TP6

04/11/2025

Bataille navale (java)

Sommaire

Introduction

L'objectif du TP consiste à faire un programmer simplifiée du jeu Bataille Navale en code Java. Le jeu se déroule sur une grille 5×5 dans laquelle le joueur et l'ordinateur placent chacun 5 pions. Chaque case du tableau contient une valeur indiquant son état (vide, pion non découvert, pion découvert ou case vide déjà tirée). L'objectif est de permettre au joueur d'affronter l'ordinateur en alternant les tirs jusqu'à ce que l'un des deux découvre toute la flotte de pions adverses.

I. Conception

La solution retenue repose sur deux tableaux d'entiers représentant les plateaux du joueur et de l'ordinateur. Le problème a été découpé en sous-problèmes : initialisation des grilles, placement des pions, gestion de l'affichage, traitement d'un tir, alternance des tours et détection de victoire. Chaque étape correspond à une méthode dédiée, ce qui facilite la compréhension et la réutilisation du programme. Les structures de données utilisées restent simples (tableaux 2D)

II. Réalisation

Etape 1 : initialisation des tableaux tabjouer & tabordi

Pour commencer, on va déclarer une table de 5 lignes pat 5 colonnes, et attribuer à chaque ligne et chaque colonne la valeur 0.

```
// Étape 1 : initialisation des tableaux
private void initialiserTableaux() {
    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 5; j++) {
            tabJoueur[i][j] = vide;
            tabOrdi[i][j] = vide;
        }
    }
}
```

```
private final int[][] tabJoueur = new int[5][5];
private final int[][] tabOrdi = new int[5][5];
```

Pour éviter d'avoir un code trop chargé, certaines parties ont été séparées dans des méthodes privées.

```
// Lancement d'une partie complète
private void lancerPartie() {
    // Étape 1 : initialisation des tableaux
    initialiserTableaux();

    // Étape 2 : placement des 5 pions du joueur
    placerPionsJoueur();

    // Étape 3 : placement des 5 pions de l'ordinateur
    placerPionsOrdi();

    nbPionTrouveJoueur = 0;
    nbPionTrouveOrdi = 0;
```

Cela permet au programme de suivre clairement les étapes du TP, une par une.

Etape 2 : On place nos 5 pions : tabJoueur

2.1 et 2.2:

```
//2.1 et 2.2
while (places < nb_pion) {
    System.out.print("Pion " + (places + 1) + " - ligne (1 à " + nb_case + ") : ");
    int ligne = sc.nextInt();
    System.out.print("Pion " + (places + 1) + " - colonne (1 à " + nb_case + ") : ");
    int colonne = sc.nextInt();
```

Ce programme va demander à quel ligne le pion doit être placer entre 1 et 5. Pareil pour la colonne.

2.3 : afficher les deux tableaux pour vérifier les placements :

```
// Attention : l'utilisateur ne doit pas placer un pion hors du périmètre
if (ligne < 1 || ligne > nb_case || colonne < 1 || colonne > nb_case) {
    System.out.println("Position du pion hors plateau !");
    continue;
}

// bloquage si cette case est déjà pris par un pion
if (tabJoueur[ligne - 1][colonne - 1] == pion_non_vu) {
    System.out.println("Il y a déjà un pion à cet endroit !");
    continue;
}

// Choix valable : on place le pion
tabJoueur[ligne - 1][colonne - 1] = pion_non_vu;
places++;
```

Ceci va déterminer si le choix d'emplacement de l'utilisateur ne va pas placer le pion hors de la grille ou placer dans une case qui contient déjà un pion.

2.4 : on répète étape 2.1 à 2.3 tant que le joueur n'a pas placé tous les pions

```
while (places < nb_pion)
```

```
private static final int nb_pion = 5;
```

```
 System.out.println("Il y a déjà un pion à cet endroit !");
 continue;
```

Pour le répété, while est la base pour en faire une boucle et avec la variable que le nombre final de nb_pion doit être jusqu'à 5. « continue » est une condition dans le cas ou le pion est déjà placer ou hors de la grille qui va recommencer de au départ le choix de la ligne et colonne.

Etape 3 : on place les 5 pions de l'ordinateurs : tabOrdi

3.1 et 3.2 placement aléatoire ligne/colonne

```
// 3.1 et 3.2 : ligne et colonne aléatoire entre 1 et 5
int ligne = (int) (Math.random() * nb_case) + 1;
int colonne = (int) (Math.random() * nb_case) + 1;
```

L'étape 3 est que l'ordi place ses pion aléatoirement ses pions entre 1 et 5 sur ligne et colonne 5 fois.

3.3 Vérification de l'emplacement

```
// 3.3 : pas d'endroit déjà pris
if (tabOrdi[ligne - 1][colonne - 1] != pion_non_vu) {
    tabOrdi[ligne - 1][colonne - 1] = pion_non_vu;
    places++;
}
```

3.4 vérifie jusqu'à combien doit être placer

```
while (places < nb_pion)
```

Etape 4: Créer la procédure affichagetab

4.1 - 4.2 création de la partie privé et création de la grille d'affichage

```
private static void affichagetab(int[][] tab, int nbcase) {
    // 4.2 : afficher les numéros de colonnes
    System.out.print("   ");
    for (int col = 1; col <= nbcase; col++) {
        System.out.print(col + " ");
    }
    System.out.println();
```

4.3 – 4.4 :

Il montre les numéros des lignes

```
// 4.3 et 4.4 : chaque ligne
for (int ligne = 1; ligne <= nbcase; ligne++) {
    System.out.print(ligne + "  ");

    //chaque colonne
    for (int col = 1; col <= nbcase; col++) {
        int valeur = tab[ligne - 1][col - 1];

        //si valeur 1 → un pion "o"
        //si valeur 0 → pas de pion "~"
        if (valeur == pion_non_vu || valeur == pion_vu) {
            System.out.print("o ");
        } else {
            System.out.print("~ ");
        }
    }
    System.out.println();
```

Il montre chaque colonne de la grille

```
//chaque colonne
for (int col = 1; col <= nbcase; col++) {
    int valeur = tab[ligne - 1][col - 1];
```

4.5 : affichage des pions visible dans la grille du joueur

```
//Étape 4.5 : affiche le tableau du joueur après le placement des pions
System.out.println("Votre champ de bataille (pions visibles) :");
affichagetab(tabJoueur, nb_case);
```

Ceci va montrer la grille du joueur et aussi l'emplacement de ses pions (un petit bonus)

Etape 5: Création de la procedure affichagetabCache

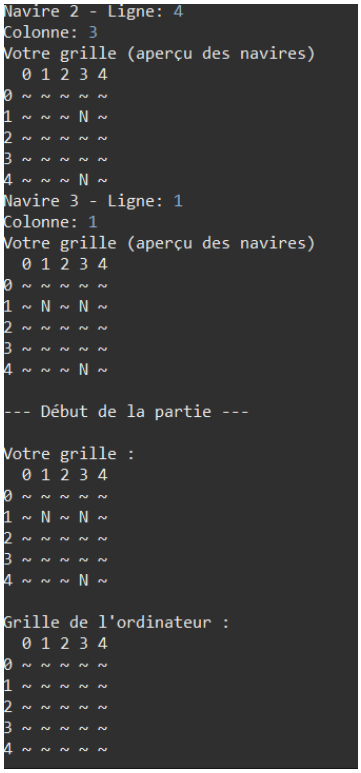
```
private static void affichagetabCache(int[][] tab, int nbcase) {
    System.out.print("   ");
    for (int col = 1; col <= nbcase; col++) {
        System.out.print(col + " ");
    }
    System.out.println();

    for (int ligne = 1; ligne <= nbcase; ligne++) {
        System.out.print(ligne + "  ");
        for (int col = 1; col <= nbcase; col++) {
            int valeur = tab[ligne - 1][col - 1];

            char c;
            if (valeur == pion_vu) {
                c = 'o'; // pion découvert
            } else if (valeur == case_sans_pion) {
                c = 'x'; // tir dans l'eau
            } else {
                c = '?'; // case non découverte
            }

            System.out.print(c + " ");
        }
        System.out.println();
    }
}
```

Cette étape consiste à cacher les pions de notre adversaire Ordi. Pour cela, on va créer une partie privée « affichagetabCache » qui va tout cacher.

Voici un petit rappel des valeurs et symbole qui sont utiliser sur la grille :

```
// Rappel :
// 0 = aucune action, aucun pion     → ?
// 1 = pion non découvert            → ?
// 2 = pion découvert                → o
// 3 = case découverte sans pion     → x
```

Voici un aperçu du jeu après la 5e étape :



Etape 6 : Le joueur part à la chasse aux pions adverse

6.1 - 6.2 : demander au joueur à quelle ligne et colonne il doit tirer son obus :

```
private void tourJoueur() {
    System.out.println("A vous tour de jouer !");

    System.out.print("À quelle ligne voulez-vous découvrir une case (1 à " + nb_case + ") ? ");
    int ligne = sc.nextInt();
    System.out.print("À quelle colonne voulez-vous découvrir une case (1 à " + nb_case + ") ? ");
    int colonne = sc.nextInt();
```

6.3 : Afficher "Tir en cours" et laisser une attente de 2 secondes pour "simuler" le tir :

```
if (ligne < 1 || ligne > nb_case || colonne < 1 || colonne > nb_case) {
    System.out.println("Coordonnées hors du plateau. Tir annulé.");
    return;
}

System.out.println("Tir en cours...");
attendre(2);

int val = tabOrdi[ligne - 1][colonne - 1];
```

Le joueur va entrer le nombre de la ligne et colonne, s'il entre un nombre supérieur, il aura se avertissement « Coordonnées hors du plateau. Tire annulée » ou sinon il aura la confirmation avec « tir en cours… » d'une attente de 2 secs pour avoir les résultats.

6.4 : Affichage du résultat :

```
// 6.4 : afficher le résultat
if (val == vide) {
    System.out.println("Raté !");
    tabOrdi[ligne - 1][colonne - 1] = case_sans_pion;
} else if (val == pion_non_vu) {
    System.out.println("Touché !");
    tabOrdi[ligne - 1][colonne - 1] = pion_vu;
    nbPionTrouveJoueur++;
} else {
    System.out.println("Tir à blanc (case déjà compromise) !");
}
```

Il aura 3 conditions du résultat :
- soit il rate il n'y a rien
- sois il touche un pion ennemi
- sois il tire sur une case déjà découvert (le tire à blancs)

Mais chaque tire fait découvrir la case ennemie.

6.5 Afficher le tableau ordi : Faites juste un appel à la procedure cree en etape 5 :

```
// 6.5 : afficher le tableau de l'ordinateur (version cachée)
System.out.println("Champ de bataille de l'ordi (vu par le joueur) :")
affichagetabCache(tabOrdi, nb_case);
```

Cette étape consiste à révéler la case découvert, tiré par le joueur.

Etape 7 : L'ordinateur cherche un pion

7.0 Afficher « Tour de l'ordi »

```
Étape 7 : l'ordinateur cherche un pion
 private void tourOrdi() {
 System.out.println("Tour de l'ordi");
```

7.1, 7.2, 7.3 : tir sur une case non découverte (valeur 0 ou 1)

```
do {
    ligne = (int) (Math.random() * nb_case) + 1;
    colonne = (int) (Math.random() * nb_case) + 1;
} while (tabJoueur[ligne - 1][colonne - 1] != vide
        && tabJoueur[ligne - 1][colonne - 1] != pion_non_vu);
```

Ceci va indiquer à l'ordi de choisir aléatoirement une case et colonne entre 1 et 5 puis la condition changera selon la case si elle contient un pion ou non du joueur.

7.4 Afficher "Tir en cours"

```
System.out.println("L'ordinateur tire en (" + ligne + ", " + colonne + ") ...");
System.out.println("Tir en cours...");
attendre(2);

int val = tabJoueur[ligne - 1][colonne - 1];
```

Ceci indique un cooldown de 2 secs lorsque l'ordi à choisi sa ligne/colonne.

7.5 Affichage de résultat

```
if (val == vide) {
    System.out.println("Raté !");
    tabJoueur[ligne - 1][colonne - 1] = case_sans_pion;
} else if (val == pion_non_vu) {
    System.out.println("Touché !");
    tabJoueur[ligne - 1][colonne - 1] = pion_vu;
    nbPionTrouveOrdi++;
}
```

Ceci va déterminer si le dire est bien sur un pion du joueur ou non. Dans le cas ou il touche, il aura « Toucher ! », sinon « Raté ! ».

7.6 Affichage de tableau joueur : même procéder que la 5e étapes

```
// 7.6 : affiche le tableau du joueur (cases découvertes)
System.out.println("Votre champ de bataille (cases découvertes) :");
affichagetabCache(tabJoueur, nb_case);
```

Etape 8 : résultat du vainqueur.

8.1 : affichage du vainqueur.

```
// Étape 8.2 : afficher le vainqueur
if (nbPionTrouveJoueur >= nb_pion) {
    System.out.println("Le joueur a gagné !");
} else {
    System.out.println("L'ordinateur a gagné !");
}
```

8.2 : menu de la fin de partie

```
private void menuPrincipal() {
    boolean rejouer = true;

    while (rejouer) {
        lancerPartie();

        System.out.print("Voulez-vous rejouer ? (o/n) : ");
        String rep = sc.next().toLowerCase();
        rejouer = rep.length() > 0 && rep.charAt(0) == 'o';
    }

    sc.close();
    System.out.println("Fin du programme.");
}
```

Et voilà ! le programme est complété et prêt à jouer désormais.

Conclusion

Ce TP m’a permis de comprendre comment structurer un programme Java complet en utilisant des tableaux, des boucles, des conditions et des procédures. La bataille navale a été un bon exercice pratique pour appliquer progressivement les notions vues en cours. J’ai pu constater qu’en organisant correctement les étapes, on arrive à créer un jeu fonctionnel et interactif. Cette réalisation m’a aidé à renforcer ma logique de programmation.