**Rapport de projet : Implémentation du jeu Puissance 4 en Java**

**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).

**Plan du rapport**

1. Présentation du jeu "Puissance 4"
2. Explication du fonctionnement du programme et de sa logique de modélisation en POO
3. Détail des différentes classes Java, avec précision sur leurs attributs et méthodes
4. Difficultés rencontrées lors de l'implémentation

**I. Présentation du jeu**

Le Puissance 4 est un jeu de stratégie pour deux joueurs. Chaque joueur insère à tour de rôle un jeton dans une colonne de la grille. Le premier à aligner un certain nombre de jetons (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 qui permet de jouer à Puissance 4 en affichant la grille et en interagissant via des entrées utilisateur.

**1. Modélisation de la grille**

La grille est représentée par une structure de données HashMap, où :

* **Clé** : un objet Coordonnees représentant une position avec une ligne et une colonne.
* **Valeur** : une instance de la classe Case indiquant l’état de la case (vide, jaune ou rouge).

Bien qu'une implémentation sous forme de matrice ait été possible, l'utilisation d'un HashMap était imposée dans les consignes du projet. Grâce à cette approche, il est possible de retrouver rapidement une case à l'aide de ses coordonnées. La classe Coordonnees redéfinit d’ailleurs les méthodes hashCode et equals pour garantir une bonne gestion des clés dans le dictionnaire.

**2. Types de cases**

Chaque case de la grille peut être de trois types :

* **Vide** : aucune pièce n’a été placée.
* **Jaune** : représente un jeton du joueur 1.
* **Rouge** : représente un jeton du joueur 2.

Ces types sont implémentés sous forme de classes héritées de la classe mère Case, qui définit les propriétés communes à toutes les cases.

**3. Déroulement d’une partie**

Au début du jeu, les joueurs définissent la largeur et la hauteur de la grille, ainsi que le nombre de jetons à aligner pour remporter la partie.

Le jeu se déroule en alternance entre deux joueurs :

1. Le joueur 1 joue avec les cases jaunes.
2. Le joueur 2 joue avec les cases rouges.

À chaque tour, un joueur choisit une colonne dans laquelle placer son pion. L'ajout de la pièce est effectué en insérant un nouvel élément dans le HashMap sous la forme :  
(coordonnées de la case jouée, instance de CaseJaune ou CaseRouge selon le joueur)

Après chaque coup, le programme vérifie si la partie est terminée en vérifiant deux conditions :

* **La grille est pleine** : aucun autre coup ne peut être joué.
* **Un joueur a aligné le nombre requis de jetons** : un alignement est recherché dans les quatre directions suivantes :
  + **Horizontale**
  + **Verticale**
  + **Diagonale bas-droite**
  + **Diagonale bas-gauche**

L’algorithme scanne la grille en partant du coin supérieur gauche et en parcourant les cases ligne par ligne. Lorsqu’une case contenant un jeton est détectée, il vérifie si elle fait partie d’un alignement gagnant en comptant les jetons adjacents de même couleur.

**4. Fin de partie et redémarrage**

Lorsqu’un joueur remporte la partie ou que la grille est remplie sans gagnant, le programme propose de rejouer. Si les joueurs acceptent, ils peuvent de nouveau configurer la grille et le nombre de jetons à aligner, puis une nouvelle partie démarre.

**III. Présentation des classes**

**1) La classe Coordonnees**

La classe Coordonnees modélise les positions des cases dans la grille à l’aide de deux attributs :

* **i** : représente l’indice de la ligne.
* **j** : représente l’indice de la colonne.

Les coordonnées (0,0) correspondent à la case située en haut à gauche de la grille.

**Utilisation dans la classe Grille**

Dans l’implémentation du jeu, les objets Coordonnees sont utilisés comme clés dans un HashMap pour stocker les cases de la grille. Pour garantir un accès efficace aux éléments de cette structure de données, il a été nécessaire de redéfinir :

1. **La méthode hashCode()** : elle génère un code de hachage unique pour chaque paire (i, j), permettant de regrouper les clés dans des sous-ensembles distincts à l’intérieur du HashMap.
2. **La méthode equals(Object obj)** : elle permet de vérifier si deux objets Coordonnees sont égaux, en comparant leurs attributs i et j.

**Fonctionnement du hashCode()**

Le HashMap fonctionne en organisant ses clés en groupes selon leur code de hachage. Lorsque l’on cherche une valeur à partir d’une clé donnée, l’interpréteur identifie d’abord le sous-ensemble de clés ayant le même hashCode(). Il effectue ensuite des comparaisons avec equals() pour retrouver la clé exacte et accéder à sa valeur associée.

Dans cette implémentation, la méthode hashCode() utilise Objects.hash(i, j), qui génère un code de hachage basé sur les coordonnées (i, j).

**Fonctionnement de equals()**

La méthode equals(Object obj) compare deux objets Coordonnees et retourne true s’ils possèdent les mêmes valeurs pour i et j, sinon false.

Grâce à ces redéfinitions, l'utilisation des objets Coordonnees comme clés dans un HashMap est optimisée et garantit une recherche rapide des cases dans la grille.

**2. Classe Case et ses dérivées**

Classe mère représentant une case de la grille. Les classes dérivées sont :

* CaseVide
* CaseJaune
* CaseRouge

Chaque classe redéfinit la méthode afficher pour représenter graphiquement les cases dans la console (X pour jaune, O pour rouge, espace vide autrement).

**3. Classe Grille**

Gère la structure du jeu :

* Stocke les cases sous forme de HashMap
* Possède des méthodes pour modifier et afficher la grille
* Vérifie si la grille est pleine

**4. Classe Partie**

Cœur du jeu, cette classe gère :

* Le déroulement de la partie
* L'altercation des tours entre joueurs
* La vérification des conditions de victoire
* La détection de l'alignement des pions

**5. Classe Main**

Point d'entrée du programme. Elle :

* Gère l'interaction avec l'utilisateur
* Lance les parties en boucle tant que l'utilisateur le souhaite
* Vérifie les saisies utilisateur

**IV. Difficultés rencontrées**

Plusieurs défis ont été rencontrés lors du développement :

* **Gestion des coordonnées dans le HashMap** : nécessité de redéfinir hashCode et equals
* **Vérification des alignements** : implémentation efficace pour parcourir la grille et détecter un gagnant
* **Interaction utilisateur en console** : vérification et validation des entrées
* **Gestion de l'affichage** : assurer une représentation correcte de la grille

**Conclusion**

Ce projet a été une excellente opportunité pour mettre en pratique les concepts de programmation orientée objet en Java. L'utilisation de structures de données avancées, la gestion des classes et l'interaction en console ont été des points-clés du développement.

L'amélioration possible serait de proposer une interface graphique pour rendre le jeu plus attractif et intuitif.

Introduction

Ce rapport à pour but de présenter le projet que j’ai réalisé dans le cadre du cours de Java, lors de ma 4ème année d’étude à Polytech Clermont.

Ce projet a consisté en l’implémentation du jeu « Puissance 4 », à l’aide de la programmation orientée objet.

Pour ce faire, j’ai modélisé plusieurs classes Java, permettant de modéliser des entités du jeu, telles que la grille du puissance 4, les cases de la grille, les coordonnées des cases, etc.

Nb : Afin de versionner ce projet, de le partager, ainsi que d’obtenir une sauvegarder Cloud de ce dernier, j’ai créé un repository GitHub pour ce programme : <https://github.com/Nathan7128/projet-puissance4>.

Annonce du plan

Dans un premier temps, je vais présenter brièvement le principe du jeu « Puissance 4 ».

Dans une seconde partie, j’expliquerai le principe de fonctionnement du programme, en expliquant la logique utilisée pour modéliser le jeu à l’aide de la POO.

Ensuite, je présenterai plus en détail les différentes classes Java du programme, en donnant des précisions sur leurs attributs et méthodes.

Enfin, j’expliquerai les difficultés que j’ai pu rencontrer durant l’implémentation de ce code.

1. Présentation du jeu

Le puissance 4 est un jeu composé d’une grille, qui nécessite 2 joueurs. Chaque joueur peut glisser un « jeton » dans une colonne, et gagne lorsqu’il a aligné le nombre de jetons requis pour gagner.

L’alignement peut être horizontal, vertical ou diagonal.

Dans le jeu que j’ai codé, il est possible de choisir la largeur de la grille (le nombre de colonnes), la hauteur de la grille (le nombre de lignes), ainsi que le nombre de pions à aligner pour gagner la partie.

1. Fonctionnement du programme

Ce programme est une application console, c’est-à-dire que lors de son exécution, le jeu « Puissance 4 » est affiché dans une console.

Le puissance 4 est modélisé par une grille (cf. image 1) (avec une largeur et une hauteur), qui est implémentée par un « HashMap » Java.

Ce HashMap est en réalité un dictionnaire clé/valeur, qui possède comme clé des objets de type Coordonnées (qui ont comme attributs une ligne et une colonne), et comme valeur des cases de la grille.

Une autre solution pour implémentée la grille aurait été de l’implémentée à l’aide d’une matrice de case (ou chaque élément de la matrice correspond à une case de la grille), cependant, il nous a été demandé d’utiliser ce HashMap.

Il est donc possible de retrouver une case de la grille grâce à un objet de type Coordonnées. Chacune de ces coordonnées possédant un « Hash code », ainsi qu’une méthode permettant de déterminé si 2 coordonnées sont égales.

J’expliquerais plus en détail dans la section sur la classe Coordonnées le fonctionnement de cet hash code et de la manière de comparer 2 objets de type Coordonnées.

Pour les cases, elles peuvent être de 3 types : vides, jaunes ou rouges.

Les cases vides représentant des cases dans lesquelles les joueurs n’ont pas placé de pion, les cases jaunes représentent les cases dans lesquelles le joueur numéro 1 à jouer, et les cases rouges celles du joueur numéro 2.

Ces 3 types sont implémentés comme des classes héritées de la classe mère « Case », qui permet de définir les points communs de ses classes filles.

En ce qui concerne la partie, il est permis aux joueurs de choisir la largeur et la hauteur de la grille, ainsi que le nombre de pions à aligner pour gagner.

La partie est jouée par 2 joueurs, qui sont identifiés comme « Joueur 1 » et « Joueur 2 ».

Le joueur 1 joue avec les cases jaunes, et le joueur 2 avec les cases rouges.

Tant que la partie n’est pas terminée (tant qu’aucun joueur n’a gagné la partie et que la grille n’est pas pleine), chaque joueur joue à son tour un pion dans la grille.

A chaque coup d’un joueur, on ajoute au HashMap le couple (« coordonnées de la case jouée », « objet de type case rouge/jaune » (en fonction du joueur qui a effectué le coup)).

Pour déterminer si une partie est finie, on regarde après chaque coup d’un joueur si la grille est pleine ou si le joueur qui a joué à aligner le nombre de pions requis pour gagner. Pour vérifier cette deuxième condition, on regarde pour chaque case (de la couleur correspondant au joueur qui a joué), 4 alignements possibles :

* La ligne droite : le nombre de cases (de la bonne couleur) alignées horizontalement avec la case qu’on regarde.
* L’alignement bas : le nombre de cases (de la bonne couleur) alignées verticalement avec la case qu’on regarde.
* La diagonale en bas à droite : le nombre de cases (de la bonne couleur) alignées dans la diagonale en bas à droite de la case qu’on regarde.
* La diagonale en bas à gauche : le nombre de cases (de la bonne couleur) alignées dans la diagonale en bas à gauche de la case qu’on regarde.

En itérant ces différentes cases, en commençant par celle en haut à gauche de la grille et en se déplaçant de gauche à droite puis de bas en haut, il est possible de vérifier l’ensemble des alignements possibles pour l’ensemble des cases.

Lorsque la partie est finie, les joueurs choisissent ou non de recommencer une nouvelle partie.

Dans le cas ou ils en recommencent une, il est donc une nouvelle fois demandé à l’utilisateur la largeur et la hauteur de la grille, ainsi que le nombre de pions à aligner.

Puis, une nouvelle partie est instanciée, avec les choix effectués par l’utilisateur.

1. Présentation des classes
2. La classe Coordonnées

Comme son nom l’indique, cette classe permet de modéliser les coordonnées des cases de la grille.

Ces coordonnées sont exprimées en ligne (attribut « i » de la classe) et en colonne (attribut « j »).

Les coordonnées (i = 0 , j = 0) correspondent à la case située en haut à gauche de la grille.

Comme expliqué dans l’explication du programme, ces objets de type Cordoonées sont utilisés comme clé d’un HashMap dans la classe Grille. De ce fait, il a fallu redéfinir la méthode permettant d’obtenir un hash code d’un objet de type Coordonnées, ainsi que la méthode permettant de savoir si deux coordonnées sont égales.

En effet, le HashMap regroupe toutes ses clés en sous-ensemble, ou chaque sous-ensemble correspond à un unique hash code (chaque élément d’un même sous-ensemble possède le même hash code).

Pour pouvoir retrouver la valeur associée à une clé d’un HashMap, l’interpreteur va alors effectuer des tests d’égalité entre chaque élément (clé) du sous-ensemble possédant le même hash code que la clé pour laquelle on souhaite retrouver la valeur.

Le hash code implémenté utilise la méthode « hash » de la classe « Objects », et permet de renvoyer le hash code de n’importe quel Objet. En l’occurrence, elle renvoie le hash code du couple (i, j) (coordonnées).

La méthode « equals », permettant de comparer 2 objets Coordonnées, renvoie true si les 2 coordonnées ont la même ligne et la même colonne, et false sinon.

1. La classe mère « Case » et ses classes dérivées

Cette classe est tout en haut de la hiérarchie des classes permettant de modéliser les cases de la grille. Elle implémente les attributs que se partagent toutes ses classes dérivées. En particulier, chaque case possède un attribut de type Coordonnées « p », représentant les coordonnées de la case dans la grille.

Ses classes dérivées sont les classes « Case vide », « Case jaune » et « Case rouge ».

Les cases vides représentent des cases dans lesquelles aucun joueur n’a encore joué, les cases jaunes sont celles ou le joueur 1 a joué, et les rouges sont celles attribuées au joueur 2.

De plus, cette classe inclue la méthode abstraite « afficher », qui devra être redéfinie par toutes ses classes dérivées, car l’affichage d’une case dépend du type de la case (vide, jaune ou rouge).

Les cases vides sont modélisées par du vide «  », les jaunes par un « X » et les rouges par un « O ».

1. La classe grille

Cette classe permet donc de modéliser la grille du jeu.

Elle sert donc à stocker les différentes cases de la grille, qui sont chacune associées à un objet de type Coordonnées dans l’attribut de type HashMap « cases ».

Chaque grille possède également une largeur et une hauteur.

Lors de la construction d’un objet Grille, le constructeur la remplie de case vide, c’est-à-dire instancie pour chaque case de la grille un objet Coordonnées auquel il attribut donc une case dans le HashMap.

J’ai aussi implémenté la méthode afficher, qui permet de renvoyer une chaine de caractères qui permettra d’afficher la grille dans la console (cf. image1), au format souhaité par notre professeur.

A chaque fois ou 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 permet aussi de définir une case de la grille et obtenir une case en fonction de coordonnées à l’aide des méthode « setCase » et « getCase ».

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.

1. La classe partie

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

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

Lors de son instanciation, on indique en lui passant en paramètre les spécificités de la partie : la largeur de la grille, sa hauteur et le nombre de pions à aligner « nbPionsAAligner » pour gagner.

Le constructeur instanciera alors une grille avec la taille indiquée par l’utilisateur, et cette grille sera 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 que chaque joueur a effectué un coup, on regarde si la partie est finie, 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, alors on regardera s’il y’a une combinaison « nbPionsAAligner » de cases jaunes alignées, dans n’importe quel alignement possible.

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 parcours 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 ».

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

Cette classe permet également à un joueur d’effectuer un coup, grâce à la méthode « jouer », qui va placer la case dans la colonne saisie au clavier par le joueur, à la ligne la plus basse possible (la case vide au dessus de la case non vide la plus haute de la colonne, ou à la ligne la plus basse de la colonne si cette dernière n’est composée que de cases vides).

Enfin, afin de savoir si un coup est possible (si la colonne saisie par l’utilisateur est remplie de cases non vides ou non), j’ai implémenté une méthode « coupPossible », qui renvoie true si la ligne la plus haute de la colonne est une instance de la classe Case vide.

1. Déroulement de la partie : 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 donc tant que « jouer » vaut true, et cette condition est implémentée à l’aide d’une boucle « while ».

Au début de chaque partie, il est demandé à l’utilisateur de saisir la largeur et la hauteur de la grille, ainsi que le nombre de pions à aligner pour gagner. On vérifie également que la hauteur et la largeur saisies sont bien supérieures à 0, sinon on demande à l’utilisateur de ressaisir la largeur et/ou la hauteur. De plus, on vérifie que le nombre de pions à aligner soit bien inférieur ou égal au minimum entre les valeurs de la largeur et la hauteur de la grille, afin de savoir s’il est bien possible d’aligner le nombre de pions saisi par l’utilisateur.

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. Nous demandons au joueur (auquel c’est le tour) de saisir une colonne.
2. Nous vérifions si le coup est possible à l’aide de la méthode « coupPossible » de la classe Partie. Si le coup n’est pas possible, retour à l’étape 1)
3. On joue le coup dans la grille à l’aide de la méthode « jouer »
4. On regarde si la partie est finie
5. Si elle n’est pas finie, on change le joueur de la partie à l’aide de la méthode « changerJoueur » de la classe Partie.
6. Enfin, on affiche la grille dans la console

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.