UNIVERSITÉ
CAEN
NORMANDIE

Rapport de Projet

Bataille Navale

Ecrit par:

AHMED Youra
DAGNON Jerome
THIAM Souleiman
HABIL Yassmine

Parcours : Licence 2 Informatique

Groupe : 4B

Chargé du TP :

Olivier Ranaivoson

April 14, 2023

Table des matières

Introduction

Le jeu de **bataille navale** est un jeu de stratégie pour deux joueurs qui se joue sur une grille. Le but du jeu est de coulé toutes les navires de l'adversaire avant que les siens ne soient coulé. Chaque joueur dispose d'une grille de jeu sur laquelle sont placées aléatoirement ses navires. Les joueurs alternent ensuite les coups pour essayer de toucher les bateaux de l'adversaire en envoyant des tirs sur sa grille.

Ce jeu est devenu populaire lors de son apparition en 1943 dans les publications américaines de divertissement de la Milton Bradley Company qui l'analysa sous la forme papier jusqu'en 1967, où elle sortit un jeu de plateau, puis en réalisa une version électronique en 1977. Ce jeu est devenu un classique dans les jeux de société et est souvent pratiqué par des joueurs de tous âges. Avec l'avènement de l'informatique, le jeu de bataille navale a été adapté en version numérique et est désormais disponible sur différents supports tels que les ordinateurs, les smartphones et les tablettes.

Dans ce rapport, nous allons présenter une implémentation informatique du jeu de bataille navale en Java. Nous commencerons par expliquer le fonctionnement général du jeu et les règles de base. Ensuite, nous décrirons l'architecture de notre application et les choix de conception que nous avons faits, nous montroront aussi quelques diagrammes de classe afin de mieux visualiser, nous expliqueront aussi notament quelques algorithmes necessaires dans la jeu et Enfin nous discuterons des améliorations possibles et des perspectives d'avenir pour ce projet.

1 fonctionnement general du jeu et regles de base

1.1 Compilation et excecution avec Ant

Dans ce projet, nous avons utilisé le gestionnaire de dépendances et de compilation Ant qui est un outil de construction de logiciel libre écrit en Java.
on distingue plusieurs taches dans le fichier build.xml qui sont :

- **init**: cette tache permet de créer le dossier build et de copier les fichiers sources dans le dossier build.
- **clean**: cette tache permet de supprimer le dossier build.
- **compile**: cette tache permet de compiler les fichiers sources.
- **javadoc**: cette tache permet de générer la documentation du projet.
- **jar**: cette tache permet de créer un fichier jar contenant les fichiers compilés.
- **dist**: cette tache permet de generer la distribution du projet(les fichiers sources, les fichiers compilés et la documentation).
- **run**: cette tache permet d'executer le programme.

- **help**: cette tache permet d'afficher l'aide.
- **install**: cette tache permet d'installer les librairies necessaires pour les tests unitaires.

1.2 Mis en place de la partie

La mis en place de la partie consiste a avoir deux grille pour chacun des joueurs, a positionner aussi les navires, ici on positionne les navires aleatoirement au debut de la partie respectant les contraintes liée a la taille de chaque navire.

1.3 Deroulement du jeu

Maintenant que tous les bateaux sont placés, la partie peut commencé.

Les joueurs vont tiré chacun leur tour sur une position de la grille adversaire on note qu'un joueur peut etre un humain, un Joueur aleatoire ou une Intelligence artificielle.

En fonction du choix du joueur, le tir sera effectué de manière différente. Si le joueur est un humain, il pourra sélectionner une case de la grille adverse en cliquant dessus dans l'interface ou alors entrer les coordonnées de la case qu'il souhaite viser.

Si c'est une Intelligence Artificielle, celle-ci devra déterminer la case à viser en fonction de sa stratégie de jeu en interface comme en console.

Si c'est un joueur aléatoire, le choix de la case sera effectué de manière totalement aléatoire aussi en interface comme en console.

1.4 Gestion des navires et des tirs

Chaque joueur dispose d'une grille de jeu sur laquelle sont disposés ses navires, tandis que l'adversaire doit essayer de les localiser en envoyant des tirs sur différentes cases de la grille.

La gestion des navires est assurée par la classe Navire, qui permet de définir la position et la direction de chaque navire, ainsi que sa taille. Lorsque les navires sont placés sur la grille, ils sont représentés par une série de cases adjacentes. La classe Grille est utilisée pour représenter la grille de jeu et permet de déterminer si une case est occupée par un navire ou non.

Le joueur peut alors envoyer des tirs sur différentes cases de la grille, en utilisant la classe Position pour définir les coordonnées de chaque tir. La classe Grille est également utilisée pour gérer les tirs, en déterminant si un tir a touché ou non un navire. Si un tir touche un navire, la classe Navire est utilisée pour déterminer quelle partie du navire a été touchée, et si le navire a été coulé ou non.

1.5 Victoire et defaite

Un joueur gagne la partie si le joueur adverse n'a plus de navires non coulés, c'est-à-dire que toutes les positions de tous les navires de l'adversaire ont été touchées. Ainsi, la méthode `isOver()` renvoie `true` si le joueur courant (celui qui vient de jouer) n'a plus de navires non coulés, c'est-à-dire si `this.joueurCourant.isDead()` renvoie `true`.

Un joueur perd la partie s'il n'a plus de navires non coulés, c'est-à-dire que toutes les positions de tous ses navires ont été touchées et que le joueur adverse possede toujours des navire non coulés. Ainsi, la méthode `getJoueurGagnant()` renvoie le joueur adverse (celui qui n'a pas été couronné) si le joueur courant (celui qui vient de jouer) n'a plus de navires non coulés, c'est-à-dire si `this.joueur1.isDead()` renvoie `true`, sinon elle renvoie le joueur courant.

2 Architecture du projet

Dans ce projet, nous avons utilisé le modèle MVC (Modèle-Vue-Contrôleur) pour la conception de notre application.
Dont le but est de séparer les différentes parties de l'application en trois parties distinctes : le modèle, la vue et le contrôleur.
Ce pendant nous avons dans notre architecture 5 principaux Modules :

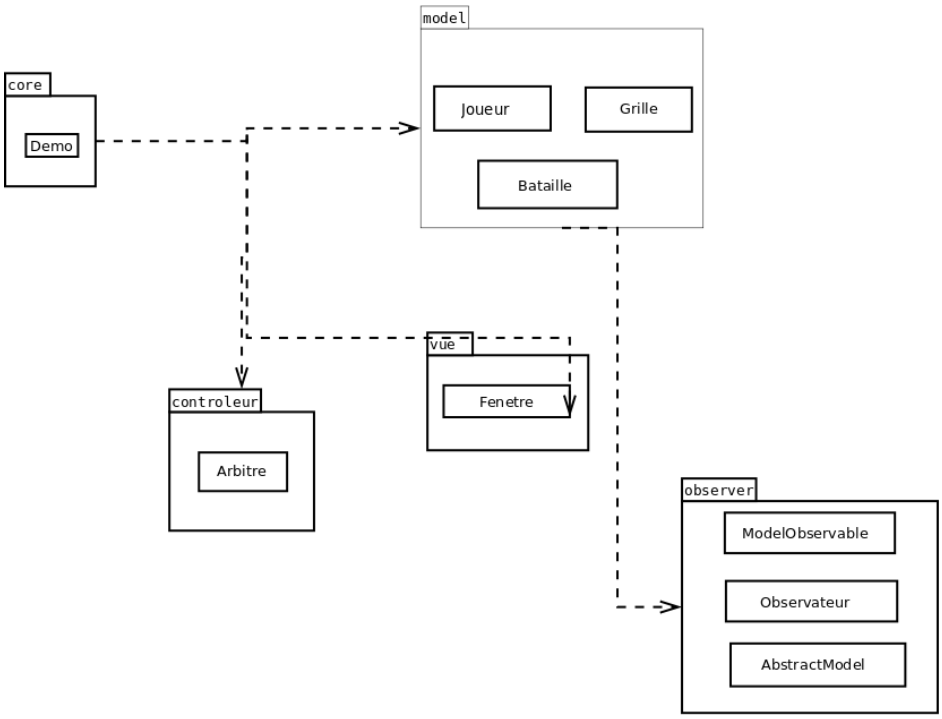

Figure 1: Diagramme de paquetage

- **Model** : Il contient les classes qui permettent de jouer au jeu de bataille navale. Il contient les classes qui permettent de gérer les navires, les tirs, les grilles, les joueurs, les positions, les parties,etc.

- **Vue** : Il contient les classes qui permettent de représenter graphiquement le jeu de bataille navale. il permet de représenter graphiquement avec les differents composants graphiques de swing les grilles, les navires, les tirs, etc.

- **Controler** : Ceci est la partie qui permet de gérer les interactions entre le modèle et la vue. Il permet de gérer les événements de l'interface graphique, de gérer les actions des joueurs, etc.

- **core** : Il contient les classes qui permettent de jouer au jeu de bataille navale en console.

- **observer** : Il contient les classes qui permettent de gérer les observateurs et les observables.

2.1 Module et classe

2.1.1 Model

le model de notre projet consiste à gerer les differents elements du jeu de bataille navale, il contient les classes qui permettent de gérer les navires, les tirs, les grilles, les joueurs, les positions, les parties, etc.

- **Navire**: Elle contient des informations telles que la position de départ du navire, sa direction, sa taille et le nombre de cases touchées. Elle dispose également de méthodes pour accéder à ces informations et pour déterminer si le navire est coulé ou non.

- **Grille**: Cette classe permet de représenter la grille de jeu et permet de déterminer si une case est occupée par un navire ou non grâce à un matrice de boolean **emplacementsOccupes**.

- **Position**: Cette classe permet de définir les coordonnées de chaque case de la grille.

- **Joueur**: Cette classe permet de définir les informations d'un joueur.

- **Bataille**: Cette classe permet de définir les informations d'une partie c'est-à-dire a determiner le joueur courant, le joueur gagnant, si le jeu est fini ou non, a executer un tir, ect.

2.1.2 Vue-controleur

La vue de notre projet consiste à gerer les differents elements graphiques du jeu de bataille navale, il contient les classes qui permettent de représenter graphiquement le jeu de bataille navale. il permet de représenter graphiquement avec les differents composants graphiques de swing les grilles, les navires, les tirs, etc.

2.2 Diagrammes de classe

3 Quelques algorithmes utilisées

On a utilisé les algorithmes suivants dans notre projet:

3.1 Execution d'un coup

On a une methode `executerCoup()` qui permet d'executer un coup sur la grille de l'adversaire.

Algorithme 1 : Exécution d'un coup

Entrées : x et y les coordonnées du coup à jouer
Output : le coup est joué sur la grille de l'adversaire

```
1  grilleAdverse ← joueurAdverse.getGrille();
2  pos ← grilleAdverse.getPosition(x, y);
3  si pos n'est pas touché alors
4  |  changer l'état de pos en touché;
5  |  si pos a un navire alors
6  |  |  si le navire est coulé alors
7  |  |  |  incrementer le nombre de navires coulés du joueur adverse;
8  |  |  fin
9  |  fin
10 |  prevenir les observateurs;
11 |  ajouter le coup à la liste des coups joués;
12 |  on change de joueur;
13 fin
```

3.2 Choisir un coup pour le joueur Machine

Algorithme 2 : choisirCoup(Bataille bataille)

Entrée : une instance de la classe Bataille
Sortie : la position choisie pour tirer

```
1  tant que la position choisie est déjà touchée faire
2  |  si un navire a déjà été touché mais pas coulé alors
3  |  |  choisir une position adjacente à un navire touché;
4  |  sinon
5  |  |  choisir une position aléatoire;
6  |  fin
7  fin
8  mettre à jour la liste des positions des navires touchés; retirer la position
   choisie de la liste des coups valides; retourner la position choisie;
```

3.3 Placement aleatoire des navires

Cette méthode permet de placer les navires aléatoirement sur la grille.

Algorithme 4 : Placement aléatoire des navires

Entrée : aucune
Sortie : les navires sont placés aléatoirement sur la grille

```
1  tableauTailles ← [5, 4, 3, 3, 2];
2  pour i allant de 0 à 4 faire
3      estPlace ← faux;
4      tant que estPlace est faux faire
5          generer aleatoirement une position;
6          generer aleatoirement une direction;
7          navire ← nouveau Navire(position, direction, tableauTailles[i]);
8          si la grille peut placer le navire à cette position avec cette
             direction alors
9              placer le navire sur la grille;
10             ajouter le navire à la liste des navires;
11             estPlace ← vrai;
12         fin
13     fin
14 fin
```

4 Amelioration possible