**Travail synthèse**

**(partie 2: classes Intersection et Echiquier)**

**Jeu d’échecs : fonctionnalités de base**

***la classe Intersection***

Nous allons débuter par modéliser une Intersection de l’échiquier. Quelles seront ses données et ses méthodes ?

données : un objet private Piece p; si l’intersection est vide piece = vide

méthodes :

* constructeurs :deux constructeurs : 1 pour créer une Intersection vide, 1 pour créer une Intersection occupée par une pièce
* get / set
* méthode estOccupee() : permet de retourner si l’intersection est occupée ou non
* méthode estOccupeeParCouleur( String couleur ) : permet de retourner si l’intersection est occupée, et qu’elle est occupée par une pièce de la couleur passée en paramètre. Ne pas écrire ``if(pièce.getcouleur=``noit``)
* méthode estOccupeeParAdversaire ( String couleur ) : permet de retourner si l’intersection est occupée, et qu’elle est occupée par une pièce de la couleur contraire à celle passée en paramètre.`couleur inverse

***la classe Echiquier***

• Un échiquier sera donc un tableau 10x9 ( 10 lignes, 9 colonnes ) d’objets Intersection ; il s’agit donc de ses seules données :

private Intersection[][] jeu ;

• Le constructeur d’Echiquier devra donc créer le tableau ( il a été seulement déclaré  ) et créer également chacun des objets Intersection qui en fait partie. On créera un échiquier avec des **intersections vides au départ.**

* Intégrez à votre projet l'interface MéthodesEchiquier située dans LÉA. Votre classe Echiquier devra donc mettre en oeuvre cette interface.

• Il nous faut une méthode getIntersection qui, à partir d’un i ( rangée /ligne ) et d’un j ( colonne ) donné, nous retournera l’intersection correspondante.

• Il nous faut également une méthode debuter() permettant d’installer chacune des pièces sur son intersection de départ. On pourra le faire à l’aide de la méthode setPiece de Intersection.

Pubblic void debutrer (){

Jeu [0][0].setPiece(new char(“c1”, noir))

Intersection

}

• La méthode cheminPossible (Position depart, Position arrivee) est certainement celle qui vous causera le plus de soucis à cause du nombre de cas à traiter. Cette méthode vérifie si la pièce peut se frayer un chemin entre sa position de départ et celle d’arrivée. De plus, on vérifiera si la position d’arrivée n’est pas occupée par une pièce de la même couleur ; si c’est le cas, le chemin n’est évidemment pas possible.

**IMPORTANT :**

* **On suppose le déplacement nul comme étant valide.**
* **On suppose le déplacement soumis à la méthode comme étant théoriquement valide (estvalide vaut true)**

**CAS SPÉCIAUX :**

* **Bombarde : pour capturer une pièce, il doit y avoir exactement une pièce entre la Bombarde et la pièce à capturer.**
* **Roi : en aucun temps, les Rois doivent se situer un en face de l’autre ( dans la même colonne, il doit y avoir une pièce entre les deux Rois ) 🡪 voir méthode suivante**
* **Cavalier : contrairement au jeu d’échecs normal, il peut être bloqué par une pièce située à l’intersection à côté de lui, dépendant de sa direction / intersection d’arrivée**

• Une dernière méthode, roisNePouvantEtreFaceAFace, sert justement à vérifier ce dernier fait. En fonction de la position de départ et de la position d’arrivée d’une pièce à déplacer, vous devrez vérifier que le déplacement de la pièce n’amènera pas les deux rois à être face-à-face ( dans la même colonne ) sans aucune pièce les séparant. N’oubliez pas que, comme dans cheminPossible, vous ne devez pas déplacer les pièces pour le moment.

* Ces deux dernières méthodes seront passablement longues; vous êtes invités à les « découper » en plusieurs sous-méthodes

Pour comprendre le mouvement des pièces :

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

public boolean cheminPossible(Position depart, Position arrive){

//vers le haut  
 if(depart.getColonne() == arrive.getColonne())  
 if(arrive.getLigne() < depart,getLinge())  
 for (int i = depart.getlingne() -1; i> arrive.getligne; i-- ){  
 jeu[i][depart.getColonne()].estOccuper())  
 return false;  
 }  
  
}