

# Rapport Projet Interfaces Graphiques et Design Patterns

DAVID MATTHIAS

LE BRIS ILAN

MARCHERON BASTIEN

PARCHEMINER NOLANN



## Table des matières

Table des matières .....	2
Introduction.....	3
Conception .....	3
Graphical User Interface.....	4
Command Line Interface .....	5
Joueurs .....	6
Tests .....	7
Conclusion .....	8

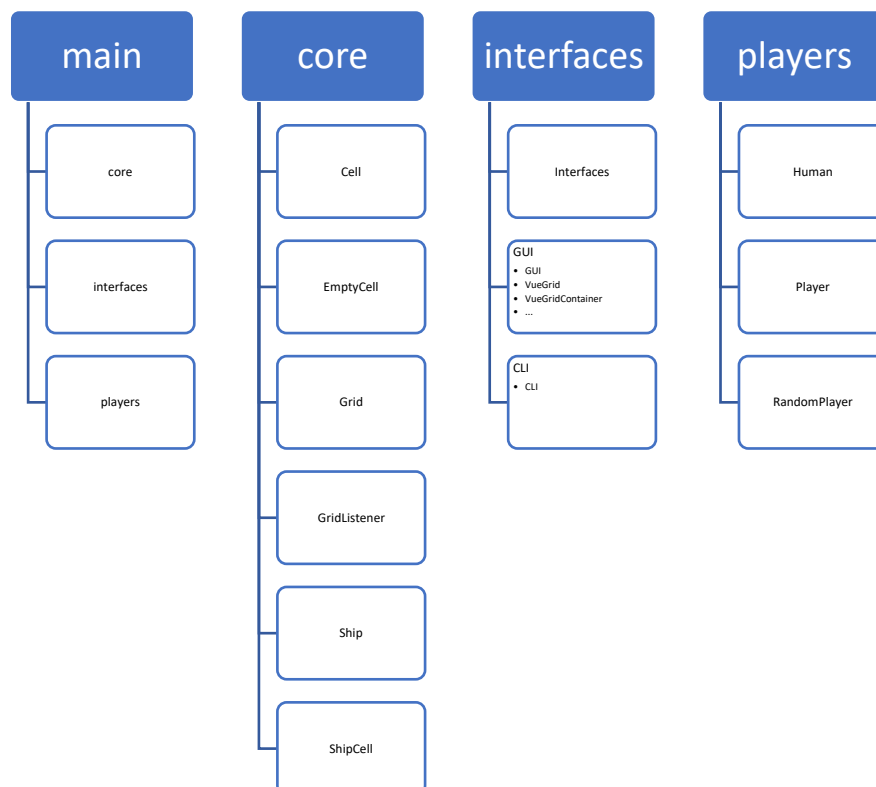
## Introduction

Pour la dernière partie de l'UE Interface Graphiques et Design Patterns, nous avons à réaliser une bataille navale. Le sujet était d'implémenter un jeu de la bataille navale entre un joueur et un algorithme qui tire des coups aléatoires. Nous avons deux implémentations à mettre en place, une version CLI (command line interface) qui s'utilise dans un terminal, et une version GUI (graphical user interface) qui s'utilise comme une application grand public.

Le jeu de la bataille navale est un jeu de type tour par tour dans lequel deux joueurs s'affrontent sur une grille représentant la mer. Chaque joueur possède une flotte de bateaux qu'il place comme il veut sur la grille. Une fois les flottes placées, la partie commence. Chaque joueur va à son tour énoncer une case de la grille, l'adversaire choisira à son tour une case. Si la case choisie est vide, alors rien n'a été touché, si un bateau se trouve sur la case concernée, alors le deuxième joueur doit dire toucher. Si toutes les cases constituant un bateau sont touchées, alors le bateau est coulé. Le premier joueur à couler tous les bateaux de son adversaire remporte la partie.

## Conception

Pour réaliser l'interface graphique, nous devons utiliser `JAVAX.SWING` et `JAVA.AWT`. Nous avons suivi le modèle MVC, en choisissant l'architecture suivante.



Nous avons créé trois packages :

- **CORE**

Ce package contient les classes qui gèrent les modèles. Ce sont les classes de ce package qui forment les briques que nous allons assembler pour créer le jeu.

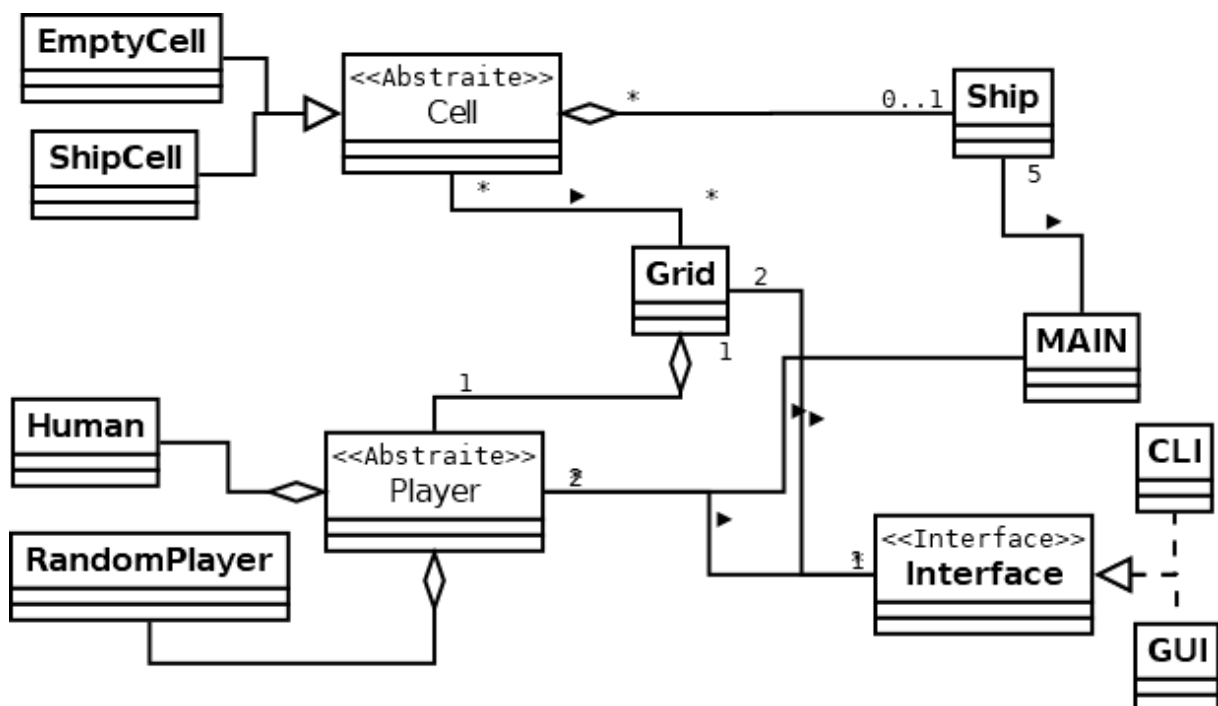
- **PLAYERS**

Ce package implémente le type `PLAYER` et rajoute les joueurs `HUMAN` et `RANDOMPLAYER`. Nous avons décidé de séparer les joueurs du reste pour faire un découpage du code intelligent, avoir une meilleure maintenabilité et faciliter l'implémentation d'un nouveau type de joueur si l'on veut faire évoluer le projet.

- **INTERFACES**

Ce package contient les versions CLI et GUI, c'est dans les packages qu'est faite la gestion des entrées/sorties du jeu, on y trouve donc les vues et les contrôleurs.

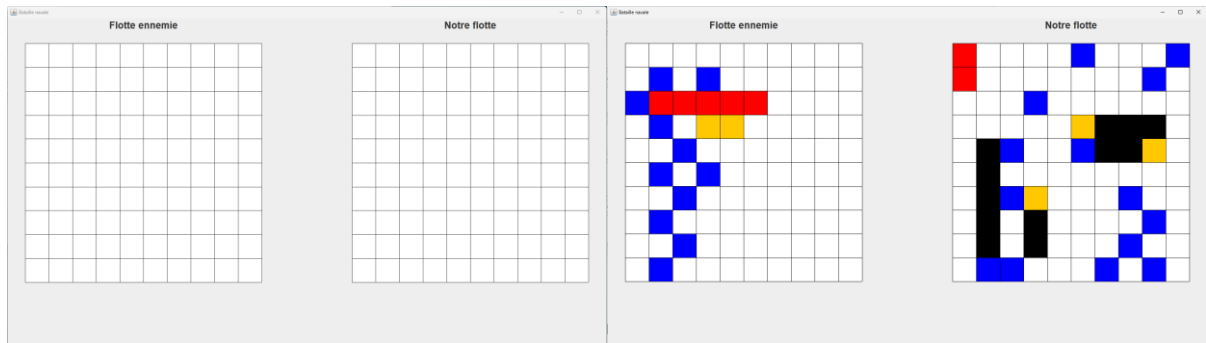
Pour faire de cette application un seul et même exécutable, la boucle d'application a été implémentée dans la classe exécutable `BATTLESHIP`. Nous avons maintenant, lors du lancement la possibilité de choisir si l'on veut exécuter l'application en mode commandes, ou en mode graphique.








## Graphical User Interface

La version GUI est exécutée par défaut, cependant il est possible de la lancer avec les options `-g` ou `-gui` depuis le terminal.

Ci-dessous, l'interface vierge sur laquelle la partie s'affiche, et un exemple d'une partie en cours.



Pour mieux visualiser l'avancée de la partie, nous avons choisi le code couleur suivant :

-  : tir manqué
-  : case vide / case sans information
-  : bateaux
-  : bateaux touchés
-  : bateaux coulés

La conception de l'interface graphique du jeu implique d'utiliser le concept de programmation MVC (Modèle-Vue-Contrôleur). Celui-ci permet de séparer la logique de l'application en trois parties distinctes :

**Le modèle :** Il s'agit de la représentation des données et de leur traitement. Dans le cas de la bataille navale, le modèle correspond aux données du jeu, tel que les différents navires, la grille de jeu etc.

**La vue :** Elle est responsable de l'affichage des données à l'utilisateur. Dans le jeu de la bataille navale, la vue permet d'afficher la grille de jeu, les navires, les coups joués, les résultats, etc.

**Le contrôleur :** Il gère les interactions de l'utilisateur avec l'application et les actions à réaliser en réponse à ces interactions. Dans la bataille navale, le contrôleur gère les actions de l'utilisateur, comme le placement des navires et les tirs, et applique les règles du jeu.

En séparant la logique de l'application en ces trois parties distinctes, le concept de programmation MVC facilite la maintenance et l'évolution de l'application.

## Command Line Interface

Pour la partie CLI, nous avons fonctionné de la même manière que pour la partie graphique.

Pour lancer la version CLI, il faut utiliser l'option -c ou --cli lors du lancement de l'application depuis le terminal.

Ci-dessous, la version CLI qui fonctionne de la même manière que GUI



Toujours dans l'optique de mieux visualiser l'avancement de la partie, nous avons choisi la représentation suivante :

- « . » : tir manqué
- « » : case vide / sans information
- « \$ » : bateaux
- « ! » : bateaux touchés
- « X » : bateaux coulés

## Joueurs

Dans notre application de bataille navale, la gestion des joueurs est séparée du cœur du jeu, de manière à rendre le code plus maintenable. Nous avons implémenté un nouveau type `PLAYER` via une classe abstraite, permettant aux classes `HUMAN` et `RANDOMPLAYER` d'avoir la même base de fonctionnement.

## Player

La classe abstraite `PLAYER`, définit les différentes méthodes de base que peuvent utiliser les classes `HUMAN` et `RANDOMPLAYER`. Telles que les méthodes qui servent à tirer, placer un bateau ou encore savoir si un joueur a perdu.

## Human

La classe `HUMAN` permet de créer un joueur qui pourra interagir avec l'interface donnée en paramètre (GUI ou CLI).

## RandomPlayer

La classe `RANDOMPLAYER` est un joueur qui joue des coups de manière aléatoire. Pour optimiser le choix de ces coups, nous créons une liste qui va enregistrer les coups non joués. Avant chaque coup l'algorithme vérifie si celui-ci est dans la liste, cela lui permet de ne pas choisir deux fois le même coup, et de maximiser ses chances de toucher un navire adverse. En parallèle, cela permet de réduire la quantité de calculs effectués pour choisir un coup. Nous avons la possibilité de faire jouer deux joueurs aléatoires l'un contre l'autre en exécutant l'application avec l'option `-random`.

Pour voir les options disponibles à l'exécution de l'application, il est possible de mettre l'option `-h` ou `-help` afin d'avoir l'aide de démarrage.

## Tests

Nous avons créé des tests afin de vérifier que les fonctions de nos classes soient capables de gérer différentes situations et d'effectuer leurs tâches correctement. Ces tests sont effectués pour identifier d'éventuels bugs ou erreurs dans le code, afin qu'aucun problème ne gêne l'exécution de celui-ci.

Pour tester nos fonctions nous avons créé quatre classes de tests :

- `TESTSHIP` vérifie qu'un bateau est bien orienté et positionné sur la grille. Elle vérifie le statut d'un bateau, quand il est touché, coulé et toujours à flot. Et que l'on puisse aussi obtenir et définir les points de vie d'un bateau par rapport à sa taille.
- `TESTCELL` teste toutes les fonctions de la classe abstraite `CELL` ainsi que celles des classes qui en hérite, tel que `EMPTYCELL` et `SHIPCELL`. `TESTCELL` vérifie qu'une cellule vide et qu'une cellule de bateau peuvent être touchées.
- `TESTGRID` vérifie toutes les coordonnées valides pour placer un bateau quand une grille est vide en vérifiant que toutes ses cellules soient vides. Elle vérifie que l'on puisse bien : placer un bateau, obtenir et définir le statut d'une cellule et obtenir la taille d'une grille. Elle vérifie également les interactions entre la grille et l'utilisateur.
- `TEST` est une classe exécutable dans laquelle sont appelées les fonctions des classes précédemment citées

Les tests servent de sécurité pour certaines fonctions, cependant ils ne garantissent pas que le code fonctionne parfaitement et qu'il ne puisse pas générer d'erreurs dans certains cas.

## Conclusion

Ce projet nous a permis d'élargir nos connaissances en java, et notamment grâce à l'apprentissage du principe du `MODEL VIEW CONTROLLER`. L'utilisation du principe M-V-C pour la réalisation de ce projet nous a permis de mieux comprendre son fonctionnement, ses intérêts ainsi que les avantages qu'elle apporte sur des projets de ce type. De plus, nous avons pu améliorer notre manière de concevoir un plan pour répartir efficacement les différents éléments d'une application.

Ayant l'habitude de travailler ensemble, ce projet nous a une nouvelle fois permis de mettre à profit les aptitudes de chacun pour la réalisation de tâches précises. Nous avons une fois de plus vu à quel point être organiser et se répartir les tâches avant de commencer le projet était un gain de temps pour la suite. Réaliser un cahier des charges et avoir une première réflexion générale quant à la manière de résoudre le problème permet de garder les meilleurs éléments des différentes idées proposées.

- FIN -