



UNIVERSITÉ DE CAEN NORMANDIE

RAPPORT

---

## Jeu de Bataille Navale

---

*Étudiants :*

Celina BEDJOU

Wassim DJEHA

Louheb KACED

Yani BELKACEMI

*Enseignants :*

Yan MATHET

Christophe CHARRIER

*Jury :*

Christophe CHARRIER

14 avril 2023

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Description générale du mini projet . . . . .	2
<b>2</b>	<b>Objectifs du projet</b>	<b>2</b>
2.1	Description des grandes étapes et fonctionnalités à implémenter . . . . .	2
2.2	Organisation et répartition des tâches . . . . .	3
<b>3</b>	<b>Comment lancer le jeu</b>	<b>3</b>
3.1	Création du fichier jar . . . . .	3
3.2	Lancement du jeu . . . . .	3
<b>4</b>	<b>Modèle MVC</b>	<b>4</b>
4.1	Architecture du projet . . . . .	4
4.2	L'Organisation en MVC de notre projet . . . . .	5
<b>5</b>	<b>Interactions avec les différentes vues</b>	<b>6</b>
5.1	Console . . . . .	6
5.2	Interface . . . . .	8
5.3	Améliorations faites . . . . .	8
5.4	Améliorations possibles . . . . .	10
<b>6</b>	<b>Conclusion</b>	<b>10</b>

# 1 Introduction

## 1.1 Description générale du mini projet

Le projet vise à développer le jeu de bataille navale qui est un jeu de société classique dans lequel deux joueurs (joueur vs ordinateur) s'affrontent sur une grille de 10x10 cases. Chaque joueur place ses navires sur la grille de manière stratégique, sans que l'adversaire ne puisse les voir, avant de commencer le jeu.

Le jeu de bataille navale est un jeu amusant et stimulant qui exige des compétences de déduction, de stratégie et de tactique pour gagner. C'est un jeu idéal pour deux joueurs, mais il peut également être adapté pour être joué en solo contre une intelligence artificielle.

# 2 Objectifs du projet

Le projet consiste à développer une version du jeu très connu "Bataille Navale" en utilisant les concepts de Java et le modèle MVC (Modèle-Vue-Contrôleur) pour séparer la logique métier, l'interface utilisateur et la gestion des entrées/sorties de manière organisée et structurée.

## 2.1 Description des grandes étapes et fonctionnalités à implémenter

### 1. CONCEPTION DU MODÈLE :

- Définir les classes pour les bateaux, les cases de jeu, les joueurs, etc.
- Implémenter la logique du jeu, telle que les règles, la gestion des états des bateaux, les validations, etc.
- Gérer les interactions avec le contrôleur pour mettre à jour l'état du jeu en conséquence.

### 2. CONCEPTION DE LA VUE :

- Créer les interfaces utilisateur pour afficher les grilles de jeu, les cases, les navires, les indicateurs d'état, etc.
- Gérer les interactions de l'utilisateur, telles que la sélection d'une case pour tirer un missile, l'affichage des résultats des tirs, etc.
- Rafraîchir la vue en fonction des mises à jour du modèle.

### 3. CONCEPTION DU CONTRÔLEUR :

- Gérer les événements de l'utilisateur, tels que les clics de souris, les touches du clavier, etc.

- Valider les entrées de l'utilisateur et communiquer avec le modèle pour mettre à jour l'état du jeu.
- Rafraîchir la vue pour afficher les changements dans l'état du jeu.

## 2.2 Organisation et répartition des tâches

La répartition des tâches pour le projet de développement du jeu de bataille navale, s'est faite de la manière suivante : deux membres travaillent en binôme sur la partie console du jeu, tandis que les deux autres membres travaillent en binôme sur la partie interface utilisateur du jeu ,sauf

L'intégration de la logique de jeu dans l'interface utilisateur en utilisant le modèle MVC pour assurer une séparation claire des responsabilités et faciliter la maintenance et l'évolution du code qui est faite par les 4 membres.

## 3 Comment lancer le jeu

### 3.1 Création du fichier jar

Tout d'abord on se positionne dans le dossier livraison

1. il faut créer le fichier **MANIFEST.MF** ou on met le fichier JAR.
2. remplacer le contenu du fichier manifest par **Main-class : src/demo/Demo** qui sert à déterminer la class exécutable .
3. Enfin pour créer le fichier JAR on doit taper la commande suivante dans un terminal **jar cvmf ../../dist/MANIFEST.MF ../../dist/bataille.jar src/cotroller/\*.class src/vue/\*.class src/demo/\*.class src/util/\*.class src/model/\*.class module-info.class**.

### 3.2 Lancement du jeu

En ce qui concerne le lancement du jeu , il existe deux méthodes pour les

2 On se positionne dans /livraison :

1. On compile tous les fichiers avec la commande : **javac -d build src/model/\*.java src/cotroller/\*.java src/vue/\*.java src/demo/\*.java src/util/\*.java**
2. Puis on exécute avec la commande **java -cp build src.demo.Demo** ou alors utilisant la 2eme methode :
1. Ou alors en exécutant la commande suivant : **"java -jar dist/bataille.jar"** dans un terminal qu'on ouvrira dans le dossier racine livraison.

## 4 Modèle MVC

### 4.1 Architecture du projet

Dans notre projet, nous avons adopté une structure de données Modèle-Vue-Contrôleur, où le modèle est essentiellement indépendant des autres parties du projet. Lorsqu'une entrée clavier ou souris est détectée, le modèle demande à la vue de solliciter le contrôleur, car la méthode d'accès aux entrées varie en fonction de la vue utilisée

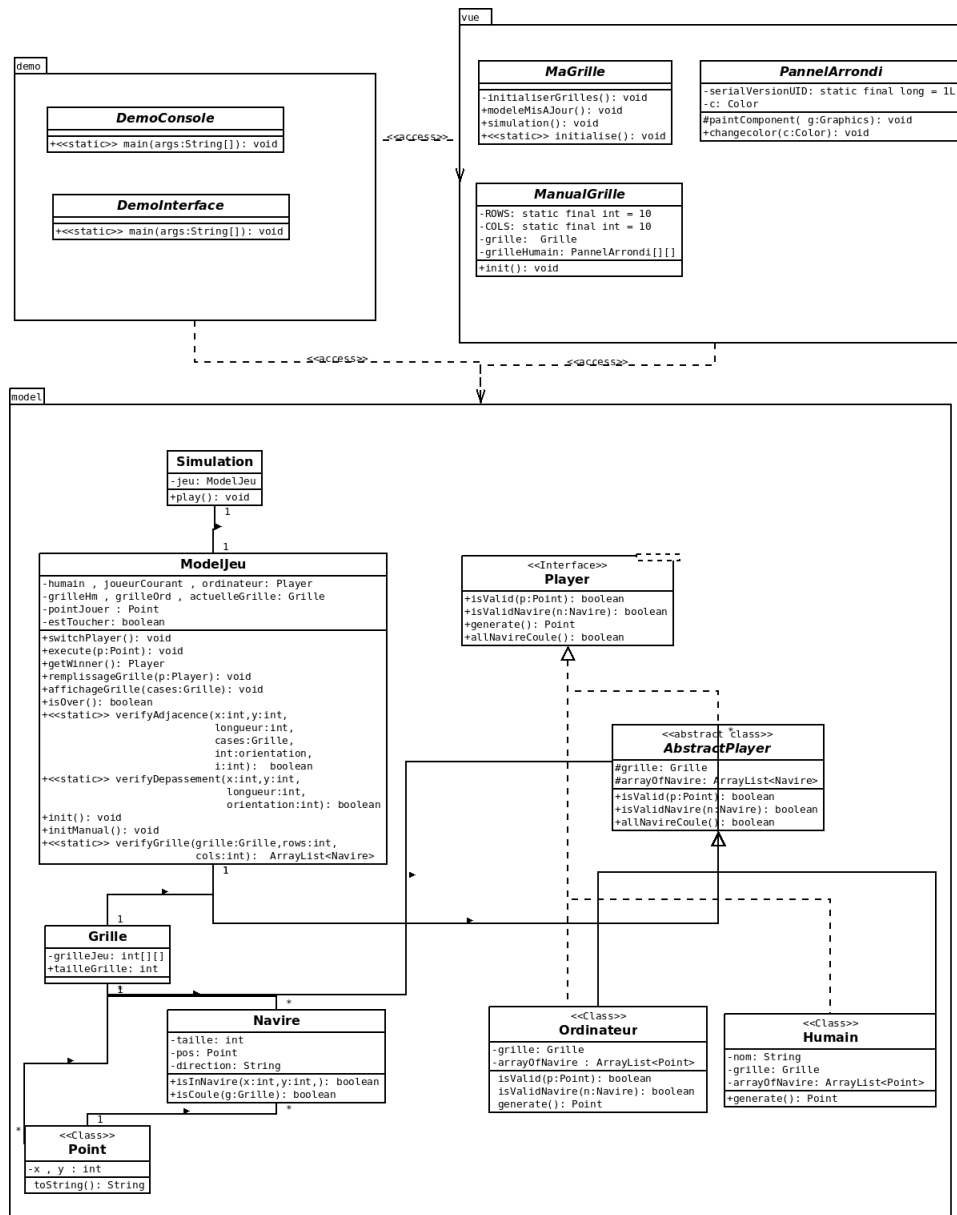
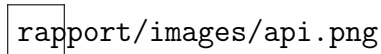


FIGURE 1 – Architecture du projet

Les flèches dans le diagramme sont utilisées pour représenter les différentes relations entre les classes, les interfaces et les autres éléments dans un diagramme de classes :

- \* Héritage : Une flèche solide dirigée de la classe fille (Comme Classe humain) vers la classe (AbstactPlayer).
- \* Interface : Une flèche pointillée dirigée de la classe implémentant l'interface(AbstactPlayer) vers l'interface(Player)



rapport/images/api.png

FIGURE 2 – Exemple d'utilisation Interface/Heritage dans notre projet

## 4.2 L'Organisation en MVC de notre projet

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

Selon l'approche du modèle MVC que nous avons suivie, nous avons souhaité créer un code qui tire parti des fonctionnalités offertes par Java afin d'éviter d'avoir du code "en dur" ou codé en dur. Nous avons cherché à rendre notre code modulable, de manière à permettre l'implémentation d'un algorithme de résolution par une personne externe au projet. Pour ce faire, il lui suffit d'implémenter une nouvelle classe qui héritera de la classe "ModeleJeu" ou bien d'ajouter directement l'algorithme en questions dans la classe "ModeleJeu" et d'y ajouter ses propres algorithmes.

De même, si cette personne souhaite retravailler les vues de l'application, elle peut implémenter une nouvelle classe qui hérite de "EcouteurModele", puis de redéfinir la méthode miseAJour en l'adaptant à la conception du jeu.

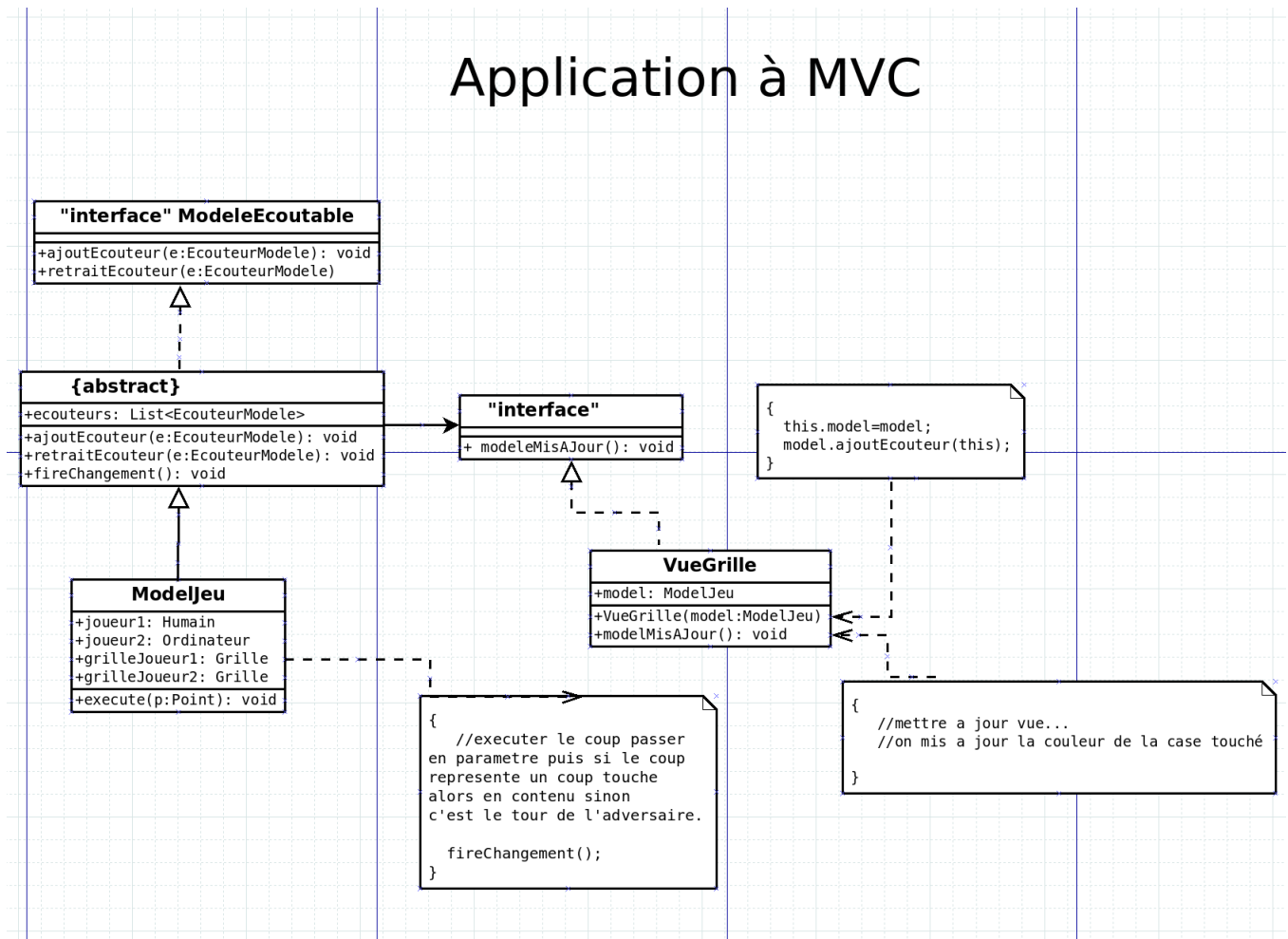


FIGURE 3 – Organisation MVC

## 5 Interactions avec les différentes vues

Dans notre projet, nous avons implémenté deux types d'interactions avec les différentes vues : le Console et l'Interface.

### 5.1 Console

**Console** : l'utilisation du mode "Terminal" permet, d'une part aux utilisateurs (non-IA) de jouer directement via la console, puis le programme demande au joueur de placer ses navires en choisissant les coordonnées (en entrant un chiffre entre 0 et 9 pour chaque axe) et leur direction. Si un joueur tente de placer un navire qui dépasse du plateau de jeu ou qui se superpose avec un autre navire, le programme demande de recommencer le placement, d'autre part il permet à l'ordinateur utilisateur (IA) de remplir ça grille de navires aléatoirement avec la méthode "remplissageGrille()" qui gère les collisions des bateaux.

Ensuite ,une fois que l'ordinateur a fini de placer ses navires, son plateau est masqué . Une fois que tous les navires sont placés, vient le moment de choisir les coordonnées pour les tirs. L'humain entre les coordonnées au clavier de 0 à 9 pour les axes X (abscisses) et Y (ordonnées) et l'ordinateur le fait aléatoirement, .

Le terminal utilise une syntaxe spécifique pour afficher l'état du jeu :

1. "!" pas de navire
2. "1" indique les navires du joueur
3. "\_" indique de l'eau ou un navire masqué

Le programme se termine dès qu'un joueur remporte la partie en affichant le gagnant le nom du joueur.

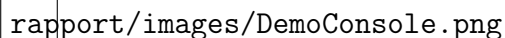
The image is a small rectangular box containing the text `rapport/images/DemoConsole.png`, which serves as a placeholder for a screenshot of the game's console output.

FIGURE 4 – Affichage Console de notre jeu



## 5.2 Interface

**Interface** : le mode "Interface" permet aux utilisateurs (IA et non-IA) de jouer à travers une interface composée de deux grilles, une pour chaque joueur. Le programme demande au joueur Humain de placer ses navires en choisissant les coordonnées en cliquant directement sur une des cases de la grille en suivant les règles du jeu. Si les joueurs tentent de placer un navire qui ne rentre pas entièrement dans sa grille ou qui se superpose avec un autre navire, le programme demandera de recommencer le placement.

Ensuite, une fois que le joueur Humain a fini de placer ses navires, l'ordinateur place ses navires de façon aléatoire en appelons la fonctions "remplissagegrille()". Une fois que tous les navires sont placés, la grille du joueur adverse est masquée et vient le moment de choisir les coups.

Le joueur sélectionne les cases de la grille en cliquant dessus c'est-à-dire en cliquant sur le plateau blue.

Les navires sont représentés par une couleur rouge pour indiquer leur direction :

- Un rond rouge = navire TOUCHE.
- Un rond vert = coup raté (pas de navire)
- le blue indique de l'eau .

Un navire coulé reste affiché à l'écran. Le programme se termine dès qu'un joueur remporte la partie.

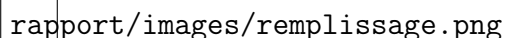
The image shows a screenshot of the game grid, which is a 10x10 grid. The grid is mostly blue, representing water. There are some red and green circles scattered across the grid, representing ships and missed shots respectively. The text 'rapport/images/remplissage.png' is overlaid on the image.

FIGURE 5 – Exemple remplissage de la grille humain avant le debut de la partie

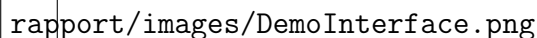
The image shows a screenshot of the game grid, which is a 10x10 grid. The grid is mostly blue, representing water. There are some red and green circles scattered across the grid, representing ships and missed shots respectively. The text 'rapport/images/DemoInterface.png' is overlaid on the image.

FIGURE 6 – Exemple fin d'une partie bataille navale ,gagnant l'humain

## 5.3 Améliorations faites

1. L'implémentation de fonctionnalités avancées dans le jeu de bataille navale peut inclure la possibilité de jouer contre l'ordinateur avec une intelligence artificielle basique. Par exemple, lorsque l'ordinateur choisit un coup aléatoire sur la grille de l'adversaire, il vérifie d'abord si la case choisie a déjà été sélectionnée en consultant la grille pour voir si elle contient un 2. Si c'est le cas, l'ordinateur choisit un autre coup aléatoire.

Si la case contient un 1, alors cela signifie qu'un bateau a été touché, et si la case contient un 0, cela signifie que le coup a été raté. En fonction de ces informations, l'ordinateur ajuste sa stratégie et prend une décision éclairée pour son prochain coup. La mise en œuvre de cette fonctionnalité peut nécessiter la manipulation de données de grille, la génération de nombres aléatoires et la logique de prise de décision pour l'ordinateur afin de simuler une intelligence artificielle basique dans le jeu.

Voici La fonction "execute(Point p)" est une méthode pour gérer la logique de jeu de la bataille navale entre un joueur humain et un joueur ordinateur.

Voici une description générale de ce que cette fonction fait :

- (a) Vérifier si c'est le tour du joueur humain ou de l'ordinateur en comparant le joueur courant avec le joueur humain.
- (b) Si c'est le tour du joueur humain : Vérifier si la case choisie par le joueur humain (représentée par l'objet "Point p") contient un navire de l'ordinateur en vérifiant la valeur de la case dans la grille de l'ordinateur. Si c'est le cas, marquer la case comme touchée (en modifiant la valeur de la case dans la grille de l'ordinateur à 2) et définir la variable "esttoucher" à vrai. Afficher un message indiquant que le coup du joueur humain est un coup "TOUCHE". Sinon, définir la variable "esttoucher" à faux, afficher un message indiquant que le coup du joueur humain est un coup "RATE", et passer au tour du joueur suivant en appelant la fonction "switchPlayer()" pour changer le joueur courant.
- (c) Sinon, si c'est le tour de l'ordinateur : Vérifier si la case choisie par l'ordinateur (représentée par l'objet "Point p") contient un navire du joueur humain en vérifiant la valeur de la case dans la grille du joueur humain. Si c'est le cas, marquer la case comme touchée (en modifiant la valeur de la case dans la grille du joueur humain à 2) et définir la variable "esttoucher" à vrai. Afficher un message indiquant que le coup de l'ordinateur est un coup "TOUCHE". Sinon, si la case choisie par l'ordinateur ne contient pas de navire du joueur humain (représentée par la valeur 1 dans la grille du joueur humain), marquer la case comme jouée (en modifiant la valeur de la case dans la grille du joueur humain à 2) pour éviter que l'ordinateur ne choisisse la même case à nouveau, définir la variable "esttoucher" à faux, afficher un message indiquant que le coup de l'ordinateur est

un coup "RATE", et passer au tour du joueur suivant en appelant la fonction "switchPlayer()" pour changer le joueur courant.

- (d) Mettre à jour la variable "pointjouer" avec la valeur de "Point p". Appeler la fonction "fireChange()" pour notifier les éventuels écouteurs ou observateurs d'un changement dans l'état du jeu.

## 5.4 Améliorations possibles

Les améliorations possibles pour le projet de bataille navale comprennent :

1. Amélioration de l'IA de l'ordinateur pour offrir un défi plus difficile aux joueurs en utilisant des algorithmes de recherche avancés.
2. Ajout de fonctionnalités multijoueurs pour permettre aux joueurs de jouer en mode multijoueur, en local ou en ligne.

## 6 Conclusion

En conclusion, le développement du jeu de bataille navale en utilisant les concepts de Java et le modèle MVC a été un projet passionnant et stimulant. Ce projet a permis aux membres de l'équipe de renforcer leur compréhension des concepts de programmation en Java, ainsi que de se familiariser avec le modèle MVC pour organiser et structurer le code. La collaboration étroite entre les membres de l'équipe a été essentielle pour atteindre les objectifs du projet et garantir une cohérence dans l'expérience utilisateur du jeu.

En conclusion, le jeu de bataille navale développé en utilisant Java et le modèle MVC constitue un exemple réussi d'application pratique des concepts de programmation et de conception de logiciels. Il offre une expérience de jeu intéressante pour les joueurs, tout en démontrant la maîtrise des compétences techniques et la capacité à collaborer en équipe dans un projet de développement logiciel.