Rapport Jeu de points (Gomoku - PuissanceK)

Nous avons 7 classes dans notre projet dont une est la classe principale :

La classe Case :

Cette classe représente une case dans le plateau, elle a comme champs : **etat** (la valeur contenue, soit ' ' pour une case vide, soit 'X' pour le joueur, soit 'O' pour l'ordinateur), et le **score** qui correspond à l'évaluation de la case. Comme elle a un tableau de **kuplets** qui représente les kuplets concernés par cette case (initialement un tableau vide). Elle comprend également des méthodes setters et getters, une méthode addKuplet() pour ajouter un kuplet au tableau des kuplets (cette méthode ne sera utiliser que lors de l'initialisation des cases de plateau), une méthode toString() pour afficher la valeur de la case, et une méthode evaluate() qui renvoie l'évaluation de la case qui va aider l'ordinateur à faire son choix de case lors son tour. Pour calculer l'évaluation d'une case, on prend l'évaluation du premier kuplet de sa liste de kuplets, puis on parcourt le tableau, on compare la valeur prise avec les autres évaluations des kuplets en prenant toujours la valeur la plus grande (comme si on cherche le maximum dans un tableau), puis on aura l'évaluation de la case qui présente le maximum des évaluations des kuplets concernés par cette case.

La classe XY :

Elle ressemble de la classe **Case** mais elle est conçue spécialement pour sauvegarder les cordonnées d'une case (**x** et **y**), comme elle comprend le **score** de la case (qui correspond à son évaluation).

La classe Kuplet :

Cette classe représente une séquence de K cases adjacentes (voisine horizontalement, verticalement ou diagonalement) tel que K est le nombre de cases alignées nécessaire pour gagner la partie (le K est un nombre qui sera définit par l'utilisateur). Elle comprend un champ **K** dont on a parlé, un champ **score** qui correspond à l'évaluation du kuplet, et un tableau **Case[]** qui doit contenir l'ensemble des cases qui constituent ce kuplet. Son constructeur prend en paramètre la futur valeur de **K**, crée un nombre de cases vides égal à K et initialise le score à 0. Elle comprend des setters et des getters, une méthode evaluateX() qui renvoie une évaluation du kuplet en fonction de nombre de 'X' contenus dans ses cases (0 pour aucun 'X', 2 pour un seul 'X', 4 pour deux 'X', …, n*2 pour n 'X'), une méthode evaluateO() qui renvoie une évaluation du kuplet en fonction de nombre de 'O' contenus dans ses cases (1 pour aucun 'O', 3 pour un seul 'O', 5 pour deux 'O', …, (n*2)+1 pour n 'O', ici on a donné un peu plus de priorité au nombre de 'O' par rapport au nombre 'X' , si par exemple on a un kuplet avec 2 'X' et un kuplet avec 2 'O', l'ordinateur doit choisir de jouer sur une case de celui qui contient 2 'O' (pour avoir la chance de gagner) que de jouer sur celui qui contient 2 'X' (pour empêcher le joueur à aligner ses cases)), et une méthode evaluate () qui renvoie l'évaluation finale du kuplet en fonction des deux évaluations précédente : si pas de 'X' et pas de 'O' alors le kuplet est vide et son score aura donc 1 (et non pas 0 pour différencier avec le kuplet qui contient au moins un 'X' et un 'O' qui aura le score 0, car un kuplet vide est avantageux qu'un kuplet avec au moins un 'X' et un 'O' qui mènera jamais à une victoire). Si pas de 'X' mais il y des 'O' le score sera la valeur renvoyée par evaluateO(). Si pas de 'O' mais il y des 'X' le score sera la valeur renvoyée par evaluateX(). Si il y a au moins un 'X' et un 'O' le score prendra la valeur 0.

La classe Plateau :

Elle représente le plateau du jeu, elle contient un tableau de **cases** de deux dimensions, un champ **longueur** qui représente la longueur du plateau souhaitée par l'utilisateur, et un champ **largeur** qui représente la largeur du plateau souhaitée par l'utilisateur. Son **constructeur** prend en paramètres la longueur et la largeur données par l'utilisateur afin de créer le tableau de deux dimensions et initialise des cases du plateau (crée des cases vides). Elle comprend également une méthode **toString**() qui affiche le plateau avec les cases qu'il contient sous un format plus lisible, par exemple :

[X][O][ ]
[X][ ][ ][ ]
[ ][ ][ ][ ]

La classe Jeu :

C'est la classe qui présente une partie du jeu Gomoku, elle contient les méthodes importantes du programme.
Un objet Jeu est caractérisé par 4 champs : un objet de la classe **Plateau**, la **longueur** et la **largeur** du plateau, et le nombre **K** qui correspond au nombre nécessaire de cases alignées pour gagner la partie. Il contient également un champ appelé **aquiletour** qui prend la valeur True si c'est le tour du joueur, la valeur False si c'est le tour de l'ordinateur. Son **constructeur** initialise la longueur, la largeur, le K, comme il crée un objet Plateau avec la longueur et la largeur données. Puis il fait appel à la méthode **initialize**() qui à son tour, initialise toutes les cases du plateau en créant tous les kuplets (horizontales, verticales et diagonales), et en associant chaque case aux kuplets correspondants.
La méthode **tourdejeu**() c'est la méthode qui permet de remplir par un 'X' la case choisie par l'utilisateur (si elle est vide). Tandis que la méthode **tourOrdinateur**() permet de remplir par un 'O' la case choisie (par intelligence artificielle) par l'ordinateur.
La méthode **promptXY**() permet de demander à l'utilisateur d'introduire les cordonnées x et y (ligne et colonne) de la case qui compte choisir pour son tour de jeu.
La méthode **getMaxEvals**() choisi et renvoie parmi les cases vides du plateau, l'ensemble des cases qui ont un score (évaluation) égale à l'évaluation maximale du plateau. Elle permet à l'ordinateur de choisir la case à jouer parmi des avec des scores élevés. Tandis que la méthode **getBestEvals**() prend ce résultat en paramètre, puis élimine les cases qui contiennent des kuplets moins avantageux (kuplet avec un score = 0). Son but est d'améliorer bcp plus le choix de l'ordinateur.
La méthode **getHighEval**() renvoie le score maximum parmi les scores de tous les kuplets du plateau (utile pour savoir s'il y a un gagnant, elle renvoie donc une valeur égale à k*2 si le joueur gagne, renvoie k*2+1 si l'ordinateur gagne, renvoie une valeur moins de k*2 si pas de gagnant encore).
La méthode **pasDeCaseVide**() parcourt toutes les cases du plateau et teste si toutes les cases sont remplies (soit par 'X' ou par 'O'). Si oui elle renvoie True.

La classe puissanceK :

C'est une classe dérivée de la classe **Jeu,** la différence entre les deux classes réside dans la méthode promptXY() qui calcule automatiquement -dans cette classe- le numéro de ligne de la case à jouer par le joueur, le joueur ne doit fournir dans ce cas que le numéro de colonne. Et la méthode **getMaxEvals**() qui doit limiter son parcours que sur les dernières cases vides du plateau. cette classe comprend deux autres méthodes (estDernierVide() qui teste si la case correspond aux cordonnées passées en paramètre est la dernière case dans sa colonne ou non, et dernCaseVide() qui renvoie les cordonnées de la dernière case vide d'une colonne donnée en paramètre) qui sont utilisées par les deux méthodes précédentes afin d'organiser le code source.

La classe Driver :

C'est la classe principale qui contient la méthode main(). Elle permet à l'utilisateur de saisir la longueur et la largeur souhaitées pour le plateau du jeu, et de choisir le type du jeu (Gomoku ou PuisanceK). Un objet de classe Jeu (ou PuissanceK selon le choix de l'utilisateur) est créé avec les paramètres fournis par l'utilisateur. A chaque tour du jeu : si le tour est au joueur, le programme appelle la méthode promptXY() afin de saisir le choix de l'utilisateur, il appelle ensuite la méthode tourdejeu() avec les cordonnées fournies par l'utilisateur. Sinon (c'est le tour de l'ordinateur), le programme fait une évaluation des cases du plateau et choisi aléatoirement une case vide parmi les cases les plus avantageux (le résultat de la méthode getBestEvals()) puis appelle la méthode tourOrdinateur() en fonction de la case choisie. Après chaque tour de jeu, le programme teste s'il y a un gagnant (le résultat de la méthode getHighEval()) sinon il teste si toutes les cases sont remplies (le résultat de la méthode pasDeCaseVide()) . si c'est le cas ou l'autre : le programme met fin au jeu et affiche le résultat (le gagnant pour le 1er cas ou match nul pour le 2ème cas) .