Compte rendu
Bataille Navale

Louis Ludovic Herzog

18 Novembre 2025

Sommaire

1 Analyse du problème

Le but de ce TP était de créer un jeu de bataille navale simplifiée en Java. Le jeu est un PvC (Player vs. Computer). Comparés au vrai jeu, tous les bateaux n'occupent qu'une seule case.
L'enjeu principal était de gérer l'état du jeu pour le joueur et l'ordinateur, en utilisant des tableaux à deux dimensions, et de gérer les tours de jeu jusqu'à ce qu'un vainqueur soit déterminé.

2 Conception de la solution

J'ai conçu ma solution en m'appuyant sur des tableaux à deux dimensions (int[][]) pour gérer les grilles de jeu, en réponse au cahier des charges.

Structures de données

J'ai utilisé quatre tableaux 5x5 distincts :

- **tabJoueur :** Pour stocker les positions des navires du joueur ("1" ou "0" suivant la présence ou non des navires) ;
- **tabOrdi :** Pour stocker les positions des navires de l'IA ;
- **tabBataille :** Pour mémoriser les tirs du joueur ("2" pour "touché", "3" pour "raté") ;
- **tabBatailleOrdi :** Pour mémoriser les tirs de l'IA sur la grille du joueur ;

Algorithme général

1. Une boucle principale do-while gère la possibilité de rejouer la partie. ;
2. Initialisation des quatre tableaux à 0 ;
3. Phase de placement du joueur : Une boucle do-while demande 5 fois les coordonnées, en vérifiant si la case est valide (dans la grille 1-5) et non déjà prise ;
4. Phase de placement de l'IA : Une boucle do-while génère 5 positions aléatoires (Math.random()), en vérifiant également si la case est déjà prise ;
5. Phase de combat : Une boucle while s'exécute tant que ni le joueur ni l'IA n'ont atteint 5 points (nbPionsTouchesJoueurs < 5 et nbPionsTouchesOrdi < 5) ;
6. Dans cette boucle, le joueur joue en premier (saisie et vérification des tirs), puis l'IA joue (tirs aléatoires) ;
7. Après chaque tour, les grilles "cachées" sont affichées et le score est vérifié pour une condition de victoire.

3 Réalisation

- **Saisie et Aléatoire :** J'ai utilisé java.util.Scanner pour toutes les entrées du joueur. Pour l'IA (int)(Math.random() * 5) + 1 a été utilisé pour générer les coordonnées (lignes/colonnes) de placement et de tir ;
- **Fonctions d'affichage :** Conformément aux étapes 4 et 5 du TP, deux méthodes statiques ont été créées :
  - **AffichageJoueur :** Affiche la grille du joueur avec ses pions visibles ;
  - **AffichageTabCache :** Affiche une grille de combat. Elle masque les cases non découvertes (0 ou 1) par un ' ?', et affiche 'o' (touché, état 2) ou 'x'(raté, état 3). J'ai utilisé les codes couleur ANSI (S_RED, S_GREEN) fournis pour améliorer la lisibilité. Pour comprendre ce que c'était et comment les utiliser, j'ai utilisé de l'IA ;

```
public static final String S_RESET = "\u001B[0m";
public static final String S_RED = "\u001B[31m";
public static final String S_GREEN = "\u001B[32m";
```

- **Logique de jeu :** Les tours s'enchainent dans la boucle while. Les variables nbPionsTouchesJoueurs et nbPionsTouchesOrdi sont incrémentées à chaque tir réussi ;

```
static void AffichageTabCache(int tab[][], int nbcase){
        System.out.println("    " + "1  " + "2  " + "3  " + "4  " + "5  ");
        System.out.println("    " + "|  " + "|  " + "|  " + "|  " + "|  ");
        for (int i = 1; i < nbcase+1; i++) {
            System.out.print(i + " - ");
            for (int j = 0; j < nbcase; j++) {
                if (tab[i-1][j] == 0){
                    System.out.print("?  "); }
                if (tab[i-1][j] == 1){
                    System.out.print("?  "); }
                if (tab[i-1][j] == 2){
                    System.out.print(S_GREEN + "o  " + S_RESET); }
                if (tab[i-1][j] == 3){
                    System.out.print(S_RED+ "x  " + S_RESET); }
            }
            System.out.print("\n");
        }
```

4 Test

J'ai réalisé une série de tests pour vérifier le bon fonctionnement :
Test de placement : J'ai essayé de placer un pion en dehors de la grille (ex : ligne 6) et sur une case déjà occupée. Dans les deux cas, le programme refuse et demande de nouvelles coordonnées.

Test de tir : Les tirs en dehors de la grille sont également bloqués.
Test de "tir à blanc" : Si le joueur ou l'IA tire sur une case déjà découverte (état 2 ou 3), le message "Tir à blanc" s'affiche et le tour passe, sans incrémenter le score.
Test de scénario complet : J'ai fait plusieurs parties et les conditions de victoire fonctionnent (le premier à 5) et la boucle "Voulez-vous rejouer ?" est opérationnelle.

```
Entrez la coordonnée "ligne" de votre tir (De 1 à 5) :
1
Entrez la coordonnée "Colonne" de votre tir (De 1 à 5) :
1
Tir en cours ...
Touché
    1  2  3  4  5
    |  |  |  |  |
1 - o  ?  ?  ?  ?
2 - o  ?  ?  x  x
3 - x  ?  x  o  ?
4 - ?  x  ?  ?  ?
5 - ?  ?  ?  ?  ?
Tir de l'ennemi en cours ...
L'ennemi vous raté !
    1  2  3  4  5
    |  |  |  |  |
1 - ?  ?  ?  x  x
2 - o  ?  ?  ?  ?
3 - ?  ?  ?  ?  ?
4 - ?  x  ?  x  o
5 - ?  ?  x  ?  x
```

5 Retour d'expérience

Ce TP a été bénéfique pour apprendre à manipuleripuler des tableaux de dimensions 2 et l'utilisation des boucles (while, do-while).

- **Difficultés rencontrées :** Lors de la rédaction du code pour le tour de l'IA (étape 7), j'ai remarqué que l'IA vérifiait son propre tableau (tabOrdi) pour savoir si elle avait touché un pion, au lieu de vérifier le tableau du joueur (tabJoueur). C'était une erreur de logique que j'ai dû corriger (remplacer tabOrdi[...] par tabJoueur[...] dans les conditions de tir de l'IA) pour que le jeu soit fonctionnel ;onnel ;
- **Amélioration possible :** L'IA est très basique (elle tire au hasard). Elle pourrait être améliorée pour ne pas tirer deux fois au même endroit (en vérifiant tabBatailleOrdi avant de tirer) ou pour chercher autour d'un tir réussi.