Projet: Atoms

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

16 novembre 2015

Atoms [1] est un jeu (presque) inconnu, à deux joueurs. Chacun son tour, les joueurs placent un atome de leur couleur sur une grille de 8 x 8, soit sur une case du joueur, soit sur une case vide. Lorsque le nombre d'atomes d'une case atteint (ou excède) le nombre de voisins de cette case, elle explose et envoie un atome sur chacun de ses voisins. S'il y a des atomes appartenant à l'autre joueur, ceux-ci sont "convertis". Une réaction en chaîne peut évidemment se produire, et c'est tout l'intérêt du jeu !

Voici un exemple de jeu (les atomes du joueur 1 sont représentés par des `@`, et ceux du joueur 2 par des =).

```
    0   1   2   3   4   5   6   7
  /================================\
0 |   |   |   |   |   |   |   |   |
  |---+---+---+---+---+---+---+---|
1 | ==|@@@|   |   |   |   |   |   |        +----->
  |---+---+---+---+---+---+---+---|        |     i
2 |   |   |   |   |   |   |   |   |        |
  |---+---+---+---+---+---+---+---|        v j
3 |   |   |   | @ |   |   |   |   |
  |---+---+---+---+---+---+---+---|
4 |   |   |   |===| @ |   |   |   |
  |---+---+---+---+---+---+---+---|
5 |   |   |   |   |   |   |   |   |
  |---+---+---+---+---+---+---+---|
6 |   |   |   |   |   |   |   |   |
  |---+---+---+---+---+---+---+---|
7 |   |   |   |   |   |   |   |   |
  \================================/
```

Voici ce qui se produit si le joueur 1 joue en (1, 1) : la case (1,1), qui a 4 voisins, explose de tous côtés lorsqu'elle se retrouve avec 4 atomes, y compris en (0,1), où les atomes du joueur 2 sont convertis au joueur 1. La case en (0,1) se retrouve alors avec 3 atomes, et n'a que 3 voisins : elle explose à son tour. Le jeu est ensuite dans un état stable.

```
    0   1   2...              0   1   2...              0   1   2...
  /=============           /=============           /=============
0 |   |   |   |           0 |   | @ |   |         0 | @ | @ |   |
  |---+---+---+-   puis     |---+---+---+-  puis     |---+---+---+-
1 | ==| 4@|   |   ----->  1 |@@@|   | @ |  ----->  1 |   | @ | @ |
  |---+---+---+-            |---+---+---+-           |---+---+---+-
2 |   |   |   |           2 |   | @ |   |         2 | @ | @ |   |
  |---+---+---+-            |---+---+---+-           |---+---+---+-
```

1. Ou *Atomic*.

Mise-en-place Téléchargez Atoms.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 et codage

La grille du jeu est représentée en mémoire comme une un tableau d'entiers à deux dimensions [2]. Chaque case doit renseigner à qui appartient la case, et le nombre d'atomes de la case. Nous choisissons de coder ces informations ainsi : le chiffre des centaines indique le numéro du joueur, et les deux chiffres des unités et des dizaines le nombre d'atomes de la case (le chiffre des unités suffirait probablement). Une case vide sera représentée par une appartenance au joueur 0, et le joueur n par une valeur de n.

Écrivez une fonction `newGame` qui renvoie une nouvelle grille de 8 x 8 entièrement initialisée à 0. Pour correspondre aux coordonnées cartésiennes (l'ordonnée est inversée, ce qui est assez courant), la première composante indiquera le numéro de colonnes, et la deuxième composante le numéro de lignes.

```
/** Renvoie un nouveau plateau de jeu: tableau de 8 x 8. */
public static int[][] newGame();
```

Écrivez une fonction `int getPlayer(int[][]game, int i, int j)` qui renvoie le numéro du joueur à la case (i, j). Dans l'exemple ci-dessous, `getPlayer(game, 1, 1)` renvoie 1.

Écrivez une fonction `int getNumber(int[][]game, int i, int j)` qui renvoie le nombre d'atomes de la case (j, i). Dans l'exemple ci-dessous, `getNumber(game, 0, 1)` renvoie 2.

```
    0   1   2...
  /=============
0 |   |   |   |
  |---+---+---+-
1 | ==|@@@|   |
  |---+---+---+-
2 |   |   |   |
  |---+---+---+-
```

2 Dessin

Nous allons créer des fonctions pour afficher une jolie représentation de ce jeu. Vous le voyez sur les exemples : chaque cellule est constituée de trois caractères. Nous voulons abstraire au maximum nos fonctions du numéro du joueur utilisé, afin de pouvoir réutiliser au mieux notre code.

Comme nous l'avons dit, les atomes du joueur 1 et 2 sont respectivement représentés par un `@` et un `=`. (*) Écrivez une fonction `playerToString` qui effectue la conversion désirée.

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

Écrivez une fonction `caseToString` qui étant donnés un nombre d'atomes et un joueur, renvoie une chaîne de 3 caractères représentant le contenu d'une cellule :

```
 /** Renvoie une chaîne de trois caractères représentant le contenu
     d'une cellule. */
public static String caseToString(int number, int player);
```

(**) Écrivez enfin une procédure `displayGame` qui affiche la grille de jeu.

2. Plus précisément un tableau de tableau d'entiers.

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

3 Passer au joueur suivant, et version 1

(*) 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);
```

Écrivez une fonction `play` qui ajoute un pion au joueur donné en paramètre aux coordonnées passées en paramètres. La fonction retournera un booléen indiquant si l'action a pu être réalisée (ce qui est le cas si la case n'appartient pas à un autre joueur).

```java
/** Ajoute un atome de player aux coordonnées (col, line).  Retourne
    vrai si le mouvement était possible, faux sinon (la cellule est
    occupée par un autre joueur). */
public static boolean play(int[][]game, int col, int line, int player);
```

Écrivez une fonction `atoms` qui exécute le jeu : votre programme demandera au joueur courant les numéros de ligne et de colonne où il veut jouer, puis il jouera le coup demandé, et il passera à l'autre joueur. (Alternativement, vous pouvez mettre directement le code dans le main, mais ça risque de vous gêner si vous voulez tester vos fonctions.)

```
$ java Atoms
    0   1   2   3   4   5   6   7
  /===============================\
0 |   |   |   |   |   |   |   |   |
  |---+---+---+---+---+---+---+---|
1 |   |   |   |   |   |   |   |   |
  |---+---+---+---+---+---+---+---|
[...]
  |---+---+---+---+---+---+---+---|
7 |   |   |   |   |   |   |   |   |
  \===============================/
Bienvenue dans Atoms, jouez en donnant le numéro de la ligne puis de la
colonne (0 à 9).
Tour 1.
Joueur 1, ligne ? 1
colonne ? 2

    0   1   2   3   4   5   6   7
  /===============================\
0 |   |   |   |   |   |   |   |   |
  |---+---+---+---+---+---+---+---|
1 |   |   | @ |   |   |   |   |   |
  |---+---+---+---+---+---+---+---|
[...]
  |---+---+---+---+---+---+---+---|
7 |   |   |   |   |   |   |   |   |
  \===============================/
Tour 2.
```

```
Joueur 2, ligne ?
```

4 Vers un état stable...

Notre jeu est un peu ennuyeux : les cases n'explosent pas, personne ne peut prendre les atomes des autres, personne ne peut gagner.

Écrivez une fonction `isUnstable` qui détecte si une case est dans un état instable, c'est-à-dire si elle contient au moins autant d'atomes qu'elle a de cases adjacentes.

```
/** Renvoie vrai si game[i][j] est instable (doit exploser). */
public static boolean isUnstable(int[][] game, int i, int j);
```

Écrivez une fonction `explose` qui fait exploser une case. Lorsqu'une case explose, un atome part dans chaque direction (disponible). S'il y a plus d'atomes dans la case que de cases adjacentes, les atomes restant restent dans la case. N'oubliez pas que cette opération permet de *convertir* des atomes ennemis !

```
/** Renvoie vrai si game[i][j] est instable (doit exploser). */
public static void explose(int[][] game, int i, int j);
```

Écrivez une fonction `convergence` qui fait exploser les cases instables du jeu, jusqu'à ce que toutes les cases soient stables. Cette fonction affichera le jeu à chaque explosion pour faire joli, et fera une pause de 50 millisecondes entre deux explosions pour qu'on y voit quelque chose (utilisez la fonction `sleep`).

```
/** Renvoie vrai si game[i][j] est instable (doit exploser). */
public static void convergence(int[][]game);
```

Modifiez votre fonction `atoms` pour faire converger le jeu vers un jeu stable, après chaque coup.

À ce stade là, nous avons un jeu qui fonctionne : il faut juste l'arrêter manuellement, par ce qu'il n'y a pas de condition de victoire. Par ailleurs, il peut arriver que le jeu ne converge jamais (perpétuelle réaction nucléaire).

5 Le jeu complet

Implémentez une fonction `winner` qui renvoie le numéro du joueur gagnant, ou 0 s'il n'en est pas.

```
/** Renvoie le numéro du joueur gagnant, ou -1 s'il n'en est pas. */
public static int winner(int[][]game);
```

Modifiez la procédure `convergence` pour être sûr qu'elle s'arrêtera sur une condition de victoire.

Modifiez la procédure `atoms` pour qu'elle arrête le jeu sur une victoire d'un joueur, et le félicite.

6 Le plus dur ?

Remportez une partie contre vos chargés de TD ou TP.

7 Extensions possibles

Nous proposons ici plusieurs extensions possibles : une intelligence artificielle, plus de joueurs, ou encore un jeu en couleurs. Ces extensions ne sont là qu'à titre d'exemple, mais n'hésitez pas à implémenter tout ce qui vous passe par la tête.

7.1 Une IA

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.

7.1.1 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 les coordonnées où jouer, et renvoie l'heuristique associée à la nouvelle configuration.

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

Écrivez une fonction `canPlay` qui renvoie si un joueur donné peut jouer dans une case (c'est-à-dire si la case lui appartient, ou est vide).

```
1 public static boolean canPlay(int [][]game, int i, int j, int player);
```

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

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

7.1.2 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 case (possible). Si ce joueur est le même joueur que `player` (pour lequel on optimise l'heuristique), on retiendra l'heuristique maximale ; sinon, l'heuristique minimale.

```
public static int[] predict(int[][] game, int x, int y, int depth,
                            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 3 coups sans avoir de très longs temps d'attente.)

7.1.3 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`.

```
    public static java.util.Scanner sc = new java.util.Scanner(System.in);
    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 :

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

7.2 Plus on est de fous...

Atoms peut se jouer à plus de deux joueurs : il suffit d'ajouter une "couleur" ! L'idéal serait de demander au début du jeu combien de joueurs vont participer (au moins 2). Et ensuite... :-)

7.3 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.

```
    public static void main(String args[]) {
        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 [3].

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