Projet: Le fameux Puissance 4

Matthieu Boutier
(boutier@pps.univ-paris-diderot.fr)

16 novembre 2015

Puissance 4 est un jeu de société à deux joueurs. Entre les deux joueurs se trouve une grille de 6 lignes et 7 colonnes, disposée verticalement. Chaque joueur, lors de son tour, doit mettre un de ses pions dans une colonne (et le pion tombe). Le premier joueur à avoir 4 de ses pions alignés (en ligne, colonne ou diagonale) remporte la partie.

Votre programme affichera quelque chose comme ça :

```
  1   2   3   4   5   6   7
|   |   |   |   |   |   |   |
|---+---+---+---+---+---+---|
|   |   |   |   |   |   |   |
|---+---+---+---+---+---+---|
|   |   |   | = |   | @ |   |
|---+---+---+---+---+---+---|
|   |   | @ | @ | = | @ |   |
|---+---+---+---+---+---+---|
|   |   | = | = | @ | = |   |
|---+---+---+---+---+---+---|
|   | @ | = | @ | = | @ | = |
\=========================/
Joueur 2, félicitations, vous avez gagné!
```

Mise-en-place Téléchargez Power4.java depuis ma page web : il s'agit d'un patron qui contient la liste des fonctions utilisables (vues en TP). Testez au fur et à mesure vos fonctions !

1 Initialisation (**)

La grille de jeu est représentée comme un tableau d'entiers de 6 lignes et 7 colonnes. On code par 0 un emplacement vide, 1 lorsqu'un pion du joueur 1 est présent, et 2 lorsqu'un pion du joueur 2 est présent.

La représentation interne de la grille d'exemple précédente est donc le tableau :

```
    game = [[ 0,  0,  0,  0,  0,  0,  0],
            [ 0,  0,  0,  0,  0,  0,  0],
            [ 0,  0,  0,  2,  0,  1,  0],
            [ 0,  0,  1,  1,  2,  1,  0],
            [ 0,  0,  2,  2,  1,  2,  0],
            [ 0,  1,  2,  1,  2,  1,  2]]
```

Écrivez une fonction `newGame` comme décrite ci-dessous, qui crée une grille vide, c'est-à-dire initialisée à 0.

```java
/** Renvoie un nouveau plateau de jeu: tableau de 6 x 7. */
public static int[][] newGame();
```

2 Dessin (**)

On représente le joueur 1 par un `@`, et le joueur 2 par un `=`. (*) Écrivez une fonction `playerToString` qui effectue la conversion désirée. Une fonction sans conditionnelle sera appréciée.

```java
/** convertit un numéro de joueur (0, 1 ou 2) en sa représentation
    textuelle (respectivement espace, @ ou =). */
public static String playerToString(int player);
```

(**) Écrivez une procédure `displayGame` qui affiche le plateau. L'idéal serait d'avoir la même représentation qu'au début du sujet. Vous pouvez faire un affichage plus simple si ça vous semble trop difficile. On devra afficher en haut le numero des colonnes (de 1 à 7).

```java
/** display the game */
public static void displayGame(int[][] game);
```

3 Passer au joueur suivant (*)

(*) Écrivez une fonction `nextPlayer` qui permet de passer au joueur suivant : si le joueur est 1, la fonction renverra 2, et si le joueur est 2, la fonction renverra 1. Une fonction sans conditionnelle sera appréciée.

```java
/** Renvoie le numéro du prochain joueur à jouer. */
public static int nextPlayer(int player);
```

4 Jouer une colonne (**)

(*) Écrivez une fonction `isFull` qui renvoie vrai ssi une colonne est pleine.

```java
/** Renvoie vrai si une colonne est pleine. */
public static boolean isFull(int[][]game, int col) {
```

(**) Écrivez une fonction `playColumn` qui place un pion du joueur donné au fond de la colonne demandée. On appréciera l'utilisation d'une boucle **`while`**.

```java
/** Ajoute un pion du joueur "player" dans la colonne "col" du jeu. */
public static void playColumn(int[][]game, int col, int player);
```

5 Condition de victoire

Nous nous intéressons maintenant aux conditions de victoire, et allons écrire une fonction qui permet de regarder s'il existe un alignement de 4 pions identiques quelque part dans la grille : si c'est le cas, la fonction nous donnera le numéro du joueur possédant ces pions.

Un alignement diagonal (**) Écrivez une fonction `d2Align` qui renvoie vrai s'il y a un alignement de 4 pions en diagonale montante droite depuis les coordonnées `i` et `j`. On fera attention à ne pas dépasser le tableau. Aidez-vous de la spécification de la fonction, qui donne en particulier le repère.

```
/**
   Renvoie vrai si (i, j) est le bas d'une diagonale montante droite:
         j                     j
         |                     |
         |     @               |     @            +----> j
         |   @                 |   =              |
         | @                   | =                |
   i --- @               i --- @                  v i
        (true)               (false)
 */
public static boolean d2Align(int[][]game, int i, int j);
```

Trouver le vainqueur (**) Écrivez une fonction `winner` qui utilise les fonctions `vAlign`, `hAlign`, `d1Align`, `d2Align` et qui renvoie le numéro du joueur gagnant, ou 0 s'il n'en est pas.

```
/** Renvoie le numéro du joueur gagnant, ou 0 s'il n'en est pas. */
public static int winner(int[][]game);
```

Indication On doit donc parcourir la grille à la recherche d'un alignement de 4 pions du même joueur. Une fois trouvé, on renvoie le numéro de ce joueur.

6 Assemblage des briques

Nous avons maintenant tout ce qu'il faut pour faire le programme : écrivez le `main`.

7 Le plus dur ?

Remportez une partie contre vos chargés de TD ou TP.

8 Extensions possibles

Toute extension est la bienvenue. Cette section en contient deux qui me semblent particulièrement intéressantes, et amusantes à programmer (et à utiliser). Si vous avez envie d'implémenter quelque chose, mais que vous ne savez pas comment vous y prendre, n'hésitez pas à demander !

8.1 Tout seul, ce n'est pas drôle.

Vos enseignants ne sont pas toujours là, et vous avez envie de jouer sans devenir schizophrène. Une mauvase solution consiste à renoncer à jouer, et une bonne solution consiste à créer une intelligence artificielle (IA).

L'IA sera codée par une fonction `iaChoice`, dont le but est de renvoyer la colonne choisie par l'IA. Cette fonction prend en paramètres le jeu actuel, le niveau de difficulté demandé, et le numéro du joueur courant. Nous l'écrirons au fur et à mesure.

8.1.1 Une heuristique simple

La base d'une intelligence artificielle est d'évaluer l'état du jeu pour un joueur donné : plus la note donnée est élevée, et mieux est l'état du jeu. La note la plus élevée doit donc être réservée à un état de victoire, et la note la plus basse à un état de défaite.

Définition de l'heuristique Écrivez une fonction `heuristic` qui évalue l'état d'un jeu pour un joueur donné. Une heuristique simpliste associe `Integer.MAX_VALUE` à un jeu gagnant pour le joueur `player`, `Integer.MIN_VALUE` à un jeu perdant pour `player`, et 0 dans les autres cas.

```
1 public static int heuristic(int[][] game, int player);
```

Utilisation de l'heuristique C'est à l'IA de jouer : pour faire le choix le plus stratégique possible, l'ordinateur va tenter de déterminer quel est son intérêt pour le coup d'après. Pour cela, `iaChoice` va appeler une fonction `predict`, qui revoie l'heuristique pour une nouvelle configuration. La fonction `iaChoice` n'aura alors qu'à en choisir une.

Commençons par permettre à l'ordinateur de réfléchir sans altérer le jeu : écrivez une fonction `copy`, qui réalise une copie du jeu.

```
1 public static int[][] copy(int[][] game);
```

Écrivez maintenant la fonction `predict`, qui prend en paramètres le jeu, le joueur courant (ordinateur), et la colonne où jouer, et renvoie l'heuristique associée à la nouvelle configuration.

```
1 public static int[] predict(int[][] game, int col, int player);
```

Enfin, écrivez `iaChoice`, qui tente de jouer chaque colonne. Votre fonction devra finalement renvoyer une colonne pour laquelle l'heuristique est la plus élevée. Attention à ne pas jouer dans les colonnes pleines. Le paramètre `player` indique le numéro du joueur courant (joué par l'ordinateur).

```
1 public static int iaChoice(int[][] game, int player);
```

8.1.2 L'algorithme min-max

En l'état, notre IA est mauvaise. Bien sûr, vous avez pu améliorer votre heuristique pour avoir quelque chose de correct, mais rien ne vaut la prédiction à plusieurs coups d'avance. L'algorithme *min-max* consiste à simuler les possibilités de jeu à plusieurs tours d'avance, et à retenir : lorsque c'est au tour du joueur (IA), la meilleure heuristique (on veut jouer au mieux — max) ; et lorsque c'est au tour de l'opposant, la moins bonne (on suppose que l'opposant va jouer au mieux — min).

Modifiez la fonction `predict`. Celle-ci prend maintenant deux nouveaux paramètres : la profondeur `depth` (le nombre de coups d'avance que l'on veut prévoir), et le tour courant `turnOfPlayer` (qui désigne le numéro du joueur jouant ce tour simulé). Si la profondeur est nulle, ou si l'heuristique obtenue rend compte d'un état de victoire ou de défaite, l'heuristique est renvoyée comme tel, et la fonction est donc similaire à la version précédente de `predict`.

Dans le cas contraire, il faut simuler l'action du joueur suivant : tout d'abord déterminer ce joueur, puis le faire jouer sur chaque colonne (non pleine). Si ce joueur est le même joueur que `player` (pour lequel on optimise l'heuristique), on retiendra l'heuristique maximale ; sinon, l'heuristique minimale.

```
1 public static int[] predict(int[][] game, int col, int depth,
2                              int player, int turnOfPlayer);
```

Voilà, vous pouvez maintenant modifier `iaChoice` pour qu'elle prenne un paramètre `level`, qui renseigne le niveau de difficulté de l'IA (et qui correspond à la profondeur demandée). (Notez que vous ne pourrez sans doute pas dépasser les 7 coups sans avoir de très longs temps d'attente.)

8.1.3 Un peu de suspense !

Très bien ! Nous avons (enfin) une IA digne de ce nom. En revanche, elle joue toujours pareil, ce qui n'est pas très palpitant. Modifiez `iaChoice` pour qu'elle calcule un tableau contenant toutes les meilleures valeurs d'heuristiques ; puis retourne aléatoirement l'une de celles-ci.

On pourra utiliser la classe `Random` de java, qui s'utilise un peu comme la classe `Scanner`. Commencez par copier cette ligne à côté de celle de `Scanner`.

```
1     public static java.util.Scanner sc = new java.util.Scanner(System.in);
2     public static java.util.Random rand = new java.util.Random();
```

Vous pouvez maintenant appeler `rand.nextInt(x)`, qui renvoie un nombre entre 0 inclus et x exclus. Par exemple :

```
1 int a = rand.nextInt(0);  /* a == 0 */
2 int b = rand.nextInt(2);  /* b == 0, 1, 2 ou 3 */
3 int c = rand.nextInt(42); /* c == 0, 1, 2,... ou 41 */
```

8.2 Un jeu en couleurs !

Nous avons déjà vu un caractère un peu spécial, le `"\n"`, qui a pour valeur 10 (ou 0x0a, ou 012), et qui correspond à un retour à la ligne. Il existe un autre caractère, de valeur 27 (0x1b ou 033), appelé caractère d'échappement (*escape*), et qui peut être utilisé pour configurer le terminal : on peut ainsi changer les couleurs de fond et de caractère lors de l'écriture d'un nouveau caractère, repositionner le curseur à un emplacement précis, faire clignoter le texte, etc.

Par exemple, le code suivant affiche `Yeah` en bleu clignotant. Avec `print`, nous envoyons une chaîne de caractère au terminal. Lorsque celle-ci contient le caractère d'échappement `esc`, ce qui suit est intercepté par le terminal. Ainsi, la chaîne `esc + "[5;34m"` configure le terminal en clignotant (5), et en bleu (34). Le `m` indique que la commande visait à changer l'affichage.

```
1     public static void main(String args[]) {
2         char esc = 0x1B;
3         System.out.print(esc + "[5;34m"); /* set: blink; blue */
4         System.out.println("Yeah");
5         System.out.print(esc + "[0m"); /* reset */
6     }
```

Remarquez qu'il est possible de mettre toutes ces opérations sur une seule ligne, et de mettre le caractère d'échappement directement dans la chaîne de caractère, en octal.

```
1     public static void main(String args[]) {
2         System.out.print("\033[5;34mYeah\033[0m");
3     }
```

Vous trouverez beaucoup de documentation sur Internet qui vous permettront d'utiliser les caractères d'échappements [1].

1. Par exemple, http ://www.termsys.demon.co.uk/vtansi.htm.