Faire jouer l'ordinateur au Mastermind

October 16, 2006

Votre classe `Finder` est à rendre par courier électronique à `Luc.Maranget@inria.fr` avant le 9 janvier, 23h59.

Objectif

Dans le TP noté c'est l'ordinateur qui choisi le secret et le joueur humain qui tente de le découvrir. Il s'agit maintenant de remplacer le joueur humain par un programme.

Comment procéder

Comme dans l'examen, le jeu est simplifié. Étant donné un entier n, le secret est une suite ordonnée de n couleurs distinctes parmi $2n$, notées comme les entiers $1, \ldots, 2n$.

Choix du secret

Le joueur qui choisit le secret est, comme dans l'examen, représenté par un objet de la classe `MasterMind`, crée par le constructeur `MasterMind(int n)`.

Le code de cette classe `MasterMind` ainsi que des classes `List` et `Result` sont donnés. Ces codes sont à peu près conformes aux descriptions de l'énoncé. Les objets de la classe `MasterMind` sont en outre équipés d'une méthode `toString` qui permet l'affichage du secret.

Approche « force brute »

Si l'entier n est raisonnablement petit, on peut avoir recours à l'approche suivante qui ne brille pas par sa subtilité.

1. Commencer par produire toutes les combinaisons possibles pour le secret. Il s'agit de l'ensemble $A(n, C)$ des arrangements de taille n des couleurs $\mathrm{C} = \{1, \ldots, 2n\}$ dont la définition inductive est rappelée ci-dessous.

$$A(k+1, C) = \bigcup_{c \in C} \{\, (c; X) | X \in A(k, C \setminus \{c\}) \,\}, \quad A(0, C) = \{\emptyset\}$$

2. Proposer au joueur `MasterMind` une des combinaisons possibles g, en invoquant sa méthode `processGuess`. Le joueur répond par un objet r de la classe `Result`.

3. Si la combinaison proposée est la bonne (méthode `winningResult` du joueur `MasterMind`) le jeu est fini.

4. Sinon, ne garder parmi les combinaisons possibles que celles qui, confrontées à l'essai g, produisent le `Result` r, et recommencer en 2.

Interface

Le jeu est contrôlé par un arbitre `Referee` dont voici la méthode `main` (un peu simplifiée).

```
public static void main(String [] arg) {
  int n = 5 ;
  if (arg.length > 0) { n = Integer.parseInt(arg[0]) ; }
  MasterMind m = new MasterMind (n) ;
  System.out.println("Le secret est: " + m) ;

  List sol = Finder.find(m) ;
  if (!m.winningResult(m.processGuess(sol))) {
    System.out.println("Finder a perdu (la tête)") ;
  } else {
    System.out.println("Finder a trouvé en " + (m.getUsedGuesses()-1) + " coups") ;
  }
}
```

C'est-à-dire que l'arbitre lance une partie pour n donné en argument (ou 5 par défault), contrôle la validité de la réponse du trouveur automatique et affiche le nombre d'essais utilisés.

Plus précisément votre trouveur est la classe `Finder` dont la méthode `static List find(MasterMind m` renvoie le secret caché dans l'objet m (notez que la taille du secret n'est pas secrète, c'est `m.length`). Bien évidemment, votre méthode `find` ne doit pas faire appel à la méthode `toString` de m, mais seulement aux méthodes `processGuess` et `winningResult`.

Quelques idées de prolongement

- La plupart du temps, le `Finder` peut se dispenser du dernier coup de vérification de sa solution et laisser ce soin à l'arbitre. Implémenter cette astuce.

- Dénoncer la triche ! Votre programme sera également confronté à un `MasterMind` tricheur qui donne des réponses incohérentes. Identifier cette possibilité et dénoncer le tricheur en invoquant la méthode `cheaterSpotted()` de la classe `Referee`.

- Programmer plus efficace. Normalement, votre `Finder` doit traiter le cas $n = 6$ sans difficulté. Essayer de resoudre $n = 7$ en un temps et surtout une consommation mémoire raisonnable.