



***Rapport de projet***

***Implémentation du jeu Puissance 4 en Java***

Nathan Talbot

***Introduction***

Ce rapport a pour but de présenter le projet réalisé dans le cadre du cours de Java, lors de ma quatrième année à Polytech Clermont. Le projet consistait en l'implémentation du jeu "Puissance 4" en utilisant la programmation orientée objet (POO).

Pour ce faire, plusieurs classes Java ont été modélisées afin de représenter les différentes entités du jeu, telles que la grille, les cases et leurs coordonnées.

Afin d'assurer le versionnement du projet, de le partager et d'obtenir une sauvegarde cloud, un repository GitHub a été créé : [Lien vers le repository](https://github.com/Nathan7128/projet-puissance4).

***Annonce du plan***

Dans un premier temps, je présenterai brièvement les règles du jeu Puissance 4 afin d’en poser le contexte.

Ensuite, j’expliquerai le fonctionnement du programme en détaillant la logique employée pour modéliser le jeu à l’aide de la programmation orientée objet (POO).

Dans la troisième partie, je décrirai en détail les différentes classes Java qui composent le programme, en mettant en avant leurs attributs et méthodes.

Enfin, je terminerais par une conclusion sur ce projet.

Table des matières

[I. Présentation du jeu 4](#_Toc189754124)

[II) Fonctionnement du programme 5](#_Toc189754125)

[III. Présentation des classes 8](#_Toc189754131)

[1) La classe Coordonnees 8](#_Toc189754132)

[2) La classe mère Case et ses classes dérivées 9](#_Toc189754133)

[3) Classe Grille 10](#_Toc189754134)

[4) Classe Partie 11](#_Toc189754135)

[5) Classe Main 13](#_Toc189754136)

[Conclusion 15](#_Toc189754137)

I. Présentation du jeu

Le Puissance 4 est un jeu de stratégie pour deux joueurs. Chaque joueur insère à tour de rôle un pion dans une colonne de la grille. Le premier à aligner un certain nombre de pions (horizontalement, verticalement ou en diagonale) remporte la partie.

Dans l'implémentation réalisée, il est possible de configurer :

* La largeur de la grille (nombre de colonnes)
* La hauteur de la grille (nombre de lignes)
* Le nombre de jetons à aligner pour gagner

II) Fonctionnement du programme

Ce programme est une application console, ce qui signifie que le jeu Puissance 4 s’exécute dans un terminal, où la grille est affichée et les joueurs interagissent via des entrées clavier.

1. Modélisation de la grille

La grille du jeu (cf. image 1) est définie par une largeur et une hauteur et est implémentée à l’aide d’un HashMap en Java.

Un HashMap est une structure clé/valeur où :

* La clé est un objet de type Coordonnees, représentant une position dans la grille avec une ligne et une colonne.
* La valeur est un objet de type Case, qui représente une case du jeu.

Bien qu’il aurait été possible d’implémenter la grille avec une matrice, le choix d’un HashMap a été imposé par les consignes du projet.

Grâce à cette structure, une case peut être retrouvée rapidement en fournissant un objet Coordonnees. Pour garantir un accès efficace, la classe Coordonnees redéfinit la méthode hashCode() ainsi que equals() afin d'assurer une bonne gestion des clés dans le HashMap. Le fonctionnement détaillé de ces méthodes sera abordé dans la section dédiée à la classe Coordonnees.

2. Représentation des cases

Une case de la grille peut être de trois types :

* Vide : le joueur n’a pas encore joué à cet emplacement.
* Jaune : la case contient un jeton du Joueur 1.
* Rouge : la case contient un jeton du Joueur 2.

Ces trois types de cases sont représentés par des classes qui héritent d’une classe mère Case, définissant les comportements communs.

3. Déroulement d’une partie

Avant de commencer, les joueurs peuvent configurer la partie en choisissant :

* La largeur de la grille (nombre de colonnes).
* La hauteur de la grille (nombre de lignes).
* Le nombre de jetons alignés requis pour gagner.

La partie oppose deux joueurs :

* Joueur 1 (pions jaunes).
* Joueur 2 (pions rouges).

Le jeu se déroule en alternance : chaque joueur place un pion dans une colonne de son choix. Lorsqu’un joueur joue un coup, un nouvel élément est ajouté au HashMap sous la forme :

(Coordonnees de la case jouée, CaseJaune ou CaseRouge)

4. Vérification des conditions de victoire

Après chaque coup, le programme vérifie si la partie est terminée. Deux cas de fin de partie sont possibles :

1. La grille est pleine, auquel cas la partie se termine sur une égalité.
2. Un joueur a aligné le nombre requis de pions, auquel cas il remporte la victoire.

Pour détecter un alignement, le programme vérifie quatre directions à partir de chaque case jouée :

* Alignement horizontal vers la droite.
* Alignement vertical vers le bas/
* Diagonale bas-droite.
* Diagonale bas-gauche.

La vérification est effectuée en parcourant la grille de gauche à droite et de haut en bas afin d’analyser tous les alignements possibles.

5. Fin de partie et nouvelle manche

Lorsque la partie se termine, les joueurs peuvent choisir de rejouer.  
Si une nouvelle partie est lancée, ils peuvent redéfinir la taille de la grille et le nombre de pions à aligner, puis le jeu redémarre avec une nouvelle instance de partie.

III. Présentation des classes

1) La classe Coordonnees

La classe Coordonnees représente les positions des cases dans la grille avec deux attributs :

* i : indice de la ligne
* j : indice de la colonne

La position (0,0) correspond à la case en haut à gauche de la grille.

Dans l’implémentation de la classe Grille, les objets Coordonnees servent de clés dans un HashMap pour stocker les cases. Afin d'assurer un accès efficace, il a été nécessaire de redéfinir:

* hashCode() : génère un code permettant de regrouper les clés dans le HashMap.
* equals(Object obj) : vérifie si deux objets Coordonnees ont les mêmes attributs.

Le HashMap regroupe ses clés en fonction de leur code de hachage. Lors d'une recherche, l'interpréteur identifie d’abord le sous-ensemble de clés ayant le même code, puis compare les éléments avec equals().

Ici, la méthode hashCode() utilise Objects.hash(i, j), qui génère un code basé sur les coordonnées.

La méthode equals() compare deux objets Coordonnees et retourne vrai si leurs attributs i et j sont identiques, sinon faux.

Ces redéfinitions optimisent l'utilisation des objets Coordonnees dans un HashMap et assurent un accès rapide aux cases de la grille.

2) La classe mère Case et ses classes dérivées

La classe Case constitue la base de la hiérarchie des classes modélisant les cases de la grille. Elle regroupe les attributs communs à toutes ses classes dérivées, notamment un attribut p de type Coordonnees, qui représente la position de la case dans la grille.

Trois classes héritent de cette classe :

* CaseVide qui représente une case où aucun pion n’a encore été joué.
* CaseJaune qui représente une case occupée par un pion du premier joueur.
* CaseRouge qui représente une case occupée par un pion du deuxième joueur.

La classe de base définit également une méthode abstraite « afficher » que chaque classe dérivée doit redéfinir en fonction du type de case. L’affichage est géré de la manière suivante :

* Une case vide est affichée comme un espace.
* Une case occupée par le premier joueur est représentée par la lettre X.
* Une case occupée par le deuxième joueur est représentée par la lettre O.

Grâce à cette structure, l’affichage des cases est personnalisé tout en respectant une architecture orientée objet claire et extensible.

3) Classe Grille

La classe Grille modélise la grille du jeu en stockant l’ensemble des cases dans une structure de type HashMap. Chaque case est associée à un objet de type Coordonnees qui sert de clé dans cette structure.

La grille possède également une largeur et une hauteur définies lors de son initialisation.

Lors de la création d’une grille, son constructeur la remplit de cases vides. Pour cela, il associe à chaque position un objet Coordonnees et lui attribue une case vide dans le HashMap.

Une méthode permet d’afficher la grille sous forme de chaîne de caractères, respectant le format requis.

A chaque fois ou l’on souhaite parcourir l’ensemble des cases de la grille, on créé donc une 2 boucles « for » imbriquées (une permettant d’instancier les indices i = 0, …, hauteur de la grille, et l’autre les colonnes j = 0, …, largeur).

Pour chaque couple (i, j), on attribut ces coordonnées à un objet de type Coordonnées, qui nous permet de retrouver la case associée dans le HashMap.

Cette classe propose également des méthodes pour modifier et récupérer une case en fonction de ses coordonnées.

Enfin, la classe Grille possède la méthode « pleine », qui permet de vérifier si cette dernière est remplie de cases (non vides) ou non.

Elle renvoie donc true si la grille est pleine et false sinon.

Pour ce faire, elle traverse l’ensemble des cases de la grille, et vérifie pour chacune d’entre elles si elles sont une instance des classes Case jaune ou Case rouge, à l’aide de la méthode « instanceof ».

Cette méthode nous a donc évité d’assigner un attribut « type » à la classe case afin de retrouver le type d’un objet instancié, comme ça aurait été le cas en C++. En effet, le C++ ne propose pas de méthode permettant de trouver le type d’un objet instancié.

4) Classe Partie

C’est vers cette dernière classe que convergent toutes les classes présentées précédemment.

Elle implémente toutes les méthodes permettant le déroulement de la partie.

Lors de son instanciation, elle reçoit en paramètre les spécificités du jeu, à savoir la largeur et la hauteur de la grille, ainsi que le nombre de pions à aligner pour gagner. Le constructeur crée alors une grille aux dimensions indiquées, qui devient un attribut de la partie.

La grille possède également un attribut « joueur », qui permet de savoir quel joueur doit jouer à un instant t.

En effet, après chaque coup, le programme vérifie si la partie est terminée, c’est-à-dire si la grille est pleine ou si le joueur qui a joué a aligné le nombre de pions requis. Par exemple, si c’est le joueur 1 qui a joué, on vérifiera uniquement si des cases jaunes sont alignées.

On regardera l’alignement des cases jaunes seulement dans ce cas, et non les cases rouges car c’est le joueur 1 qui a joué donc le joueur 2 ne peut pas avoir gagné après le coup du joueur 1.

C’est la méthode « partieFinie » qui permet de savoir si la partie est terminée (c’est-à-dire si la grille est pleine ou si un joueur a gagné), tandis que c’est la méthode « gagne » qui permet de savoir si le joueur qui vient de jouer a gagné.

Le fonctionnement de cette méthode est le suivant : on itère chaque case de la grille, et on regarde si c’est bien une instance du type de case associé au joueur qui vient de jouer. Si c’est le cas, on va alors tester tous les alignements possibles pour cette case (alignement vers la droite, vers le bas, en diagonale bas droite et en diagonale bas gauche). Comme lors de chaque itération des classes de la grille, on regarde en premier la case située ligne 0, colonne 0, puis celle ligne 0, colonne 1, etc.

On a créé une méthode pour tester chaque alignement possible : « ligneDroite », « ligneBas », « ligneBasDroite » et « ligneBasGauche ».

Ces méthodes parcourent à l’aide d’une boucle « while » les cases de la grille dans la direction correspondante à partir de la case itérée par la méthode « gagne ».

Cette boucle s’arrête dans 3 cas :

* On a atteint le bord de la grille
* Le nombre de pions aligné avec la case instanciée (stocké dans la variable « compteur ») est supérieur ou égal au nombre de pions à aligner pour gagner
* La case itérée par cette boucle « while » n’est pas une instance de la classe de la case itérée par la méthode « gagne ».

Ces méthodes renvoient true si la variable « compteur » est supérieur ou égal au nombre de pions à aligner pour gagner.

Enfin, cette classe permet également à un joueur de jouer un coup en insérant un pion dans la colonne choisie, à la position la plus basse disponible. Une dernière méthode vérifie si un coup est possible en s’assurant que la colonne choisie n’est pas entièrement remplie.

5) Classe Main

Le lancement de ce programme se réalise par l’exécution du fichier « Main.java ».

Nous créons une variable booléenne « jouer », qui indique si l’utilisateur désigne jouer ou non. De base, cette variable vaut donc true, et on demande à la fin de chaque partie si l’utilisateur désir recommencer une partie, en modifiant la variable « jouer » conditionnellement au choix de l’utilisateur.

Le programme s’exécute tant que cette variable est vraie, ce qui est géré par une boucle while.

Au début de chaque partie, l’utilisateur doit saisir les dimensions de la grille ainsi que le nombre de pions à aligner pour gagner. Des vérifications sont effectuées pour s’assurer que :

* La largeur et la hauteur sont bien supérieures à zéro.
* Le nombre de pions à aligner ne dépasse pas le plus petit côté de la grille, garantissant ainsi qu’il est possible de former un alignement valide.

Après cette saisie des spécificités de la partie, nous affichons une première fois la grille (remplie de cases vides).

Ensuite, tant que la partie n’est pas terminée :

1. Le joueur actif choisit une colonne.
2. Le programme vérifie si le coup est possible à l’aide de la méthode « coupPossible ». Si le coup est invalide, le joueur doit choisir une autre colonne.
3. Le pion est placé dans la grille.
4. Le programme vérifie si la partie est terminée.
5. Si ce n’est pas le cas, on change le joueur de la partie à l’aide de la méthode « changerJoueur » de la classe Partie.
6. La grille affichée.

Une fois que la partie est terminée, on regarde si le joueur qui vient de jouer a gagné.

Si c’est le cas, on affiche un message pour féliciter le joueur en question.

Sinon (si la grille est pleine), on affiche un message pour indiquer que la partie s’est terminée par un match nul.

Enfin, on demande à l’utilisateur s’il souhaite recommencer une partie, ce qui conditionne l’arrêt du programme ou non.

Conclusion

En conclusion, ce projet d'implémentation du jeu Puissance 4 en Java a permis d'appliquer les principes de la programmation orientée objet (POO) pour concevoir un jeu fonctionnel et modulaire. Grâce à une architecture bien définie, incluant des classes spécifiques pour gérer les coordonnées des cases, la grille, les différents types de cases, et la gestion du déroulement de la partie, le projet respecte les bonnes pratiques de développement logiciel.

L'utilisation d'un HashMap pour la gestion de la grille et des objets Coordonnees a permis d'optimiser l'accès aux cases tout en respectant les exigences du projet. La structure du programme est flexible et extensible, avec la possibilité de modifier les dimensions de la grille et le nombre de pions à aligner pour gagner. La vérification des conditions de victoire et de fin de partie a été soigneusement implémentée pour assurer un jeu fluide et sans erreur.

Enfin, cette implémentation m’a permis de renforcer mes compétences en Java et d'approfondir mes connaissances en programmation orientée objet, tout en développant une meilleure compréhension du versionnement avec GitHub. Le projet a été une occasion précieuse de mettre en pratique la théorie abordée en cours et d'acquérir des compétences utiles dans le cadre de futurs projets de développement.