Hippolyte CAMUS-JUGIEAU
Loric GRENIER
Paul SAGUEZ

Documentation Technique - Puissance 4

1. Introduction

1.1 Présentation du Projet

Ce projet implémente le jeu classique **Puissance 4** en Java avec une interface graphique Swing. Le jeu permet à deux joueurs de s'affronter en tour par tour sur un plateau de 6 lignes et 7 colonnes.

1.2 Objectifs de la Documentation

Cette documentation technique vise à :

- Décrire l'architecture logicielle du système
- Expliquer la conception et l'implémentation des classes
- Documenter les algorithmes de jeu
- Fournir des informations sur les tests et la qualité du code
- Faciliter la maintenance et l'évolution du projet

1.3 Technologies Utilisées

- **Langage** : Java 17
- **Build Tool** : Maven 3.x
- **Framework de Tests** : JUnit 5.10.0
- **Interface Graphique** : Java Swing

2. Architecture du Système

2.1 Vue d'Ensemble

L'application suit une architecture **MVC (Model-View-Controller)** simplifiée :

- **Model** : Classes Game, Player, PlayerColor
- **View** : Classe Plateau (interface graphique)
- **Controller** : Logique intégrée dans Plateau et Game

2.2 Structure des Packages

```
game/
├── Game.java # Logique principale du jeu
├── Player.java # Représentation d'un joueur
├── PlayerColor.java # Énumération des couleurs
├── board.java # Interface graphique (Plateau)
└── Main.java # Point d'entrée de l'application
```

2.3 Diagramme de Classes Simplifié

```
Player
├── name: String
├── couleur: PlayerColor
├── id: int
└── méthodes: getName(), getCouleur(), getId(), setId()

PlayerColor (enum)
├── ROUGE
└── JAUNE

Game
├── plateau: int[][]
├── playerBoards: HashMap<Player, int[][]>
├── activePlayer: Player
├── lastOne: int[]
├── lastRow: int
└── lastCol: int

Plateau extends JPanel
├── game: Game
└── gestion des événements souris
```

3. Conception des Classes

3.1 Classe Player

Responsabilités

- Représenter un joueur avec ses attributs (nom, couleur, identifiant)
- Fournir l'accès aux propriétés du joueur

Attributs

```
private String name; // Nom du joueur
private PlayerColor couleur; // Couleur des pions (ROUGE/JAUNE)
private int id; // Identifiant unique (1 ou 2)
```

Méthodes Principales

- `Player(String name, PlayerColor couleur, int id)` : Constructeur
- `String getName()` : Retourne le nom du joueur
- `PlayerColor getCouleur()` : Retourne la couleur du joueur
- `int getId()` : Retourne l'identifiant du joueur
- `void setId(int id)` : Modifie l'identifiant du joueur

Caractéristiques de Conception

- **Immutabilité partielle** : Seul l'ID peut être modifié après création
- **Encapsulation** : Tous les attributs sont privés avec des getters
- **Documentation Javadoc** : Classe entièrement documentée

3.2 Énumération PlayerColor

Responsabilités

- Définir les couleurs disponibles pour les joueurs
- Assurer la type-safety pour les couleurs

Valeurs

```
ROUGE, // Couleur du joueur 1
JAUNE // Couleur du joueur 2
```

3.3 Classe Game

Responsabilités

- Gérer l'état du jeu (plateau, joueurs, tour actuel)
- Implémenter la logique de jeu (placement des pions, vérification de victoire)
- Contrôler le déroulement des tours

Attributs Principaux

```
public int[][] plateau; // Grille de jeu 6x7
public HashMap<Player, int[][]> playerBoards; // Plateaux individuels (non utilisé)
public Player activePlayer; // Joueur dont c'est le tour
public int[] lastOne; // Dernière ligne occupée par colonne
public int lastRow, lastCol; // Position du dernier pion placé
```

Méthodes Principales

Constructeur

```
public Game(Player player1, Player player2)
```

- Initialise le plateau 6x7 avec des zéros
- Configure les joueurs et définit player1 comme joueur actif
- Initialise le tableau lastOne à -1 pour toutes les colonnes

Placement de Pion

```
public boolean play(int col)
```

- **Paramètre** : col - numéro de colonne (0-6)
- **Retour** : true si le placement est valide, false sinon
- **Logique** :

1. Vérifie la validité de la colonne ($0 \leq col < 7$)
2. Vérifie si la colonne n'est pas pleine
3. Place le pion à la première position libre
4. Met à jour les variables de suivi

Changement de Joueur

```
public void switchActivePlayer()
```

- Alterne entre les deux joueurs enregistrés
- Utilise l'ID pour identifier le joueur suivant

Vérification de Match Nul

```
public boolean checkDraw()
```

- Vérifie si toutes les colonnes sont pleines
- Retourne true si le plateau est complet

Vérification de Victoire

```
public boolean checkVictory()
```

- Vérifie les 4 directions possibles : horizontale, verticale, diagonales
- Utilise la méthode auxiliaire countDirection()
- Retourne true si 4 pions alignés sont trouvés

Méthode Auxiliaire

```
private int countDirection(int row, int col, int dx, int dy, int playerId)
```

- Compte les pions consécutifs dans une direction donnée
- Utilisée par checkVictory() pour vérifier les alignements

3.4 Classe Plateau

Responsabilités

- Afficher l'interface graphique du jeu

- Gérer les interactions utilisateur (clics souris)
- Dessiner le plateau et les pions

Attributs

```
static final int ROWS = 6; // Nombre de lignes
static final int COLS = 7; // Nombre de colonnes
private Game game; // Référence vers la logique de jeu
```

Méthodes Principales

Constructeur

```
public Plateau(Game game)
```

- Initialise la référence vers l'objet Game
- Configure le gestionnaire d'événements souris

Gestion des Événements

- **MouseListener** : Capture les clics pour placer les pions
- **Calcul de colonne** : Convertit les coordonnées souris en numéro de colonne
- **Gestion des états** : Victoire, match nul, colonne pleine

Rendu Graphique

```
protected void paintComponent(Graphics g)
```

- Dessine le plateau bleu avec des cercles
- Affiche les pions rouge et jaune selon l'état du jeu
- Utilise l'antialiasing pour un rendu de qualité

4. Logique de Jeu

4.1 Règles Implémentées

Placement des Pions

1. Les pions tombent par gravité dans la colonne choisie
2. Un pion ne peut être placé que dans une colonne non pleine
3. Les pions s'empilent du bas vers le haut

Conditions de Victoire

Un joueur gagne en alignant **4 pions consécutifs** dans l'une des directions :

- **Horizontale** : 4 pions sur la même ligne

- **Verticale** : 4 pions dans la même colonne
- **Diagonale montante** : 4 pions en diagonale (/)
- **Diagonale descendante** : 4 pions en diagonale ()

Condition de Match Nul

Le jeu se termine par un match nul si le plateau est entièrement rempli sans qu'aucun joueur n'ait aligné 4 pions.

4.2 Algorithmes Clés

Algorithme de Vérification de Victoire

```
public boolean checkVictory() {
int playerId = (activePlayer.getId() == 1) ? 1 : 2;

int[][] directions = {
{0, 1}, // Horizontale
{1, 0}, // Verticale
{1, 1}, // Diagonale descendante
{1, -1} // Diagonale montante
};

for (int[] dir : directions) {
int count = 1; // Le pion actuel
count += countDirection(lastRow, lastCol, dir[0], dir[1], playerId);
count += countDirection(lastRow, lastCol, -dir[0], -dir[1], playerId);

if (count >= 4) {
return true;
}
}
return false;
}
```

Optimisations

- **Vérification locale** : Seule la position du dernier pion placé est vérifiée
- **Arrêt précoce** : Dès qu'un alignement de 4 est trouvé, l'algorithme s'arrête
- **Directions symétriques** : Chaque direction est vérifiée dans les deux sens

5. Interface Utilisateur

5.1 Architecture Graphique

L'interface utilise **Java Swing** avec les composants suivants :

- JFrame : Fenêtre principale
- JPanel personnalisé (Plateau) : Zone de jeu
- JOptionPane : Messages de victoire/match nul

5.2 Rendu Visuel

Dimensions et Couleurs

```
static final int ROWS = 6;
static final int COLS = 7;

// Couleurs utilisées
Color.BLUE // Fond du plateau
Color.RED // Pions du joueur 1
Color.YELLOW // Pions du joueur 2
Color.WHITE // Cases vides
```

Calculs de Positionnement

```
int cellWidth = getWidth() / COLS;
int cellHeight = getHeight() / ROWS;
int cellSize = Math.min(cellWidth, cellHeight);

int boardWidth = COLS * cellSize;
int xOffset = (getWidth() - boardWidth) / 2; // Centrage horizontal
int yOffset = getHeight() - boardHeight; // Alignement en bas
```

5.3 Interactions Utilisateur

Gestion des Clics

1. **Détection de colonne** : Conversion coordonnées souris → numéro de colonne
2. **Validation** : Vérification que le clic est dans la zone de jeu
3. **Placement** : Appel à game.play(col)
4. **Feedback** : Mise à jour visuelle et messages

Messages Utilisateur

- **Victoire** : "Le joueur [nom] a gagné !"
- **Match nul** : "Match nul !"
- **Erreur** : "Colonne pleine !"

6. Configuration et Déploiement

6.1 Prérequis Système

Configuration Minimale

- **Java** : JDK 17 ou supérieur
- **Maven** : 3.6.0 ou supérieur
- **Mémoire** : 512 MB RAM minimum
- **Système** : Windows, macOS, ou Linux avec support Java