Université de Caen

Deuxième année de Licence en Informatique

Promo 2019 - Groupe 4A

Rapport de Complément de POO

Bataille Navale

Réalisé par :
Jasmine Chaid Akacem
Oumaima Chammakhi
Hugo Baudin
Mamadou Diallo

Encadré par :
*M*me. Celine Alec

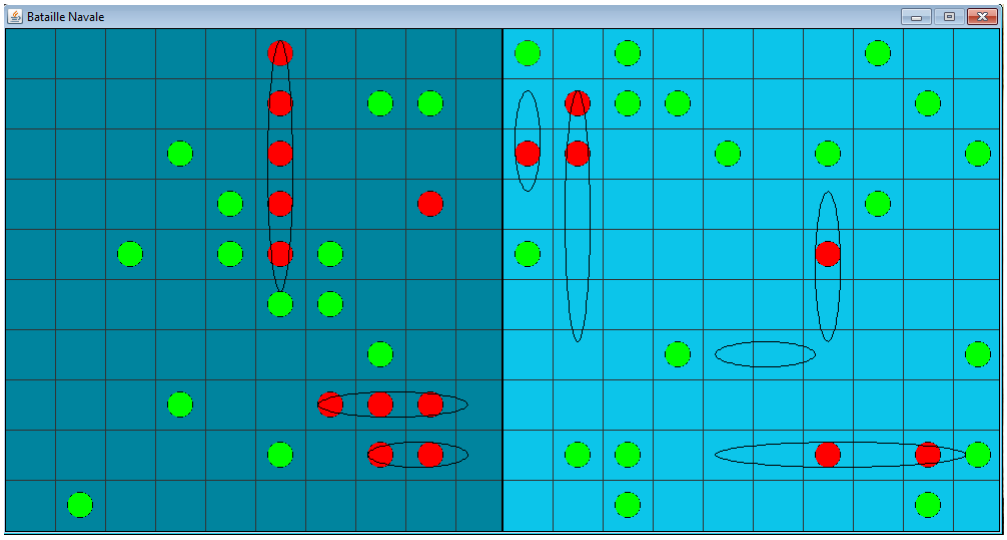
1er Avril 2019

Introduction

Dans le cadre de nos modules de formation en Licence 2 en Informatique, nous sommes amenés à modéliser un jeu de société et à le coder sous Java afin de parfaire notre maîtrise de ce langage. Ce projet fait partie de l'unité d'enseignement de Complément de POO, et est réalisé par groupe de quatre personnes. Nous allons vous présenter notre réalisation.

L'objectif de ce projet consiste à réaliser une application permettant la modélisation du jeu de la Bataille Navale. L'application est accessible via une interface graphique ou via le terminal, et ce selon le choix de l'utilisateur. Le seul mode de jeu disponible permet à un humain de jouer contre l'ordinateur (se comportant d'une manière aléatoire). Il est spécifié que la réalisation du modèle est en MVC.

Le principe du jeu de la bataille Navale consiste en deux joueurs qui s'affrontent, chacun possédant une flotte de 5 Bateaux de différentes tailles. Chaque joueur doit placer ses navires sur une grille tenue secrète et tenter d'atteindre les navires adverses en les touchant. Pour gagner, un joueur doit faire en sorte que tous les navires de son adversaire coulent avant les siens. On considère qu'un navire "coule" si l'ensemble de ses cases a été touché par des coups de l'adversaire.



Dans ce rapport, nous allons détailler l'organisation de notre équipe pour la réalisation de ce projet, l'architecture logicielle retenue et les algorithmes implémentées.

Table des matières

Chapitre 1

Notre réalisation

1.1 Organisation du projet

Le projet est composé de plusieurs parties. Avant de commencer à coder, nous avons d'abord travaillé tous ensemble à la création d'un plan de travail. Un plan qui nous a permis de fixer nos objectifs et de répartir les tâches à réaliser tout en respectant le délai du projet. En organisant nos idées, nous avons ainsi pu vérifier la concordance et la faisabilité de notre projet.

Alors afin d'optimiser le temps de travail, nous avons décidé de nous concentrer en premier lieu sur le modèle ; et de développer le jeu en console. Hugo s'occupa de la classe centrale `Bataille Navale`, qui était prévue pour le déroulement du jeu. Jasmine prit en charge l'implémentation d'une grille de Bataille Navale, et s'occupa aussi des classes de joueurs, avec Oumaima. Mamadou se chargea de donner une représentation objet des bateaux.

Après une première mise en commun, nous pouvions nous concentrer sur la partie interface graphique, même si quelques détails restaient à régler sur la partie console. Jasmine s'occupa de ces détails avec Oumaima tandis que Hugo et Mamadou commençaient à travailler sur l'affichage des grilles de la bataille navale. Enfin, la partie "contrôleur" de l'interface graphique, c'est à dire la possibilité de jouer grâce à la souris, fut conçue par Jasmine.

Chapitre 2

Architecture du logiciel

2.1 Architecture générale du projet

Le logiciel est divisé en quatre packages. Le package `game` englobe les classes qui servent à faire tourner le modèle du logiciel, ainsi que le sous-package `player`. Le contenu package est plus détaillé en section 2.2. Le package `gui` possède les classes permettant l'affichage graphique de la bataille navale. Ces classes fonctionnent conjointement avec celle du package `util` afin que l'affichage soit MVC. Tout comme `game`, nous détaillons son architecture en section 2.3. Le dernier package est `main`, qui contient la classe `Main` qui sert à lancer le jeu et demande à l'utilisateur s'il souhaite lancer le jeu avec la console ou avec l'interface graphique.

2.2 Partie Modèle

Le modèle est consitué de trois classes ainsi qu'un sous-package : le package `player`.

La classe `Mer` permet de représenter un plateau de jeu, une grille avec des bateaux dessus. Pour bien repérer les rôles des joueurs, nous distinguons le propriétaire de la grille, qui possède les bateaux qui sont sur la grille, et l'attaquant, qui veut couler les bateaux. Sur cette grille, nous pouvons accéder aux bateaux via une méthode, les placer, mais également envoyer un coup sur une case. Cette méthode est différente de la méthode `estTouche`. Envoyer un coup permet de spécifier au niveau du modèle que c'est le joueur qui appelle cette méthode (donc il faut mettre à jour), alors que la méthode `estTouche` permet simplement de vérifier si un bateau est touché à la case `coords`.

La classe `Bateau.java` quant à elle, gère les bateaux dans le jeu. Elle dispose les méthodes suivantes : les méthodes getX() et getY(), qui retournent respectivement l'abscisse et l'ordonnée (qui est le point de départ) du bateau qu'on veut créer, la méthode estTouche() permet de diminuer le point de vie d'un bateau quand il est touché, la méthode estCoule() retourne en booléen qui teste si le bateau est coulé (le point de vie dans ce cas est 0) ou pas, la méthode getDirection() qui retourne la direction d'un bateau (horizontale ou verticale) et enfin la méthode getTaille() qui retourne la d'un bateau (nombre de cases qu'il occupe).

La classe `BatailleNavale` orchestre le jeu. Nous utilisons l'attribut `joueurActu` afin de représenter le joueur dont c'est le tour. La méthode `jeu` lance une boucle de jeu qui appel deux méthodes à chaque tour : `jouer`, qui effectue un coup pour le joueur actuel et qui renvoie un message si un bateau a coulé ; et la méthode `changeJoueur`, qui change de joueur actuel. les deux autres méthodes principales de la classe sont `fin` et `gagnant`, qui détermine respectivement si le jeu est fini et qui est le gagnant en vérifiant la liste des bateaux actuels de chaque joueurs.

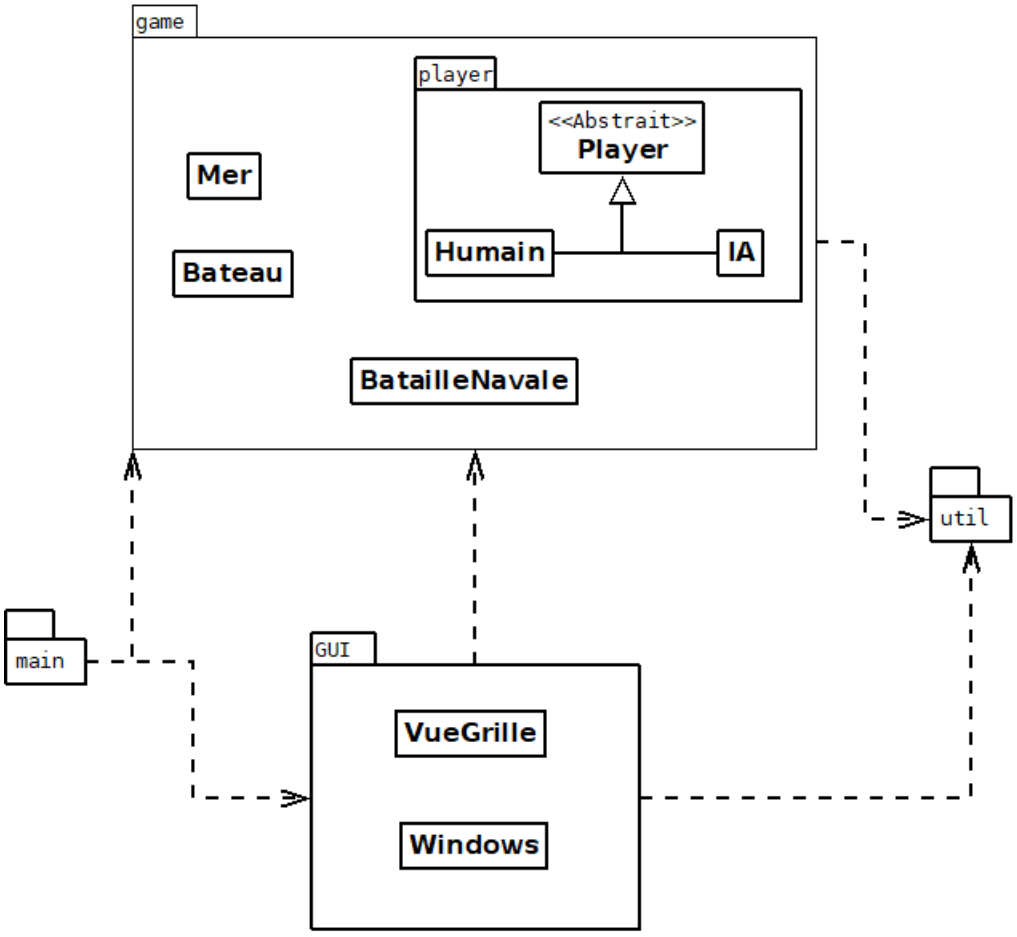


FIGURE 2.1 – Architecture générale

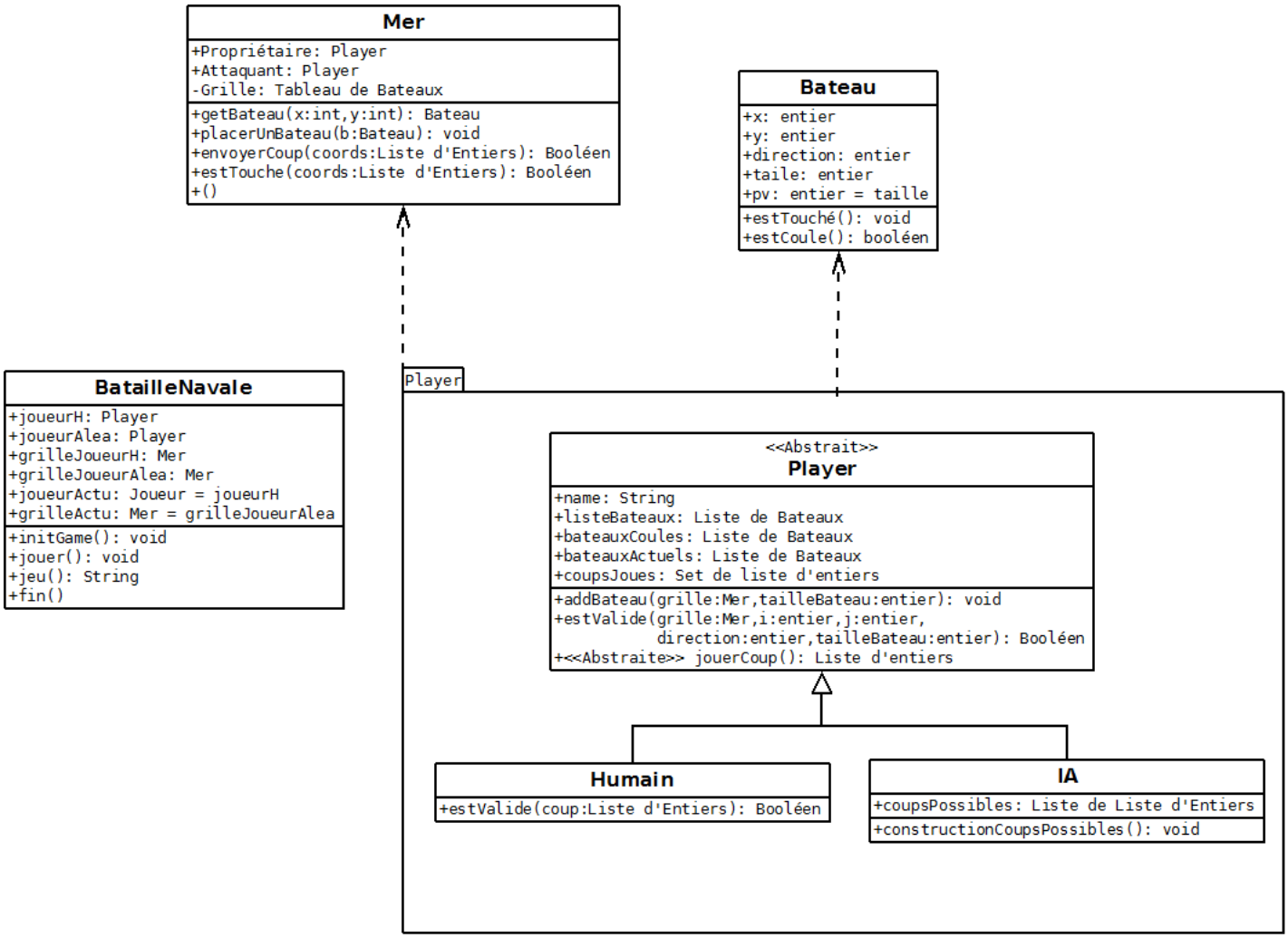


FIGURE 2.2 – Architecture de la partie modèle

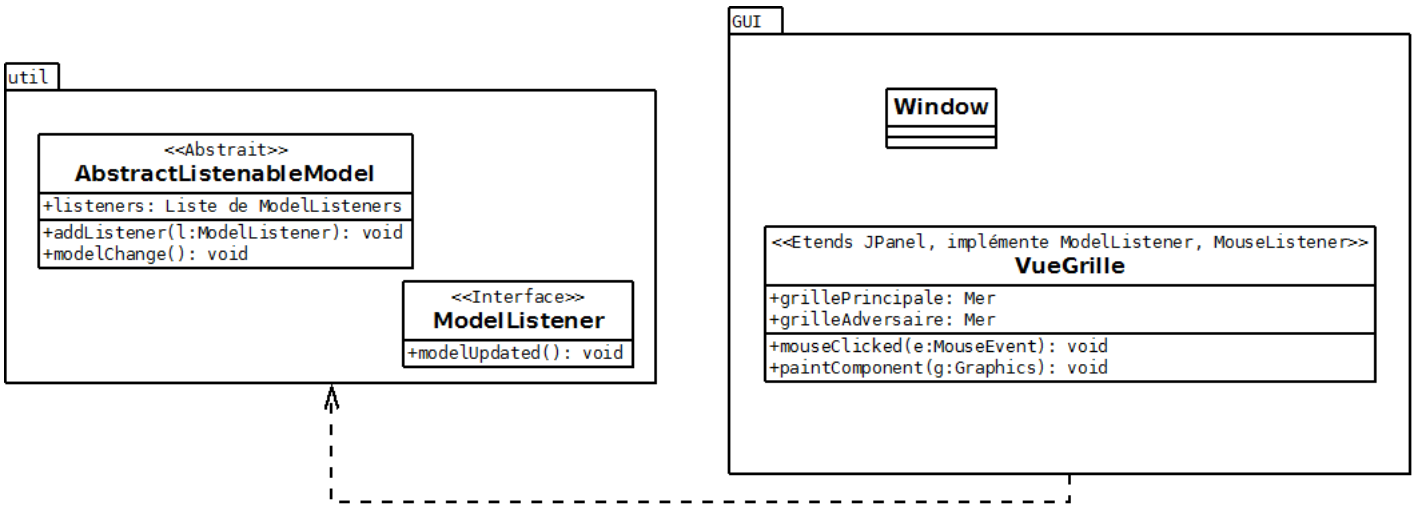


FIGURE 2.3 – Architecture de la partie Vue-Contrôleur

2.3 Partie Vue-Contrôleur

La vue du plateau de jeu est gérée principalement par deux classes : Windows.java qui crée la fenêtre principale et qui est séparée en deux sous-parties qui correspondent aux grilles du jeu, et VueGrille.java qui gère l'affichage du jeu.

À cela, s'ajoutent la classe abstraite AbstractListenableModel.java et l'interface ModelListener.java pour interagir avec la grille. Afin de mettre l'interface à jour selon les actions des joueurs, nous faisons principalement appel à trois méthodes de la classe VueGrille.java.

D'abord, la méthode mouseClicked() récupère les coordonnées du clique de la souris afin d'avoir les coordonnées de la case où l'on a cliqué. Une fois le clique effectué, on fait appel à une méthode de AbstractListenableModel.java qui fera appel à modelUpdate(). ModelUpdate() fait ensuite appel à la méthode repaint() de swing qui elle-même fera appel à paintComponent().

Enfin, paintComponent() efface ce qui est déjà tracé pour redessiner la grille de jeu, ainsi que les coups joués ainsi que celui qui vient d'être joué, pour finir par dessiner les bateaux des joueurs. Les bateaux du joueur Humain sont dessinés en continue tout le long de la partie, tandis que les bateaux de l'I.A. ne se dessinent qu'une fois coulés.

Chapitre 3

Algorithmes

3.1 Coups du joueur aléatoire

Lors de la création du joueur aléatoire, une liste des coups possibles est créée.

Lorsque ce joueur va jouer son coup, elle mélange cette liste avec la méthode shuffle() de la classe Collection. On récupère ensuite le premier item de la liste et on le retire de la liste avec la méthode remove() avant de renvoyer le coup.

3.2 Placement des bateaux

Le placement des bateaux se fait à l'aide de deux méthodes, qui sont `placerUnBateau` dans la classe `Mer`, et `addBateau` de la classe `Player`.

Afin des les ajouter à la grille, on fait appel à `addBateau`, qui génère des coordonnées i et j aléatoirement, ainsi qu'une direction (horizontale ou verticale). On vérifie ensuite, via la méthode `estValide` de la même classe, si on peut placer le bateau (dans la grille, aucun autre bateau sur la trajectoire), sinon, on génère de nouvelle coordonnées. Une fois cela effectué, on créé le bateau, on l'ajoute à la liste du Player et on le place dans la grille avec la méthode `placerUnBateau`.

En fonction de la direction du bateau, nous changeons de colonnes mais pas de lignes, ou inversement, afin de placer une instance de `Bateau` sur chaque case que le bateau doit occuper.

3.3 Affichage de la grille en mode terminal (provisoire)

Pour afficher les coups d'un joueur sur une grille, nous n'avons besoin que de deux structures de données : le Set contenant les coups joués du joueur et la grille de Bateaux.

Pour chaque coordonnée de la grille, nous regardons si elle est contenu dans le Set **coupsJoues**. Si oui, nous regardons dans la grille si il y a une instance de bateau à ces coordonnées. Si oui, alors le coup a touché, sinon, le coup a raté.

Cet algorithme s'effectue en temps constant par rapport à l'exécution du programme, car la taille de la grille de jeu ne change pas. Comme nous pouvons accéder aux bateaux, nous pourrions également les "dessiner" quand le joueur Humain arrive à en couler un. Néanmoins cette méthode n'a pas été reprise pour l'affichage graphique.

Conclusion

La réalisation de ce projet nous permis de mettre en pratique les différentes connaissances acquises autour du langage Java, mais aussi d'acquérir de nouvelles compétences en termes de développement d'interface graphique et de conception d'un modèle MVC. En effet, nous avons pu modéliser et développer une application de jeu de société (Bataille Navale) accessible via une interface graphique entre l'humain et l'ordinateur, mais qui peut également être jouée en utilisant la ligne de commande.

Bien que ce projet répond aux objectifs fixés, quelques améliorations restent possibles pour parfaire son fonctionnement. En effet, il est possible d'implémenter un certain nombre de fonctionnalités supplémentaires pour intervenir directement sur l'interface graphique, notamment pour Reset le jeu ou pour le relancer. Aussi, le design de l'application pourrait être amélioré via des options supplémentaires.

Enfin, la réalisation de cet exercice nous a permis de développer de nouvelles compétences de gestion de projet et de travail en équipe. Nous avons appris à mieux organiser notre temps, à bien répartir les tâches entre les membres du groupe et à gérer efficacement les contraintes techniques rencontrées.