IUT de Paris Descartes – Base de la Programmation Objet

Travaux Dirigés – Sujet n°11

Objectifs

— Étude et critique de code existant.
— Restructuration pour corriger ses défauts et augmenter son extensibilité.

Fin de partie d'échec

Le point de départ de notre travail est le code source fourni en annexe. Il s'agit d'un échiquier pour lequel on ne s'intéresse qu'aux fins de partie où ne s'opposent que deux rois et une dame. Le joueur blanc dispose de son roi et de sa dame, le joueur noir uniquement de son roi. Une première étude a porté sur les règles de déplacement. Un roi peut se déplacer d'une case dans n'importe quelle direction, une dame se déplace en diagonale d'un nombre quelconque de cases (qui ne doivent pas être occupées). Dans la réalité une dame peut se déplacer dans toutes les directions. Cette simplification est réalisée pour simplifier les algorithmes mais ne change pas la nature de l'exercice.

Les méthodes `peutAllerEn` des classes `Roi` et `Dame` implémentent ces règles. Elles vérifient (1) que la destination est bien dans l'échiquier, (2) que le déplacement prend une direction légale (d'une case pour les rois, en diagonal pour les dames) et que les cases survolées sont libres dans le cas particulier des dames.

Les attributs et les autres méthodes de ces deux classes sont faciles à comprendre.

À terme, la classe `Echiquier` doit permettre de jouer une fin de partie. Seule une ébauche a été réalisée. Elle porte toujours sur la légalité des déplacements. Deux situations sont à prendre en compte.

— Un roi n'a pas le droit de se déplacer vers une case si celle-ci est attaquée par une pièce adverse.
— Le roi noir peut manger la dame blanche.

La classe `Echiquier` implémente ces dernières règles. La dame blanche peut atteindre sa cible si la case est libre et le déplacement légal. Pour pouvoir se déplacer, le roi blanc doit en plus vérifier que le roi noir n'attaque pas cette case. Le cas du roi noir est un tout petit peu plus compliqué. Si la case n'est pas libre, elle peut être occupée par la dame blanche (qui sera mangée). De plus, il faut s'assurer qu'aucune pièce blanche (dame ou roi) ne peut attaquer cette case.

1. Lisez le programme donné en annexe et décrivez les défauts (pas les solutions) que vous remarquez.
2. Énoncez des corrections possibles aux défauts que vous avez remarqué.
3. Proposez une classe (nommée `Pièce`) factorisant ce qui est commun aux pièces.Simplifiez en conséquence les classes `Roi` et `Dame` en vous reposant sur l'héritage.
4. Faites en sorte que la classe `Echiquier` ne repose que sur cette classe.

Annexe

```
public class Roi {
  private int colonne, ligne;

  public Roi(int colonne, int ligne) {
```

```
    this.colonne = colonne; this.ligne = ligne;
  }

  public boolean occupe(int colonne, int ligne) {
    return this.colonne == colonne && this.ligne == ligne;
  }

  public boolean peutAllerEn(int colonne, int ligne) {
    if (colonne < 1 || colonne > 8 || ligne < 1 || ligne > 8)
      return false;
    if (Math.abs(this.colonne - colonne) > 1 || Math.abs(this.ligne - ligne) > 1)
      return false;
    return true;
  }
}
```

```
public class Dame {
  private int colonne, ligne;

  public Dame(int colonne, int ligne) {
    this.colonne = colonne; this.ligne = ligne;
  }

  public boolean occupe(int colonne, int ligne) {
    return  this.colonne == colonne && this.ligne == ligne;
  }

  public boolean peutAllerEn(int colonne, int ligne, Echiquier e) {
    if (colonne < 1 || colonne > 8 || ligne < 1 || ligne > 8)
      return false;
    if (Math.abs(this.colonne - colonne) != Math.abs(this.ligne - ligne))
      return false;
    int dx = this.colonne - colonne > 0 ? -1 : 1, dy = this.ligne - ligne > 0 ? -1 : 1;
    for (int i = 1; i < Math.abs(this.colonne - colonne); ++i)
      if (e.estLibre(this.colonne + i * dx, this.ligne + i * dy))
        return false;
    return true;
  }
}
```

```
public class Echiquier {
  private Dame dameBlanche = new Dame(4, 1);
  private Roi roiBlanc = new Roi(5, 1);
  private Roi roiNoir = new Roi(5, 8);

  public boolean estLibre(int colonne, int ligne) {
    return !dameBlanche.occupe(colonne, ligne) && !roiBlanc.occupe(colonne, ligne)
        && !roiNoir.occupe(colonne, ligne);
  }

  public boolean dameBlancheJouable(int colonne, int ligne) {
    return estLibre(colonne, ligne) && dameBlanche.peutAllerEn(colonne, ligne, this);
  }

  public boolean roiBlancJouable(int colonne, int ligne) {
    return estLibre(colonne, ligne) && roiBlanc.peutAllerEn(colonne, ligne)
        && !roiNoir.peutAllerEn(colonne, ligne);
  }

  public boolean roiNoirJouable(int colonne, int ligne) {
    return ((estLibre(colonne, ligne) || dameBlanche.occupe(colonne, ligne))
        && roiNoir.peutAllerEn(colonne, ligne) && !roiBlanc.peutAllerEn(colonne, ligne)
        && !dameBlanche.peutAllerEn(colonne, ligne, this));
  }
}
```