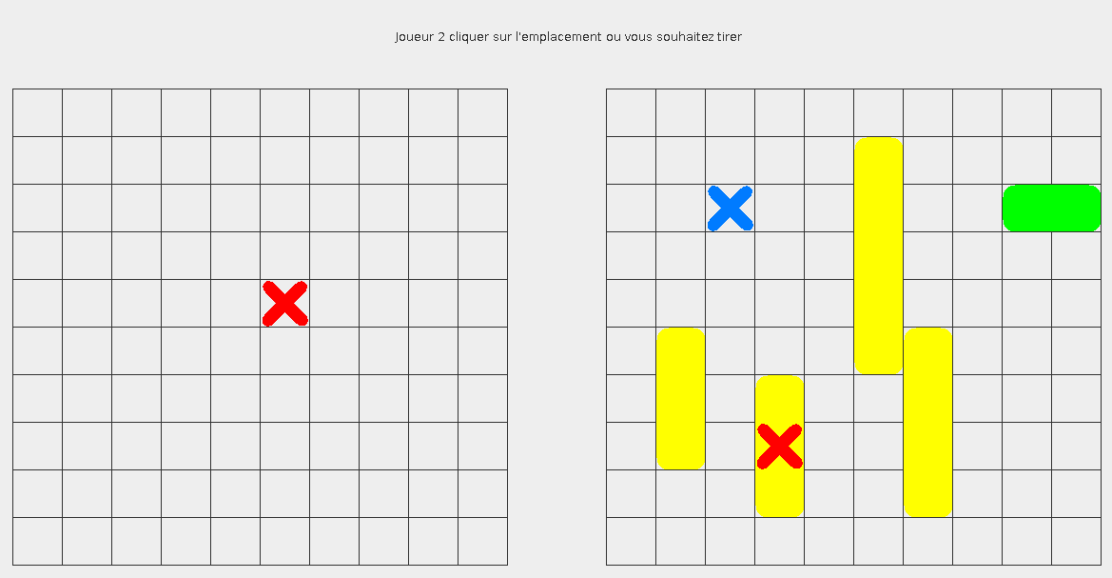

FIGURE 3.2 – Vue fenêtre

4 Améliorations possibles

De nombreuses améliorations sont possibles :

— Composants modèle et vue moins interdépendants

— Mise en place d'une structure complète d'écouteur

— Utilisation de JavaFX au lieu de Swing

Nous trouvons que le modèle de notre projet n'est pas assez indépendant de la vue, ce point aurait sûrement pu être facilité par la mise en place d'un système d'écouteurs d'événements qui aurait permis de mieux exploiter le multi-threading offert par l'utilisation de Swing sur java.

Enfin il aurait sûrement été préférable d'utiliser JavaFX au lieu de Swing. Le développement de ce dernier étant abandonné depuis Java 8 en 2013 au profit de JavaFX, qui offre des fonctionnalités qui auraient pu être intéressantes pour le développement de notre projet : un éditeur de scène graphique, un meilleur support du css, testFX ou encore un modèle MVC plus consistant comparé à Swing.