Ghassen Ben Abdeljelil

Binh Minh Tran

Diop Serigne

JEU DU MORPION

25-10-2025

Encadré par
Madame Pereira

Table des matières

1. Introduction.

Ce projet vise à développer un jeu de morpion en Java, offrant une expérience de jeu intuitive et interactive pour deux joueurs. L'objectif initial est de créer une version standard du morpion avec des fonctionnalités de base, tout en permettant l'ajout de nouvelles améliorations pour une meilleure jouabilité. Pour répondre à cet objectif, deux versions du code ont été développées :

- Morpion_base. java (180 lignes de code) - Le code qui répond aux exigences standards du projet.
- Morpion_amelioration.java (287 lignes de code) - Le code avec des améliorations.

2. Détails des fonctionnalités du code.

Cette partie est dédiée au fichier Morpion_base.java – **le code qui répond aux exigences standards du projet.**

2.1. nouveauDeplacement() :

Fonction principale :

La méthode **nouveau_deplacement** gère un nouveau mouvement d'un joueur dans un jeu.

Elle permet à l'utilisateur d'entrer les coordonnées de la ligne et de la colonne pour effectuer un coup, tout en vérifiant la validité de ce dernier.

Paramètres :

- **schema** : Un tableau 2D de type char[][] représentant la grille de jeu. Chaque élément contient soit le symbole d'un joueur ('X' ou 'O'), soit un espace vide (' ').
- **tour** : Le symbole du joueur actuel ('X' ou 'O').
- **ligne** et **colonne** : Les coordonnées saisies par le joueur pour le coup.

Fonctionnement détaillé :

1. **Initialisation du scanner** : Un objet Scanner est créé pour lire les entrées de l'utilisateur.
2. **Validation des coordonnées** :
    - Une boucle while (true) permet de demander les coordonnées jusqu'à ce qu'elles soient valides (0, 1 ou 2).

- Si les coordonnées sont invalides, un message d'erreur est affiché et le joueur est invité à entrer à nouveau ses choix.

3. **Validation de l'emplacement** :
    - Une seconde boucle while (true) vérifie si la case choisie est libre.
    - Si la case est vide, elle est marquée avec le symbole du joueur et la boucle se termine.
    - Si la case est occupée, un message est affiché pour informer le joueur, et il est invité à entrer de nouvelles coordonnées.

2.2. aGagne() :

Fonction principale :

La méthode **aGagne** vérifie si un joueur a remporté la partie dans un jeu de morpion. Elle évalue les lignes, les colonnes et les diagonales pour déterminer si le joueur actuel a réussi à aligner ses symboles.

Paramètres :

- **schema** : Un tableau 2D de type char[][] représentant l'état actuel du jeu. Chaque élément peut contenir un symbole de joueur ('X' ou 'O') ou un espace vide.
- **joueur** : Le symbole du joueur à vérifier (par exemple, 'X' ou 'O').

Fonctionnement détaillé :

1. **Vérification des lignes et colonnes** :
    - Une boucle for parcourt chaque ligne et colonne du tableau.
    - Pour chaque ligne, elle vérifie si les trois cases contiennent le même symbole du joueur. Si c'est le cas, la méthode renvoie true.
    - De même, elle effectue la vérification pour les colonnes.
2. **Vérification des diagonales** :
    - La méthode vérifie ensuite les deux diagonales :
        - La diagonale principale (de l'angle supérieur gauche à l'angle inférieur droit).
        - La diagonale secondaire (de l'angle supérieur droit à l'angle inférieur gauche).

- Si l'une de ces diagonales contient le même symbole du joueur, la méthode renvoie également true.

3. **Aucun gagnant** :
    - Si aucune des conditions ci-dessus n'est remplie, la méthode retourne false, indiquant que le joueur n'a pas encore gagné.

2.3. main() :

Fonction principale :

La méthode **main** constitue le point d'entrée du programme et orchestre l'exécution du jeu de morpion. Elle initialise le jeu, gère le déroulement des tours des joueurs, et vérifie les conditions de victoire.

Déclarations et initialisations :

- **Variables ligne et colonne** : Ces variables sont utilisées pour stocker les coordonnées saisies par l'utilisateur pour effectuer un mouvement.
- **Scanner** : Un objet de la classe Scanner est créé pour lire les entrées de l'utilisateur depuis la console.

Fonctionnement détaillé :

1. **Initialisation de la grille** :
    - La méthode **initialiser(schema)** est appelée pour configurer la grille de jeu, en s'assurant que toutes les cases soient vides au début de la partie.
2. **Boucle principale du jeu** :
    - Une boucle while (true) est mise en place pour permettre un déroulement continu du jeu jusqu'à ce qu'une condition de victoire ou un match nul soit atteint.
3. **Affichage de la grille** :
    - À chaque itération, l'état actuel de la grille est affiché via la méthode **voir(schema)**, permettant aux joueurs de visualiser le plateau de jeu.
4. **Demande de coordonnées** :
    - Une boucle while (true) permet de demander à l'utilisateur de saisir des coordonnées valides pour la ligne et la colonne.
    - Les saisies sont lues avec scanner.nextInt(), et si l'utilisateur entre une valeur invalide (autre qu'un entier), une exception

InputMismatchException est levée, et un message d'erreur est affiché. L'entrée invalide est ensuite supprimée pour éviter des boucles infinies.

5. **Déplacement et vérification de la victoire** :
    - La méthode **nouveau_deplacement(schema, tour, ligne, colonne)** est appelée pour effectuer le coup sur la grille.
    - Le compteur de mouvements est incrémenté.
    - La méthode **aGagne(schema, tour)** est ensuite appelée pour vérifier si le joueur actuel a gagné. Si c'est le cas, le message de victoire est affiché et la boucle est interrompue.
6. **Vérification d'égalité** :
    - Si tous les mouvements ont été effectués sans qu'il y ait un gagnant (après 9 mouvements), un message indiquant l'égalité est affiché, et la boucle est également interrompue.
7. **Changement de joueur** :
    - À la fin de chaque tour, le symbole du joueur courant est alterné entre 'X' et 'O'.

2.4. initialiser() :

Fonctionnalité :

La méthode **initialiser** est responsable de la configuration initiale de la grille de jeu pour le morpion. Elle garantit que toutes les cases de la grille sont prêtes à recevoir les symboles des joueurs (soit 'X' soit 'O'), en les initialisant à un état vide.
Détails de l'implémentation

Paramètre :

- **schema** : Ce paramètre est un tableau 2D de type char, représentant la grille de jeu de dimensions 3x3.

Fonctionnement :

1. **Boucle d'initialisation** :
    - La méthode utilise deux boucles imbriquées pour parcourir chaque case de la grille.
    - La première boucle, indexée par i, itère à travers les lignes (0 à 2).
    - La deuxième boucle, indexée par j, itère à travers les colonnes (0 à 2).
2. **Initialisation des cases** :

- À chaque itération des boucles, chaque case de la grille est assignée à un espace vide (' ').
- Cela assure que chaque case est prête à être occupée par un symbole de joueur lors des tours suivants.

2.5. voir() :

Fonctionnalité :

La méthode **voir** est chargée de l'affichage de la grille de jeu pour le morpion. Elle présente les éléments du tableau dans un format visuel qui facilite la compréhension de l'état actuel du jeu. Cette visualisation est cruciale pour que les joueurs puissent suivre leurs mouvements et ceux de leur adversaire.

Paramètre :

- **schema** : Ce paramètre est un tableau 2D de type char[][], représentant l'état de la grille de jeu. La méthode suppose que la grille est de dimensions 3x3.

Fonctionnement :

1. **Affichage du cadre de la grille** :
   - La méthode commence par afficher une ligne de séparation ("-------------"), qui marque le début de la grille.
2. **Boucles d'affichage** :
   - La première boucle, indexée par i, parcourt chaque ligne de la grille (0 à 2).
   - Pour chaque ligne, la méthode commence par imprimer un caractère |, qui représente le début d'une ligne dans la grille.
3. **Affichage des colonnes** :
   - La deuxième boucle, indexée par j, parcourt chaque colonne de la ligne en cours.
   - À chaque itération, le contenu de la case correspondante (schema[i][j]) est affiché avec un espace et un séparateur | autour, offrant ainsi une visualisation claire des symboles.
4. **Fin de la ligne** :
   - Après avoir affiché tous les éléments de la ligne, la méthode imprime une nouvelle ligne de séparation, indiquant la fin de la ligne en cours.
5. **Répétition** :
   - Ce processus se répète pour chaque ligne de la grille, assurant que chaque case est correctement visualisée dans un format de grille.

3. Résultat.

3.1. Gagnant :

À chaque fois que la diagonale ou trois cases consécutives verticales ou horizontales sont remplacées par le symbole d'un joueur, ce joueur remporte la victoire.

```
-------------
| O | X | X |
-------------
| O | X | X |
-------------
| X | O | O |
-------------
FÉLICITATIONS X, TU AS GAGNÉ !!!
```

3.2. Égalité :

Lorsque le nombre de mouvements atteint 9 sans qu'il y ait de gagnant, le résultat est déclaré nul.

```
-------------
| X | O | O |
-------------
| O | X | X |
-------------
| X | X | O |
-------------
Rien à faire... EGALITÉ !
```

3.3. Cas d’erreur :

3.3.1. L'erreur de valeur invalide.

Toute entrée qui n'est pas un nombre entier entraînera un message d'erreur indiquant une entrée invalide, et le système demandera à nouveau une saisie correcte.

```
[Tour X] Entrez la ligne (0, 1 ou 2) et la colonne (0, 1 ou 2) :
1
-
Entrée invalide. Veuillez entrer un nombre entier.

[Tour X] Entrez la ligne (0, 1 ou 2) et la colonne (0, 1 ou 2) :
0
0
Ligne: 0, Colonne: 0
-------------
| X |   |   |
-------------
|   |   |   |
-------------
|   |   |   |
-------------
```

3.3.2. L'erreur sur les valeurs hors limites.

Si l'entrée se situe en dehors de la plage [0, 1, 2], le système demandera à l'utilisateur de saisir à nouveau une valeur valide.

```
[Tour O] Entrez la ligne (0, 1 ou 2) et la colonne (0, 1 ou 2) :
-2
3
Ligne: -2, Colonne: 3
0, 1 et 2 sont les numéros possibles que vous pouvez choisir.

[Tour O] Entrez la ligne (0, 1 ou 2) et la colonne (0, 1 ou 2) :
1
1
-------------
| X |   |   |
-------------
|   | O |   |
-------------
|   |   |   |
-------------
```

3.3.3. L'erreur liée aux cases déjà occupées.

Si l'utilisateur saisit des coordonnées déjà occupées, le système lui demandera de saisir de nouvelles coordonnées.

```
[Tour X] Entrez la ligne (0, 1 ou 2) et la colonne (0, 1 ou 2) :
1
1
Ligne: 1, Colonne: 1
-------------
|   |   |   |
-------------
|   | X |   |
-------------
|   |   |   |
-------------
[Tour O] Entrez la ligne (0, 1 ou 2) et la colonne (0, 1 ou 2) :
1
1
Ligne: 1, Colonne: 1
Place déjà occupée, essayez à nouveau.
```

4. Difficultés rencontrées et solutions.

Dans le cadre du développement de ce projet, plusieurs difficultés ont été rencontrées, nécessitant des ajustements et des solutions spécifiques :

- **Vérification des entrées utilisateurs** : Initialement, des entrées invalides provoquaient des erreurs. Pour y remédier, des boucles de validation ont été ajoutées afin de garantir que seules des entrées valides soient acceptées.
- **Gestion de l'occupation des cases** : La vérification des cases occupées a été ajoutée pour garantir que chaque case ne puisse être occupée qu'une seule fois.
- **Fin de partie et détection de victoire** : La fonction aGagne() a été optimisée pour s'assurer que toutes les possibilités de victoire (lignes, colonnes, diagonales) soient correctement couvertes.

5. Conclusion.

En conclusion, le projet de morpion en Java a permis de mettre en pratique des concepts essentiels de programmation tels que la gestion des entrées utilisateur, la validation des données, et le contrôle du flux d'exécution. La version améliorée a été conçue pour enrichir l'expérience de jeu en offrant plus de compétitivité et en introduisant de nouvelles fonctionnalités.

Les perspectives d'amélioration futures incluent l'ajout d'une interface graphique pour rendre le jeu plus attrayant, ainsi que l'implémentation d'une intelligence artificielle permettant à un joueur de jouer contre l'ordinateur, ce qui augmenterait la complexité et l'intérêt du jeu.

6. Amélioration.

Cette section est dédiée au fichier Morpion_amelioration.java – **Le code améliore les standards de base du projet.**

6.1. Idée.

Pour augmenter la compétitivité du jeu, il est nécessaire d'organiser des matchs sur plusieurs parties. Avec cette amélioration, l'utilisateur a la possibilité de choisir un niveau de compétition en optant pour 3 ou 5 parties, tout en maintenant l'option 1 pour ceux qui préfèrent jouer rapidement.

6.1.1. Nouvelles fonctionnalités.

- Le système affichera le match actuel par rapport au nombre total de matchs.
- Le système stocke l'historique des jeux afin de déterminer le gagnant final.
- Le système identifie rapidement le gagnant en comptant le nombre de victoires consécutives par joueur (par exemple : si le nombre de matchs est égal à 5, alors un joueur qui gagne consécutivement 3 fois est déclaré gagnant).

6.2. Algorithme.

Cette amélioration effectue des modifications uniquement sur la méthode main().

Algorithme du jeu :

1. **Initialisation :**
    - Demander à l'utilisateur de saisir le nombre de jeux (1, 3 ou 5).
    - Valider l'entrée. Si valide, poursuivre ; sinon, redemander.
    - Initialiser un tableau pour stocker les résultats des jeux.
    - Définir des compteurs pour les victoires consécutives de X, O, et les égalités.
2. **Boucle principale :** Pour chaque jeu (jusqu'à nb_jeu) :
    - Initialiser la grille de jeu.

- Remettre à zéro le compteur de mouvements.
- Boucle de jeu :
  - Afficher la grille actuelle.
  - Demander à l'utilisateur d'entrer les coordonnées (ligne et colonne).
  - Valider les entrées. Si valide, effectuer le mouvement.
  - Incrémenter le compteur de mouvements.
  - Vérifier s'il y a un gagnant :
    - Si X ou O gagne, afficher le message de victoire, stocker le résultat, et mettre à jour les compteurs de victoires.
    - Vérifier les victoires consécutives et décider si le jeu doit se terminer.
  - Si tous les mouvements sont faits sans gagnant, déclarer une égalité et vérifier si le jeu doit se terminer.

3. **Changement de joueur :** Alterner le tour entre X et O.
4. **Vérification finale :** Après tous les jeux, vérifier qui est le gagnant final en comptant le nombre de victoires de X, O, et les égalités :
   - Si un joueur a plus de 60 % de victoires, l'annoncer comme gagnant.
   - Si plus de 60 % des jeux sont des égalités, déclarer une égalité.
5. **Fin du jeu :** Afficher le message de fin du jeu.

6.3. Résultat

Le résultat est affiche à la fin de chaque série de matchs, indiquant les victoires pour chaque joueur et éventuellement l'égalité :

```
-------------
| X | O | O |
-------------
| X | X | O |
-------------
|   | X | O |
-------------
FÉLICITATIONS O, TU AS GAGNÉ !!!
Jeu: 1 sur 5
```

```
-------------
| O |   |   |
-------------
| X | O |   |
-------------
|   | X | O |
-------------
FÉLICITATIONS O, TU AS GAGNÉ !!!
Jeu: 5 sur 5

Gagnant est O avec 3.0 gagnés sur 5 jeu(x).
FIN DU JEU DE MORPION
```

Résultat d'égalité : 2 matchs consécutifs dans un jeu de 3.

```
-------------
| X | O | X |
-------------
| X | O | O |
-------------
| O | X | O |
-------------
Rien à faire... ÉGALITÉ !
Jeu: 2 sur 3

Égalité ! Fin du jeu.
```