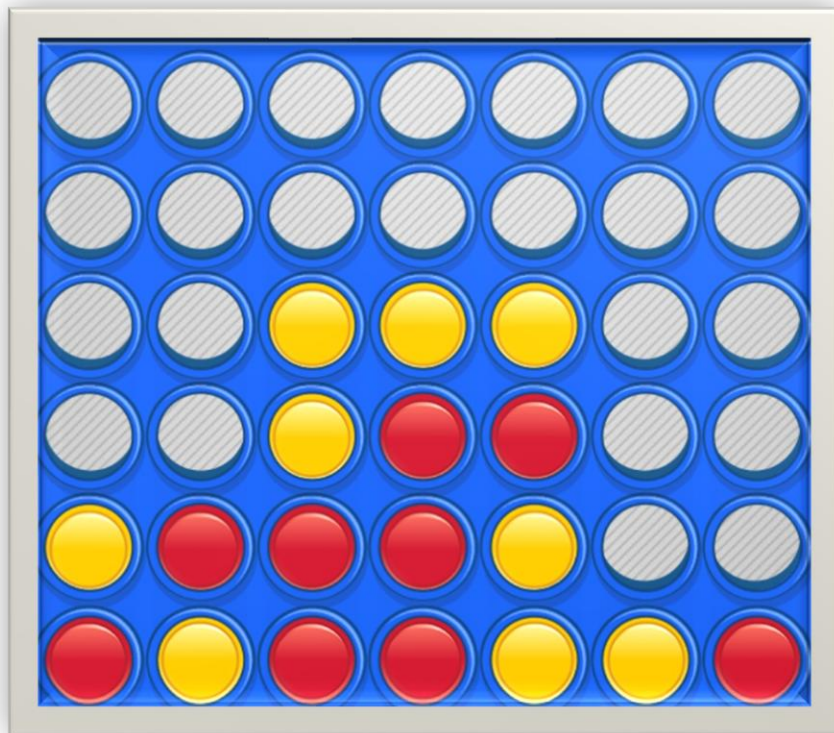




Rapport du TP : Puissance 4



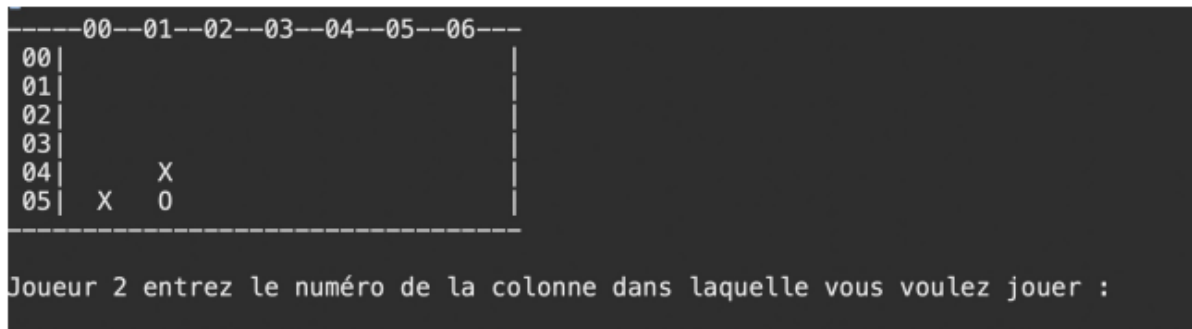
Joseph AKUETE & Nicolas NOGARET 4A IMDS
19/02/2025

Table des matières

Introduction	2
Diagramme de classes.....	3
Modifications du code apportées par rapport au diagramme donné en cours	4
Difficultés rencontrées.....	7
Fonctionnement du Jeu	8
Conclusion	9

Introduction

Dans le cadre de ce TP noté en Java, nous avons réalisé un projet consistant à coder le jeu classique du Puissance 4 en mode console (c'est-à-dire sans utiliser une interface graphique mais on passe directement par le terminal) comme sur l'image ci-dessous :



```
-----00--01--02--03--04--05--06-----
00|                                     |
01|                                     |
02|                                     |
03|                                     |
04|                                     X |
05|  X   O                             |
-----
Joueur 2 entrez le numéro de la colonne dans laquelle vous voulez jouer :
```

Le jeu se joue entre deux joueurs qui alternent pour placer des pions dans une grille. Le but est d'aligner horizontalement, verticalement ou en diagonale un nombre défini de pions.

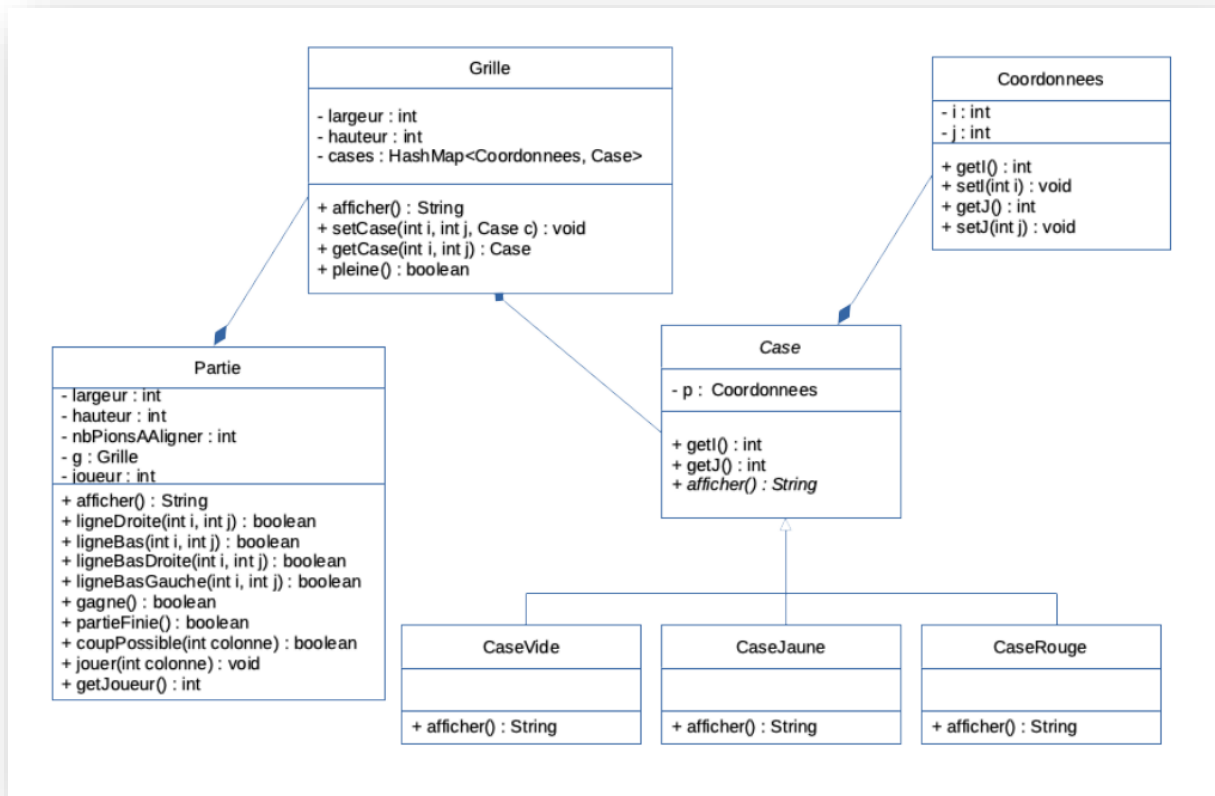
L'objectif principal de ce projet est de nous familiariser avec ce langage et de mettre en pratique les différentes notions de programmation en orientée objet (POO) que l'on a pu apprendre.

Un diagramme de classes nous a également été fourni pour nous guider sur les différentes classes nécessaires ainsi que sur les fonctions potentielles à implémenter pour assurer le bon déroulement du jeu.

Pour structurer ce rapport, nous commencerons par présenter le diagramme de classes qui nous a été fourni afin d'avoir une vue d'ensemble sur l'architecture initiale du projet. Ensuite, nous détaillerons les modifications que nous avons apportées par rapport aux consignes de départ. Nous aborderons également les difficultés rencontrées lors de l'implémentation de notre jeu, avant d'expliquer en détail son mode de fonctionnement. Enfin, nous conclurons par un bilan du projet et proposerons des pistes d'amélioration pour enrichir l'expérience de jeu.

Diagramme de classes

Le diagramme de classe qui nous a été fourni initialement est structuré autour de 4 classes principaux comme le montre l'image ci-dessous :



Chaque classe joue un rôle spécifique dans la gestion de notre jeu :

Classe Grille : Elle permet de définir le plateau de jeu du Puissance 4 en gérant sa structure, son affichage dans la console ainsi que l'état des différentes cases qui le composent.

Classe Coordonnées : Elle stocke la position (coordonnées) d'une case dans la grille, permettant ainsi d'identifier chaque case de manière unique. Elle est utilisée comme clé dans la HashMap de la classe Grille.

Classe Case : Elle représente l'état des cases en stockant leurs coordonnées. Elle possède trois classes filles : **CaseVide**, **CaseJaune** et **CaseRouge**, qui sont responsables de l'affichage. Une case affiche un « X » si le joueur 1 a joué (**CaseJaune**), un « O » si c'est le joueur 2 (**CaseRouge**), et reste vide si elle n'a pas encore été jouée (**CaseVide**).

Classe Partie : Elle permet de générer une instance du jeu et contrôle le déroulement de la partie. Elle gère l'alternance des joueurs, vérifie si un coup est possible dans une colonne

lorsqu'un joueur tente de jouer, et détermine l'état du jeu, notamment si la partie est terminée.

Néanmoins on avait la liberté de faire quelques modifications dans ce diagramme si jamais on le désirait. Dans la suite, nous allons présenter les différentes modifications que nous avons apportées au diagramme qui nous avait été donné initialement.

Modifications du code apportées par rapport au diagramme donné en cours

Fonction `gagneLigne` dans la classe `Partie` :

Dans le diagramme initial, quatre fonctions booléennes étaient utilisées pour détecter si un coup était gagnant. Nous avons choisi de les regrouper en une seule fonction, que nous avons appelée `gagneLigne`.

Cette fonction permet de déterminer si un coup est gagnant ou non. Elle prend en paramètres les coordonnées (i, j) du dernier pion joué et vérifie s'il existe un alignement horizontal, vertical ou diagonal du nombre de pions requis. Si un tel alignement est détecté, elle renvoie `true`, indiquant que le joueur a gagné. Sinon, elle renvoie `false`.

Comme mentionné en introduction, nous avons préféré utiliser une seule fonction pour vérifier tous les types d'alignements plutôt que de créer plusieurs fonctions distinctes (`ligneDroite`, `ligneBas`, `ligneBasDroite`, `ligneBasGauche`). Cette approche nous a permis de nous concentrer sur une seule fonction plus intuitive et centralisée, plutôt que de gérer plusieurs petites fonctions indépendantes.

Explication du fonctionnement de la fonction :

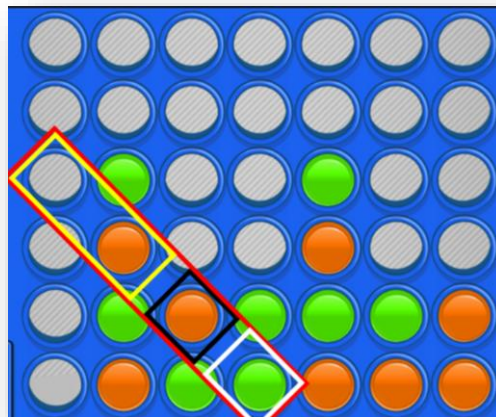
- La fonction utilise une variable `cmt` (compteur) qui sert à compter le nombre de pions alignés du joueur dans chaque direction. Quatre vérifications sont effectuées, chacune suivant une direction spécifique.
- Cette fonction réalise 4 vérifications, elles désignent toutes un sens différent (vertical, horizontal et les 2 diagonales) et le code de chacune d'entre elle est sensiblement identique à l'exception des indices qui changent selon les différentes directions observées. De plus la vérification verticale demande de vérifier une seule direction (vers le bas car il ne peut pas avoir de jeton au-dessus du dernier coup joué) tandis que les autres demandent d'en vérifier 2 (par exemple pour le sens horizontal il faut vérifier à gauche et à droite du dernier coup joué).

De ce fait, nous allons expliquer uniquement le fonctionnement de la vérification diagonale (Haut Gauche -> Bas Droit) étant donné les similitudes entre les vérifications.

Vérification diagonale (Haut Gauche -> Bas Droite) :

A partir du dernier pion joué la fonction doit explorer 2 directions à savoir la diagonale en haut à gauche du pion et celle en bas à droite du pion.

Par exemple dans le cas ci-joint (le plateau a été rétréci pour être plus clair sur l'image) un pion vient d'être joué en (2,3) (dans le cadre noir). Pour vérifier si la diagonale Haut Gauche -> Bas Droite (en rouge) est gagnant il faut donc regarder en haut à gauche (cadre jaune) et en bas à droite (cadre blanc).



Afin de vérifier en haut à gauche, il va falloir faire progressivement incrémenter les lignes (car on monte dans la grille) et diminuer les colonnes (car on va sur la gauche de la grille) tant que l'on trouve des jetons appartenant au joueur qui vient de jouer.

Une fois que l'on a fini de compter le nombre de jeton en haut à gauche, on examine en bas à droite, pour cela on refait varier l'indice des lignes et des colonnes mais ce coup-ci on réduit les lignes (car on descend) et on augmente les colonnes (car on va vers la droite).

Ensuite on compare le nombre de jetons consécutifs trouvés et si on a la condition `cmt > NbPionsAAaligner` qui est valide alors on est dans un cas où le joueur a gagné et donc on renvoie `True`, sinon cela signifie qu'on n'a toujours pas trouvé de victoire et donc on vérifie l'autre diagonale.

Maintenant que nous venons de voir comment fonctionne une vérification nous allons examiner le fonctionnement de la fonction de dans son ensemble grâce à un exemple précis avec une victoire sur une diagonale (haut gauche -> bas droite).

Explication sur un exemple :

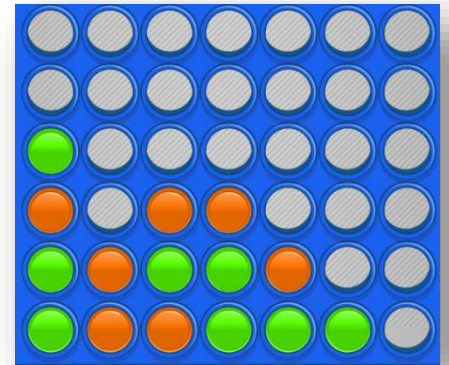
On se place dans le cadre de la partie ci-jointe et il s'agit au joueur vert de jouer. On voit dans que s'il joue dans la colonne 2, il va gagner la partie.

Voyant comment fonctionne notre fonction dans le cas où le joueur vert fait le coup (3,2).

Dans un premier temps il va faire la vérification verticale, néanmoins comme on aura que 1 pion aligné la condition `cmt >= NbPionsAAaligner` ne sera pas validée et donc on passera à la suite.

Maintenant on réinitialise `cmt` à 0 et on passe à la vérification horizontale, le code va donc chercher combien de jetons vert consécutif se trouvent à gauche (ici 0 donc `cmt=1` car on compte le point joué) puis à droite (0 donc `cmt` reste à 1). La condition de boucle `cmt >= NbPionsAAaligner` est donc invalide. On va donc étudier si le coup est gagnant grâce à la diagonale allant de en haut à gauche vers en bas à droite.

On réinitialise à nouveau `cmt` à 0 et on regarde donc en haut à gauche (on incrémente la ligne et on décrémente la colonne), on trouve alors 1 jeton vert donc on obtient `cmt = 2` (car on prend en compte le pion joué). Maintenant on vérifie en bas à droite combien de jetons verts consécutifs on trouve, on en trouve donc 2 donc `cmt` passe à 4 ($2+2$). La condition de boucle de la fonction est donc validée car on a `cmt = 4 = NbPionsAAaligner`, la fonction renvoie donc `True` (ce qui est effectivement ce qui est à faire car on a détecté une victoire du joueur vert sur cette diagonale-là).



Ajout de l'attribut `m_type` dans la classe `Case` :

Dans la conception initiale, il est possible d'identifier le type de chaque case (`CaseJaune`, `CaseRouge`, `CaseVide`) en utilisant `instanceof`. Cependant, afin de rendre notre code plus clair, notamment dans la fonction `gagneLigne`, nous avons décidé d'ajouter l'attribut `int m_type` dans la classe `case`.

Cet attribut prend les valeurs suivantes :

- 0 si la case est vide,
- 1 si la case est une `CaseJaune`,
- 2 si la case est une `CaseRouge`.

Cette modification nous permet de comparer plus facilement si une case correspond au joueur qui a joué dedans, puisque l'attribut `joueur` de la classe `Partie` est également un entier en améliorant la lisibilité du code.

Ajout de la méthode `changerJoueur()` dans la classe `Partie` :

Cette méthode permet d'alterner le tour des joueurs en modifiant la valeur de l'attribut `joueur` après chaque coup joué.

- Si le joueur actuel est le joueur 1, alors il devient joueur 2.
- Sinon, si c'est le joueur 2, alors il redevient joueur 1.

Elle est utilisée après chaque coup valide pour passer la main au joueur suivant, assurant ainsi le bon déroulement du jeu et une alternance correcte entre les joueurs.

Ajout de `getter` dans la classe `Partie` :

Nous avons ajouté quatre nouvelles méthodes dans la classe `Partie` :

- `getNbPionsAAaligner` : récupère le nombre de pions qu'il faut aligner pour gagner la partie.
- `getHauteur` : récupère la hauteur du plateau.
- `getLargeur` : récupère la largeur du plateau.
- `getJoueur` : récupère le joueur actuel.

Ces méthodes permettent d'accéder aux valeurs de certains attributs privés de la classe. Ces attributs étant déclarés comme privés, il était impossible de récupérer leur valeur directement dans le main.

Grâce à ces `getter`, nous pouvons notamment personnaliser les messages envoyés dans nos mails en incluant des informations sur la configuration du jeu, comme les dimensions de la grille, le nombre de pions à aligner et le joueur en cours.

Difficultés rencontrées

La création du jeu ne nous a pas posé de grandes difficultés, car nous avons déjà des bases en programmation orientée objet, notamment grâce aux travaux pratiques précédents, comme la réalisation du jeu du labyrinthe en C++. Cependant, nous avons dû adapter nos connaissances du C++ au Java afin d'assimiler les particularités de ce langage.

Le principal défi de ce projet a été l'implémentation de la fonction `gagneLigne` dans la classe `Partie`. Il a fallu concevoir une méthode efficace pour détecter les alignements gagnants dans toutes les directions (horizontale, verticale et diagonales). Pour simplifier cette vérification, nous avons ajouté un attribut `m_type` à la classe `Case`, ce qui nous a permis d'identifier plus facilement les cases appartenant au joueur en cours. Cette approche a rendu la détection des victoires plus fiable et plus intuitive à nos yeux.

Fonctionnement du Jeu

Nous allons maintenant expliquer le fonctionnement de notre programme principal (main).

Notre main permet de gérer le déroulement d'une partie de Puissance 4 en interagissant avec l'utilisateur. Lors de son exécution, il commence par demander au joueur s'il souhaite utiliser la configuration par défaut ou s'il préfère personnaliser les dimensions de la grille ainsi que le nombre de pions à aligner pour gagner.

- **Si le joueur choisit la configuration par défaut**, une grille de 7 colonnes sur 6 lignes est créée, avec une condition de victoire fixée à 4 pions alignés.

Exemple d'affichage de notre grille par défaut :

```
Vous avez choisi la grille par défaut (7x6) avec 4 pions à aligner.
La partie commence avec une grille de 7x6 et 4 pions à aligner !
  0  1  2  3  4  5  6
-----
0 |  |  |  |  |  |  |
-----
1 |  |  |  |  |  |  |
-----
2 |  |  |  |  |  |  |
-----
3 |  |  |  |  |  |  |
-----
4 |  |  |  |  |  |  |
-----
5 |  |  |  |  |  |  |
-----
Tour du joueur 1 : Saisir la colonne dans laquelle jouer (0 à 6) :
```

- **Si le joueur préfère une configuration personnalisée**, il doit saisir la largeur et la hauteur de la grille, ainsi que le nombre de pions nécessaires pour gagner. Une vérification est effectuée pour s'assurer que cette valeur est bien inférieure aux dimensions de la grille, afin de garantir un jeu équilibré.

Exemple de grille avec des dimensions personnalisés :

```

La partie commence avec une grille de 8x7 et 5 pions à aligner !
  0  1  2  3  4  5  6  7
-----
0 |  |  |  |  |  |  |  |
-----
1 |  |  |  |  |  |  |  |
-----
2 |  |  |  |  |  |  |  |
-----
3 |  |  |  |  |  |  |  |
-----
4 |  |  |  |  |  |  |  |
-----
5 |  |  |  |  |  |  |  |
-----
6 |  |  |  |  |  |  |  |
-----
Tour du joueur 1 : Saisir la colonne dans laquelle jouer (0 à 7) :

```

Une fois la configuration établie, le programme affiche un message récapitulatif et la partie peut commencer. La gestion du jeu repose ensuite sur une boucle principale qui permet aux joueurs de déposer tour à tour un pion dans la colonne de leur choix, tout en vérifiant après chaque coup si l'un des joueurs a remporté la partie ou si la grille est pleine, entraînant alors un match nul.

Conclusion

Ce projet en Java fut très enrichissant car il nous a permis de mettre en pratique nos connaissances en programmation orientée objet en développant un jeu de Puissance 4 fonctionnel en mode console.

On remarque cependant qu'il est possible de poursuivre ce projet en y ajoutant différentes fonctionnalités pour enrichir l'expérience de jeu. Par exemple, on pourrait améliorer l'affichage en passant à une interface graphique offrant un rendu plus esthétique, avec des boutons interactifs et une grille colorée. Il serait également intéressant de proposer une option pour jouer contre un adversaire contrôlé par l'ordinateur. Celui-ci pourrait suivre des règles simples pour réagir aux actions du joueur, comme bloquer un alignement ou saisir une opportunité de victoire immédiate. Toutefois, développer une logique plus avancée, capable d'analyser plusieurs coups d'avance (que ce soit pour défendre ou pour essayer de gagner) et de mettre en place des stratégies efficaces, serait une tâche bien plus complexe et nécessiterait beaucoup de temps pour atteindre un niveau de défi satisfaisant.