RAPPORT
Complément de programmation orienté objet
Projet Bataille Navale

Siepka Aurélien 21906664
Pronost Sacha 21901956
Mathieu Vallée 21910887
Willenbucher Gurvan 21908377

3 mai 2021

UNIVERSITÉ
CAEN
NORMANDIE

Table des matières

1 Introduction

1.1 Description général du projet

Nous allons vous présenter, durant ce rapport, le projet réalisé lors de la matière complément de POO (Programmation Orientée objet). Le but de ce devoir maison était de créer un jeu de Bataille Naval à l'aide du langage de programmation Java. Les règles de ce jeu sont simples, chaque joueur dispose d'un tableau où il va placer des bateaux, la position de ces bateaux étant inconnue pour l'adversaire et inversement. Le but de chaque joueur sera de couler la flotte adverse en tirant sur les bonnes coordonnées. Pour réaliser ce projet nous nous sommes basés sur les CM (Cours Magistraux) et des TP (Travaux Pratiques) de complément POO, mais aussi sur les CM et TP d'introduction à la POO, fait en troisième semestre. Le sujet du projet est commun à tous les groupes, et le but est de coder un jeu de bataille navale tout en respectant le modèle MVC (que nous détaillerons plus tard).

1.2 Présentation du rapport

Durant la création de notre projet, différents éléments techniques de java ont été utilisés, et de nombreuses notions ont dû être implémentées. Nous allons, durant ce rapport, tenter de vous relater au maximum ses différentes informations. Pour cela nous allons suivre un plan précis, dans un premier temps nous parlerons de notre organisation, des différents choix que nous avons dû effectuer ainsi que les objectifs que nous avons dû atteindre. Ensuite nous vous expliquerons les différents éléments techniques de notre codage, que ce soit pour la réalisation de notre jeu basique, mais aussi la création de notre interface graphique. Par la suite nous vous présenterons l'architecture de notre programme à l'aide de différents diagrammes de classes, mais aussi d'une chaîne de traitement expliquant le fonctionnement étape par étape du programme. Finalement nous ferons un bilan de toutes ces étapes. Le but de ce rapport est donc de vous présenter le projet, en vous montrant les différentes étapes nécessaires à la réalisation de ce dernier. Nous essayerons tout du long de cet écrit d'apporter un regard critique sur notre production.

2 Organisation, choix et objectifs

2.1 Description détaillée du projet

Comme expliqué dans la partie précédente, l'objectif du projet était de créer un jeu de bataille naval en Java, et d'implémenter une interface graphique respectant le model MVC. Notre jeu doit donc être jouable sans cette interface (Model indépendant), comme avec. Toujours avec cette idée en tête, il nous a fallu rapidement nous organiser entre nous dans notre codage. C'est pour cela que nous avons créé un fil conducteur que nous avons suivi lors de la création de notre code. Ci-dessous notre fil conducteur :

- La classe Case :
  - ☐ Créer une Case, contenant des variables permettant de vérifier les informations que possède une Case (si elle contient un Bateau, ou si elle a été touché par un Obus).
- La classe Bateau :
  - ☐ Créer, à l'aide d'une taille, un Bateau (concrétiser par une liste de Case), qui peut vérifier si il est coulé ou non.
- La classe Obus :
  - ☐ Permet d'instancier un objet de type Obus. Son but est d'apporter du réalisme au codage, en effet si une case est touchée, elle contient donc un Obus. Cette classe est donc vide.
- La classe ChampBataille :
  - ☐ Créer un plateau de jeu, et permet de placer un bateau sur le plateau ou de viser sur celui-ci (fonction primordiale pour le jeu).
- La classe Joueur :
  - ☐ Créer un joueur possédant un champ de bataille. Ce joueur peut soit placer un bateau sur son plateau, soit viser un autre champ de bataille.

2.2 Organisation et répartition des tâches

Notre ligne conductrice étant créée, il nous suffisait donc plus qu'à nous organiser et à nous répartir les tâches. Notre organisation a été la suivante : Dans un premier temps nous avons créé notre package Model implémentant toutes les classes que nous avons décrites précédemment. Puis nous avons créé l'interface graphique dans deux autres package nommée respectivement Vue et Controller.

La répartition des tâches s'est faite de manière naturelle, en effet Aurélien et Sacha ont été plus à l'aise quand à la création du package model, c'est pour cela que cette dernière a été faite par eux même. La partie interface graphique contenant le package Vue et Controller a été créée par Mathieu et Gurvan. C'est ainsi que nous nous sommes organisés, et que nous avons réussi à terminer notre projet.

3 Element technique du codage

3.1 Model

Différents éléments techniques de Java ont été implémentés dans notre codage du package Model. On peut notamment retrouver les notions de Variables, de liste, de boucles et bien sûr toutes les notions concernant la programmation orientée objet. Chacune de nos classes sont reliées entre elles, en effet certaines font appel à d'autres.

La création d'un plateau de jeu est effectué dans notre classe ChampBataille, pour cela nous faisons une double boucle, et nous ajoutons a une liste a chaque itération de cette boucle une Case avec une position précise. Cette même classe est aussi capable à l'aide d'une méthode de vérifier si un bateau peut être placé sur ce plateau. Pour cela la méthode prend en argument une taille, une position (x et y) et une direction. Elle va tout d'abord regarder si il est possible de placer un bateau à la position demandé, puis si c'est le cas elle renvoie un boolean true. Ci-dessous l'algorithme de cette méthode :

Algorithme 1 : Placement possible d'un bateau

```
Entrées : Un entier taille, deux entiers x et y représentant une position, et un entier direction.
Sortie : Un boolean placementPossible qui est à true si le placement du bateau est possible sur le terrain de jeu,
         ou false sinon.
placementPossible ← True
si direction == 0 alors
    pour yCase ← y à yCase != y + taille faire
        si yCase <= 9 alors
            si getCase(x, yCase).bateauHere() == null alors
                placementPossible ← True
            sinon
                placementPossible ← False
            fin
        sinon
            placementPossible ← False
        fin
    fin
sinon
    pour xCase ← x à xCase != x + taille faire
        si xCase <= 9 alors
            si getCase(yCase, y).bateauHere() == null alors
                placementPossible ← True
            sinon
                placementPossible ← False
            fin
        sinon
            placementPossible ← False
        fin
    fin
fin
retourner placementPossible
```

Il existe aussi deux autres méthodes de la classe champBataille nommée viser et isOver. Viser permet de lancer un Obus a une position précise sur le terrain adverse en appelant la méthode visée de la Case qui va placer sur elle l'obus lancer et vérifié si un bateau a été touché. isOver quand a elle vérifie si le jeu est terminé (si tous les bateaux du champ de bataille sont touchés). C'est à l'aide de ces trois méthodes qu'une partie de jeu peut fonctionner correctement.

3.2 Vue et Controller

Le but des packages vue controller est de créer une interface graphique interactive, appliquant le code du model. Le package controller fait le lien avec la vue et le model. On trouve, dans la classe Controller, différentes variables très importantes pour la compréhension du programme :

- Les variables des joueurs, dont celle du joueur courant permettant de définir le joueur qui doit jouer à ce tour.
- La variable "phase", permettant d'avoir l'état actuel du jeu (phase de placement de bateaux, phase de tir sur le plateau du joueur adverse). Elle peut être égale à trois valeurs : 1 pour la phase de placement, 2 pour la phase de tir et 3 qui est déclenchée si un des deux joueurs remporte la partie.
- La variable "bateauAPlacer", qui est une ArrayList contenant des entiers. Elle permet de définir quels sont les bateaux à placer, en fonction de leur taille, elle sert uniquement pour la phase 1. Par exemple, si elle contient les entiers 2, 3, 3, 4 et 5, le joueur devra placer cinq bateaux de taille respective 2, 3, 3, 4 et 5. L'affichage de ces bateaux à placer, ainsi que l'écouteur, sont ajustés en fonction de cette liste.

Il existe aussi des fonctions importantes :

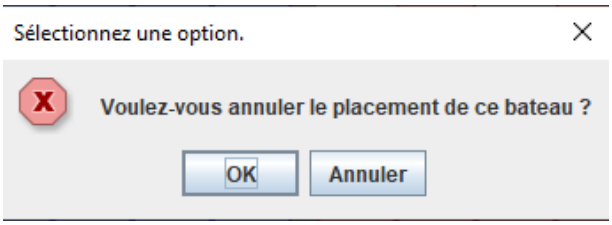
- Les fonctions "placement" et "annulerPlacement", qui permettent respectivement de placer un bateau, ou alors d'annuler son placement en appelant les fonctions du model.
- La fonction "suivant", qui est utile pour tout le déroulement de la partie. Elle change le joueur courant, lorsqu'il exécute une action. Elle va aussi changer la phase, si le placement des bateaux est terminé par exemple.

Les deux joueurs peuvent donc grâce au controller placer des bateaux, jouer au jeu, et ce à l'infini grâce à une fonction qui permet de réinitialiser la partie lorsque celle-ci est terminée.

La classe EcouteurSouris permet d'ajouter un écouteur à la vue pour la souris. Elle hérite de MouseAdapter, une classe de awt permettant d'utiliser la fonction "mousePressed", qui détecte un clique de souris sur la fenêtre du jeu, ce qui nous permet d'obtenir la position de la souris. Ainsi on peut détecter sur quelle case le joueur a cliqué, s'il y a un bateau sur celle-ci.

Le package vue, quant à lui, s'occupe uniquement de l'affichage graphique, il y a une classe par composants placés sur la fenêtre. Les modules swing et awt nous ont permis de créer une interface graphique comme nous l'avions imaginée. La classe la plus importante est la classe Vue héritant de JFrame, elle regroupe tous les autres classes du package, c'est la classe qui va afficher la fenêtre principale sur laquelle tous les éléments seront ajoutés un à un. Celle-ci contient plusieurs fonctions :

- "afficheFenetre" est la fonction qui va servir à actualiser la fenêtre actuelle, en fonction du controller. C'est aussi elle qui place tous les éléments dans la fenêtre, et qui règle quelques options, comme la taille, le titre ainsi que le fait que ce soit possible de fermer la fenêtre si l'on clique sur la croix en haut à droite.
- Il y a aussi des fonctions pour afficher des messages, utilisées pour afficher d'erreur ou alors pour confirmer si l'on veut annuler le placement d'un bateau.



La classe “Images” charge toutes les images du dossier “images” (sous Windows ou Linux) et les met dans une variable. Celles-ci peuvent donc être utilisées dans d’autres classes. Toutes nos images ont une taille de 45x45 pour bien correspondre à une case, et pour pouvoir obtenir un affichage simple et efficace, sans que ce soit trop grand pour les plus petits écrans, ni trop petits.

Nous avons aussi trois classes qui héritent de JPanel, panneaux sur lesquels on peut afficher ce que l’on veut grâce à la fonction “paintComponent” qui est déclenchée lorsque l’on met la fonction setVisible(true), ce sont les classes :

- “VueJoueurCourant” qui affiche un plateau en fonction du champ de bataille du joueur courant. Elle affiche les bateaux qui ont été placés, ainsi que les tirs adverses (rouges pour "toucher" et verts pour “raté”). C’est le plateau à gauche de l’écran.
- “VuePlacer” et “VueTirer” sont des classes qui vont afficher soit les bateaux à placer dans la phase une avec VuePlacer, soit le champ de bataille sur lequel le joueur courant devra tirer sur le joueur adverse dans la phase 2.
- Les deux boutons en bas de l’écran sont des JButton qui servent à activer des fonctions du controller.

Tout ceci donne l’affichage ci-dessous, lors de la phase 1 puis 2 :

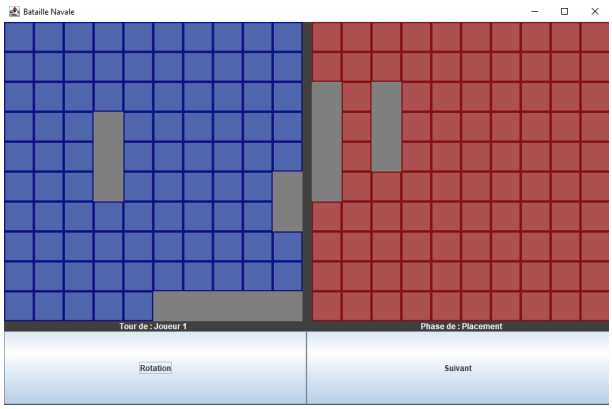


FIGURE 1 – Phase de placement

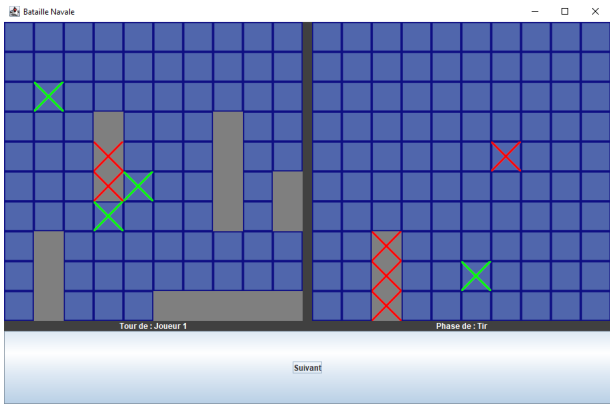


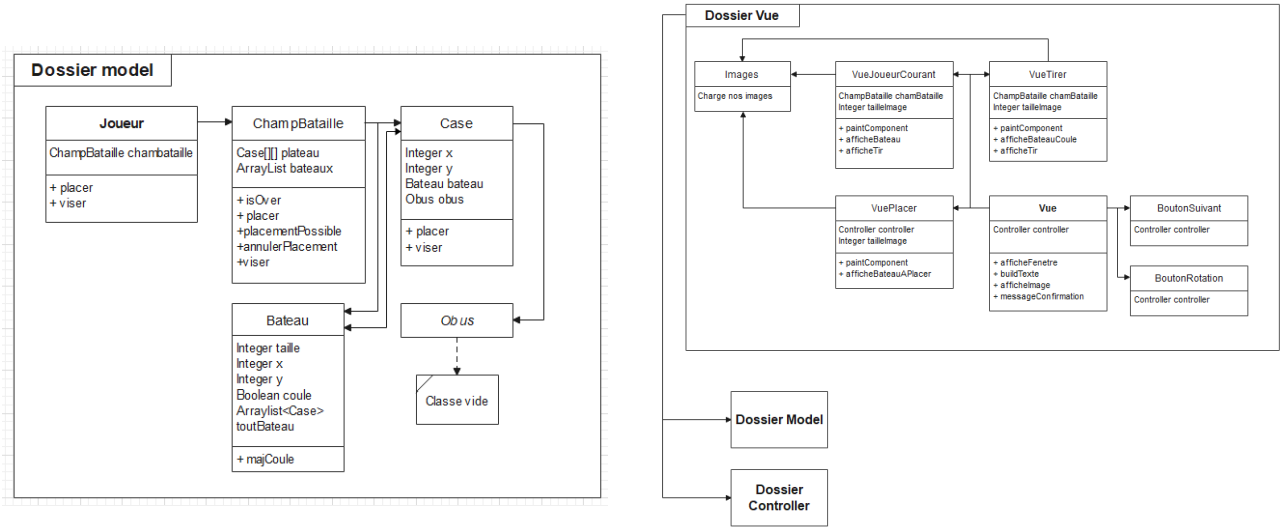
FIGURE 2 – Phase de tir

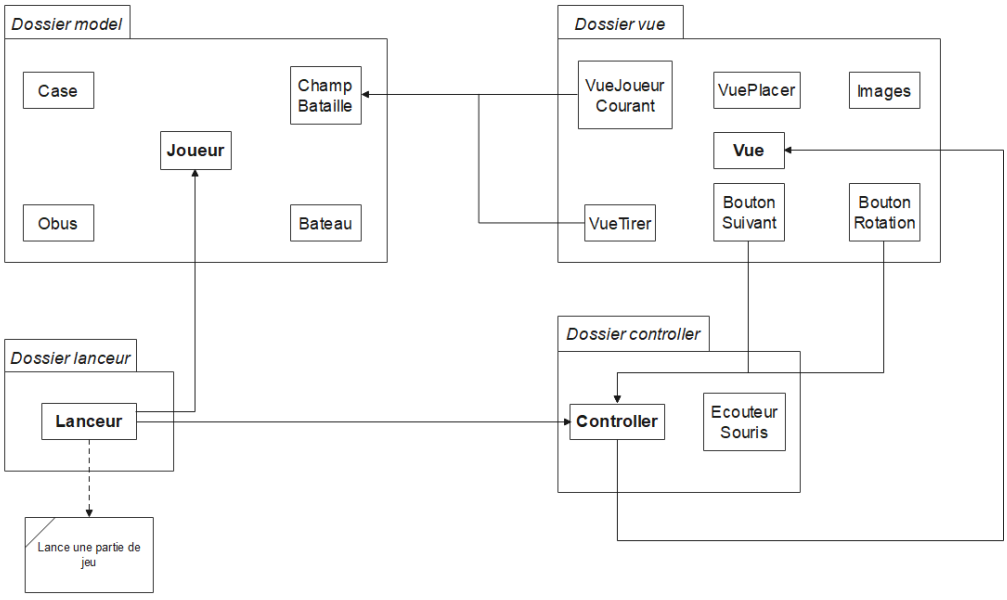
4 Architecture du programme

L'architecture des fichiers de notre jeu a été une partie essentielle de notre projet, en effet il est important que notre programme soit clair, et qu'il soit simple de retrouver une classe en fonction de ce que l'on recherche. Cela permet de gagner du temps non seulement pour l'utilisation du jeu, mais aussi et surtout afin de faciliter le travail et la cohésion entre chacun des membres du groupe, notamment pour apporter des modifications. Comme nous l'avons déjà dit précédemment, le model MVC a été implémenté, c'est donc à partir de cela que nous sommes arrivés à notre architecture finale.

4.1 Diagrammes des classes

Afin de mieux comprendre notre architecture de fichiers, nous allons vous présenter trois diagrammes différents, l'un représentant le contenu précis de notre dossier Model, un représentant le contenu de notre dossier Vue et enfin une représentation de l'ensemble de nos dossiers.





4.2 Lien entre les packages, chaînes de traitement

Nous allons vous expliquer dans cette partie comment notre programme agit étape par étape, en voyant ainsi les liens entre nos différents packages et classes. Notre jeu fonctionne de façon simple et ordonnée, suivant des étapes précises et impliquant chacune de nos classes de manière logique :

- Dans un premier temps la classe Lanceur est appelée, celle-ci va tout d'abord instancier deux joueurs de la classe Joueur en leur ajoutant un nouveau champ de bataille, puis créer un Controller de ces deux joueurs.
- La classe Joueur permet de placer un bateau sur le champ de bataille, ou de viser sur un autre champ de bataille. Le champ de bataille quand a lui créé une liste de Case et une liste de bateau, il peut regarder si la partie est terminée pour lui, placer un bateau sur lui même, ou annuler un placement. Une Case contient une position, un bateau et un obus, elle peut placer un bateau sur elle-même, et placer un obus. Un Bateau contient une position, une taille et une liste de Case (soit toutes les cases qu'il contient). C'est à l'aide de ces différentes classes qu'une partie de jeu peut être effectuée.
- Le Controller précédemment créé va (je me permet de prendre le relais) initialiser quelques variables, comme les bateaux que le joueur aura à placer. Puis le Controller va créer la vue, pour ensuite lui ajouter un EcouteurSouris, classe du même package qui permet d'ajouter un écouteur (listener) à un composant.
- Enfin la vue va permettre, grâce au controller, d'afficher son contenu, grâce à une JFrame principale, ainsi que les JPanel et JButton qui sont contenues dans cette JFrame.

C'est en effectuant successivement ces différentes étapes, que notre jeu fonctionne. Il est ainsi capable d'effectuer toutes les fonctionnalités souhaitées.

5 Conclusion

Finalement, nous somme parvenus à créer un jeu de bataille navale fonctionnel, et une interface graphique possédant diverses fonctionnalité tout en respectant le modèle MVC. Nous avons réussi à adapter fidèlement les règle du jeu de bataille navale, et notre code nous semble clair, et optimisé. Certes certaine chose pourrait être améliorée, mais selon nous, à notre niveau et compte tenu du temps que nous avions à notre disposition, nous somme satisfait du résultat. Pour conclure, si notre projet n'est pas parfait, il nous semble satisfaisant, et remplit les critères les plus importants.

6 Index

6.1 Mode d'emplois

Pour initialiser notre programme (compilé, crée un javadoc, crée un .jar) il vous suffit de placer votre bash dans notre dossier de jeu, puis d'entrer la commande :

```
ant
```

6.1.1 Lancement avec et sans ant

Afin de lancer notre jeu, il vous suffit de lancer le fichier BatailleNavale.jar se trouvant dans le répertoire dist.

De plus, vous pouvez le lancer en utilisant ANT, pour cela placer votre bash dans notre dossier de jeu, puis entrée la commande :

```
ant run
```

Il est aussi possible de lancer le programme en ligne de commande bash. Pour cela placer votre bash dans le dossier principal de notre jeu (ou se situe notre fichier build.xml), ensuite exécuter ces deux lignes de commande :

```
javac -d ./build src/*/*.java

java -cp ./build lanceur.Lanceur
```

6.1.2 Utilisation du jeu

Une fois le jeu lancé, il vous faut tout d'abord placer les bateaux pour les deux joueurs. Pour cela il vous suffit de cliquer sur un des bateaux présent sur le plateau de droite, puis le placer sur votre plateau (à gauche). Vous pouvez cliquer sur le bouton "Rotation" pour changer la direction des bateaux à placer, de plus si vous voulez annuler le placement d'un bateau, il vous suffit de cliquer sur celui-ci. Pour placer les bateaux du joueur suivant, il vous suffit de cliquer sur le bouton "Suivant" (si vous avez placé tous les bateaux du joueur précédent) puis d'effectuer les mêmes manipulations que précédemment. Une fois les bateaux placés, le jeu commence, pour jouer cliquez sur une case du plateau de droite (le plateau de votre adversaire), une croix verte s'affiche sur la case visée si aucun bateau ne se trouve dessus, rouge sinon. Le plateau de gauche représente votre plateau de jeu. Si un bateau est complètement touché, il va s'afficher car il est coulé. Une fois le jeu terminé, vous pouvez cliquer sur le bouton "Suivant" pour rejouer.

6.2 Sources

Model MVC / JavaSwing :

https://www.jmdoudoux.fr/java/dej/chap-awt.htm
https://stackoverflow.com/questions/7764602/how-is-paint-running-without-being-called
Sprites du jeu (objectif)