### TRAVAIL SYNTHÈSE : le jeu de XIANGQI

**( étape 1 : classe Piece et ses sous-classes )**

Les origines du jeu d'échecs remontent à environ 200 av. J.-C. . Elles sont attribuées à Han Xin, un leader militaire chinois, qui a trouvé ce moyen pour occuper ses troupes bloquées durant les rudes hivers dans les steppes. Ce travail pratique modélisera donc un jeu d'échecs chinois ( le XIANGQI « échiquier des éléphants » ); à ce moment-ci, on s'en tiendra aux mouvements « théoriques » des pièces.

Ce jeu est similaire au jeu d’échecs occidental à la différence que les pièces reposent sur les intersections des cases plutôt que dans les cases. Le déplacement des pièces est dans certains cas différent.

La classe Position vous est fournie. Elle représente la position ( donc l’intersection ) que pourrait prendre une pièce sur un échiquier. Allez voir au verso la numérotation des lignes et des colonnes.

La classe abstraite Piece vous est fournie. Elle comprend une méthode concrète, soit la norme mathématique entre la position de départ et celle d'arrivée de la pièce. La norme équivaut à l'équation x² + y² = norme où x et y sont les distances horizontales et verticales séparant les positions/intersections de départ et d'arrivée. Elle vous sera peut-être utile à certains moments.

Votre travail consiste à construire 7 sous-classes de la superclasse abstraite Piece, soit Roi, Mandarin, Cavalier, Elephant, Char, Bombarde et Pion. Elles doivent être sous-classes de Piece. Pour chacune de ces classes, le travail consiste principalement à deux choses :

1. Coder un constructeur pour chacune des classes
2. Construire la méthode estValide(Position depart, Position arrivee) pour chacune des classes

La méthode estValide(Position depart, Position arrivee) prend en paramètres les positions de départ et d'arrivée de la pièce. **Elle retourne si oui ou non la location d'arrivée est possible étant donné la location de départ.**

**À ce stade-ci, on ne se préoccupe pas des autres pièces qui pourraient bloquer le déplacement. Également, on considère un déplacement nul ( où le depart est identique à l'arrivee ) comme étant VALIDE.**

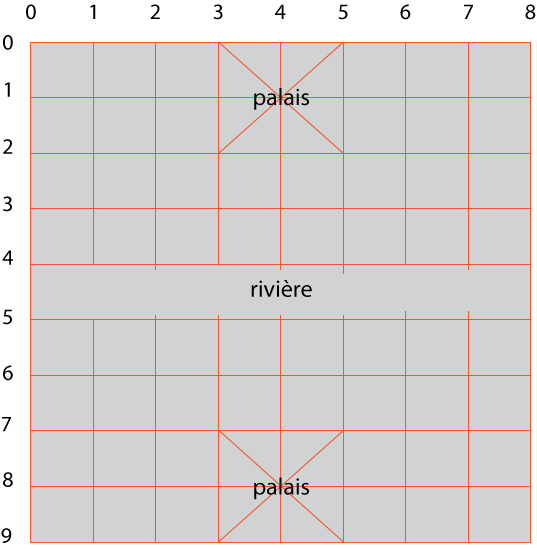
1. Finalement, vous devrez tester votre projet à l'aide d'une classe de tests JUnit. ( 10 tests environ )

ANNEXE

Nom des pièces :

|  |  |
| --- | --- |
| Roi | r |
| Mandarin / garde | m1, m2 |
| Elephant | e1, e2 |
| Cavalier | c1, c2 |
| Char | t1, t2 |
| Pion | p1,p2,p3,p4,p5 |
| Bombarde / Canon | b1, b2 |

***NOIRS ( en haut )***



***ROUGES ( en bas )***

Pour comprendre le mouvement des pièces :

<http://jeuxdesociete.free.fr/jeux/jeu-xiangqi.html> ( très bien )

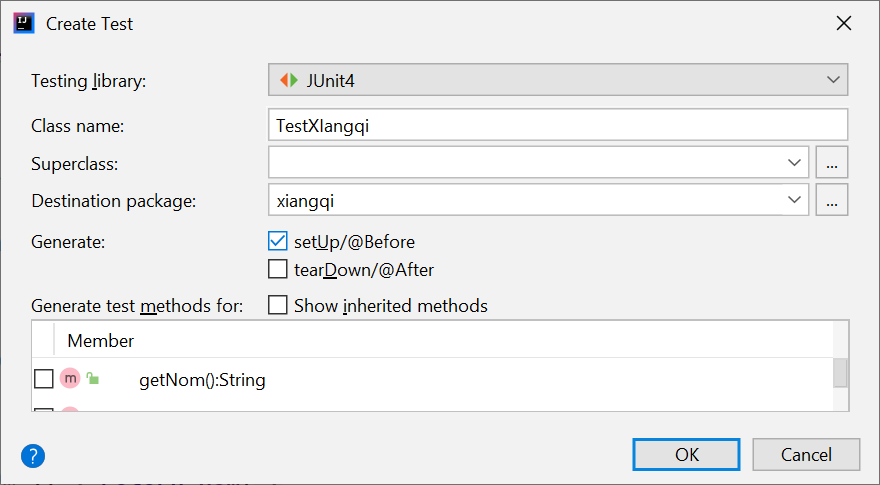
<https://fr.wikipedia.org/wiki/Xiangqi>

<https://www.youtube.com/watch?v=0UuDiL9CLPI> ( en anglais )

<https://brainking.com/en/GameRules?tp=68>

Procédure pour créer une classe de tests JUnit

* Cliquer sur le nom de la classe Piece dans l'entête de votre code et faire ALT – ENTER
* Sur la fenêtre qui apparaît, cliquer sur create Test



* Une autre classe sera ainsi créée, rappelez-vous ( voir TP1 )
  + La méthode setUp sera appelée avant chaque méthode de test, c'est un bon endroit pour créer les pièces
  + Les méthodes de tests doivent commencer par l'annotation @Test
  + À l'intérieur des méthodes de test, vous utilisez assertEquals, assertTrue ou assertFalse