Rapport projet Gomoku

Le choix des packages :

On a choisi de segmenter notre jeu en deux paquets : l'un gère plutôt l'affichage et le fonctionnement du jeu. Tandis que l'autre gère le plateau, les cases, les positions, etc…

Ce choix nous paraît évident car la séparation de ces derniers nous paraît bien plus cohérents, de plus nous n'avons pas décider découper plus le programme car nous trouvions que les classes respectives de chaque paquet allaient bien ensemble.

Le package Game :

Classe Game : Cette classe gère le déroulement du jeu (menu, joueur, fin de partie, etc…). On a choisi cette classe car elle est composée du main et ce dernier gère tout le temps le déroulement du jeu. Elle appelle la fonction match pour gérer la partie.

Classe Match : Elle est composée des deux joueurs et du plateau. Cette classe gère les tours de jeu de chaque joueur, leurs actions possibles ainsi que la vérification de l'alignement des pions. On a choisi de faire cette classe pour gérer tous les fonctionnements

internes relatifs au jeu sans « polluer » la fonction main (pose de pions, vérifier l'alignement des pions, …).

Classe abstraite Player : C'est la base des deux modèles de joueur. On a choisi de faire cette classe pour éviter de dupliquer du code entre les deux types de joueur. C'est une classe abstraite et non une interface car elle comporte des attributs et du code en commun.

Classe HumanPlayer : Elle hérite de la classe Player et permet de créer un nouveau joueur avec un nom. Elle implémente la fonction selectedMove.

Classe RobotPlayer : Elle hérite de la classe Player et permet de créer un IA avec un nom prédéfini et commun à toutes les IA. Elle implémente aussi la fonction selectedMove.

Classe GamePlayerLeaves : Cette classe étend Exception. Elle permet de gérer quand un joueur quitte le jeu. On a choisi de faire cette exception pour faciliter et rendre plus agréable notre code.

Le package Board :

Classe Board : Classe qui gère le plateau, son affichage, sa taille, le changement de couleur d'une case. Cette classe a été faite pour s'occuper du plateau qui est un tableau de tableau de cases (classe Tile).

Classe Tile : Ce sont les cases du plateau, elle contienne une couleur qui les attribue un affichage. Cette classe nous parait obligatoire pour gérer une à une les cases du plateau. Faire un tableau de tableau de cases nous parait plus claire qu'un tableau de tableau de couleur qui aurait été possible aussi.

Enum Color : Liste les trois couleurs possibles d'une case (noir, blanc ou vide). On devait la faire pour faciliter la création de case et bien distinguer les couleurs des cases.

Classe Position : Ce sont les coordonnées des cases à rentrer dans le plateau. On a choisi de faire cette classe pour faciliter la transition des entrées du joueur en coordonnées utilisables au jeu mais aussi pour rassembler les lignes et les colonnes d'un plateau. Mais aussi pour récupérer les positions des cases adjacentes.

Enum Direction : Contient toutes les directions cardinales et leur attribue une valeur de déplacement horizontale et verticale. Cette énumération était très utile pour récupérer les coordonnées des pions adjacents ou pour vérifier l'alignement des pions.

Classe InvalidCoordinatesException : Hérite d'Exception. Permet de gérer la rentrée invalide de coordonnées d'un joueur. Cette exception est utile car elle permet de redemander au joueur de donner une bonne position avec un message adéquat.

On utilise une HashSet pour la méthode possibleActions dans la classe Match car on a besoin de quelque chose de dynamique étant

donné que le nombre de coups proposé par cette méthode est variable. De plus, on a choisi de faire un HashSet car possibleActions ne doit pas contenir de doublons.

On utilise une ArrayList pour la méthode near de la classe Position car ArrayList est un tableau dynamique or on en a besoin car le nombre de voisins diffère d'un pion à l'autre.