Université de Caen Normandie
UFR des Sciences
Département Informatique

2ème année de licence d'informatique

Rapport : Complément POO

Hasti Khayami, Thomas Neveu, Valentin Lucas - L2/4B

Avril 2021

Table des matières

0.1 Introduction

Pour les étudiants de Complément POO, il nous a été demandé par groupe de 3-4 de réaliser un devoir. Le but de ce devoir est de réaliser une application de jeu, dotée d'une interface graphique (indépendente de celle-ci), qui consiste à réaliser un jeu de bataille navale.

Les règles sont simples :
≪ *Chacun des deux joueurs dispose d'une flotte (ensemble de navires) positionnée sur une grille (représentant une portion de mer en 2 dimensions). Les joueurs tirent chacun à leur tour sur une position du camp adverse. Si un bateau adverse est impacté, le tir apparaît d'une façon spécifique (rouge dans l'illustration) par rapport à un tir raté (vert dans l'illustration). Le gagnant est le premier joueur parvenant à couler l'ensemble de la flotte adverse.* ≫
Ce devoir doit impérativement respecter les règles d'une réalisation intégralement MVC [1], avec un modèle totalement indépendant de l'interface graphique.

0.2 Architecture du devoir

Concernant l'architecture du devoir, nous avons simplement décidé de le décomposer en deux packages bien distincts au modèle MVC.

games qui comme son nom l'indique, représente le jeu, plus précisément le Modèle.

graphics qui représente l'interface graphique, la partie Vue-Contrôleur venant se greffer au modèle.

0.3 Elements Techniques

0.3.1 Optimisation algorithmique

Au niveau de l'optimisation algorithmique, nous allons vous montrer quelques éléments qui nous paraissent essentiels. Ces éléments sont centrés sur la réalisation d'un tir et d'un tour complet d'un joueur, nous allons tout d'abord présenter comment nous avons structuré notre façon de réaliser un tir et son optimisation de parcours, et ensuite comment nous l'avons implémenté dans la méthode pour jouer.

Dans cette méthode nous n'avons besoin d'ancune boucle, notre interface **Box** nous

1. MVC signifie "Modèle-vue-contrôleur" est un pattern d'organisation de code pour un programme

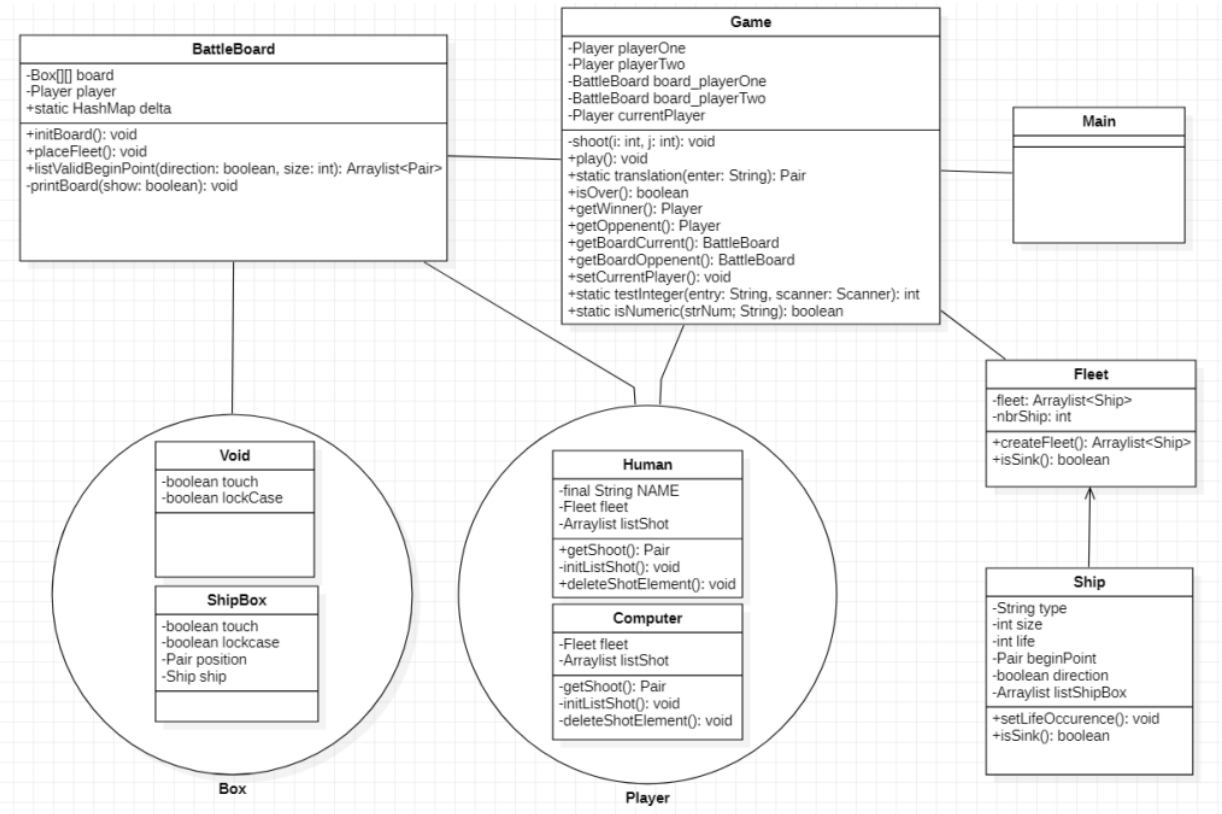


FIGURE 0.1 – Diagramme du package Games

FIGURE 0.2 – Méthode Shoot

```
1     public void shoot (int i,int j) {
2      Box box = getBoardOpponent().getBoard()[i][j];
3          box.setTouch(true);
4          currentPlayer.deleteShotElement(i, j);
5     }
6
```

permet directement d'itérer sur le plateau grâce aux coordonnées données en paramètre. Ensuite Box ayant en booléen **isTouch** nous permet de savoir si la case du plateau a été touché ou non sans besoin de parcours. Et pour finir nous retirons les coordonnées du coup joué dans les coups jouables du joueur (sert principalement pour le joueur ordinateur)

FIGURE 0.3 – Méthode Play

```
1  public void play() {
2    Pair<Integer, Integer> shot = currentPlayer.getShoot();
3    while (getBoardOpponent().getBoard()[shot.getA()][shot.getB()].
       isTouch()){
4      System.out.println("Ce coup a deja ete joue, veuillez en choisir
       un autre :");
5      shot = currentPlayer.getShoot();
6    }
7    shoot(shot.getA(), shot.getB());
8    Object box = getBoardOpponent().getBoard()[shot.getA()][shot.getB()
       ];
9      if (box instanceof ShipBox) {
10       ((ShipBox) box).getShip().setLifeOccurence();
11         if (((ShipBox) box).getShip().isSink()) {
12             getOpponent().getFleet().setNbrShipOccurence();
13         }
14     }
15   }
16
```

Dans cette méthode, grâce à la méthode isTouch() nous pouvons vérifier directement si un coup a déjà été joué, si c'est le cas on redemande un coup le temps que le joueur ne rentrera que des coups non valides. Ensuite on vérifie si la case est une case contenant un bateau afin de lui retirer un point de vie si il a été touché et vérifier sa vie si le bateau est coulé ou non. On retire ensuite 1 au nombre de bateau que possède le joueur adverse. Cette optimisation n'est possible que grâce à la simplification de parcours algorithmiques faites dans la méthode shoot.

0.4 Expérimentation et usages

Nous allons vous montrer ci-dessous les différentes étapes possibles sur notre interface graphique lors du déroulement d'une partie. Avant tout je vais expliquer la légende de

l'application ci-dessous :

- En **Vert** représente les bâteaux non touchés.
- En **Rouge** représente les bâteaux touchés.
- Les **Croix rouges** représentent une case touché sans que ce ne soit un bâteau.
- En **Noir** représente les bâteaux coulés.

FIGURE 0.4 – Début d'une partie

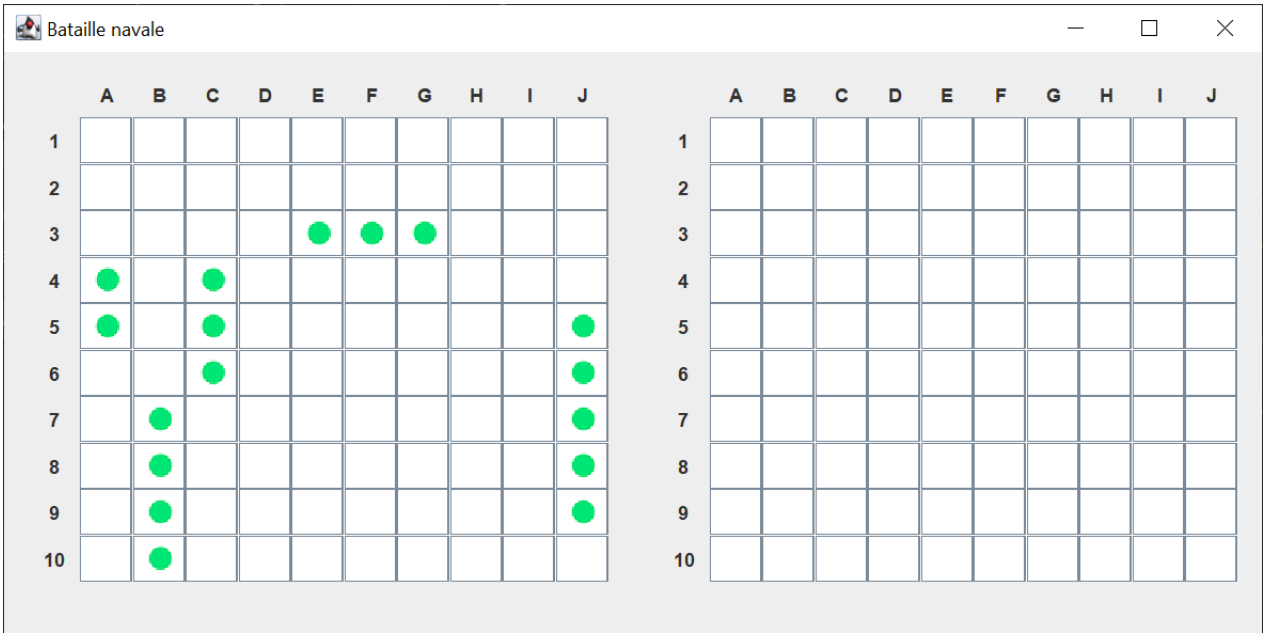


FIGURE 0.5 – Milieu d'une partie

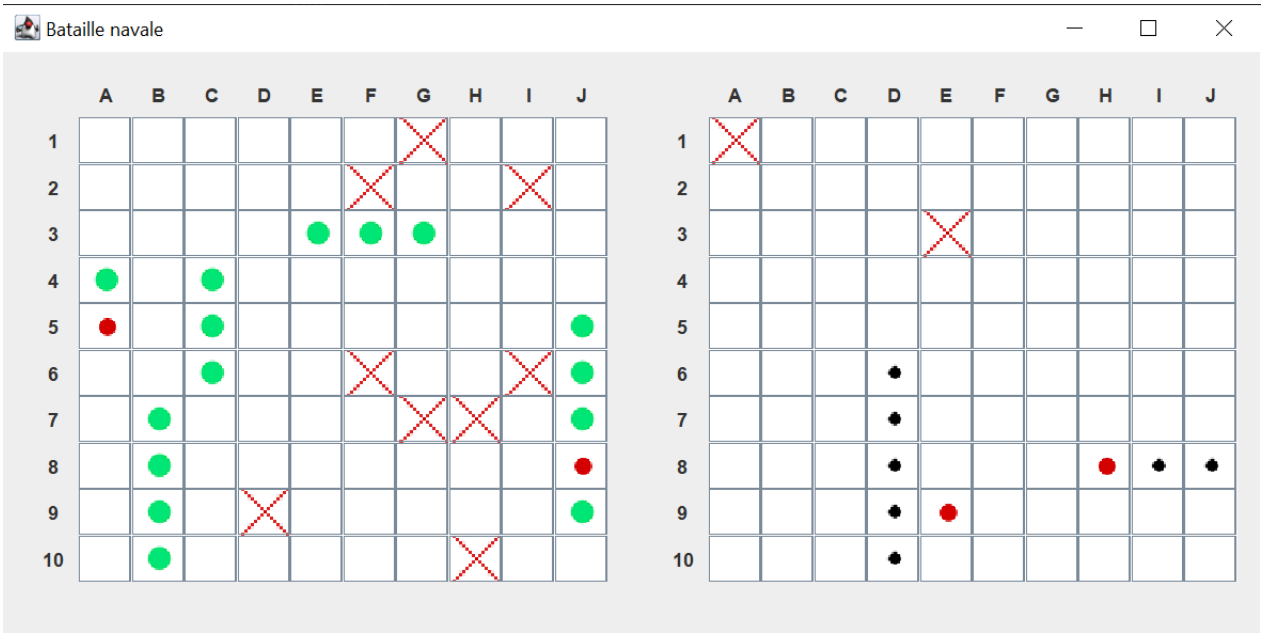
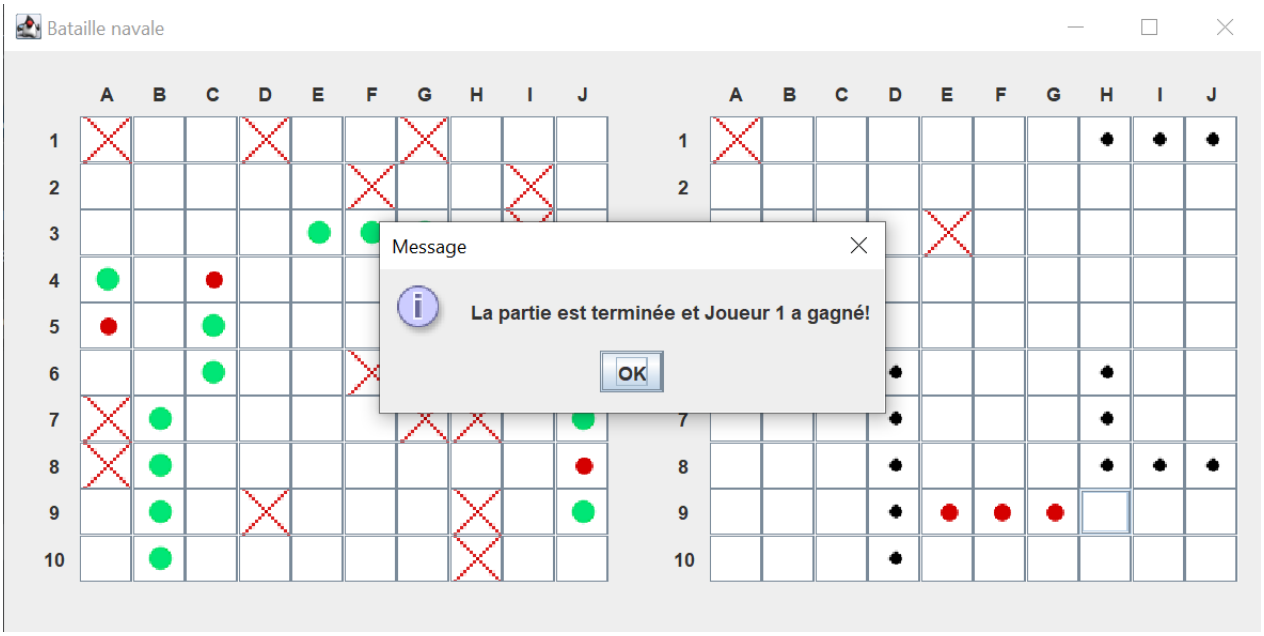


FIGURE 0.6 – Fin d'une partie



0.5 Conclusion

Pour Conclure, ce devoir nous a été bénéfique pour mieux comprendre, pratiquer ce que nous avons vue en cours sur le pattern MVC mais aussi sur la bibliothèque graphique SWING.