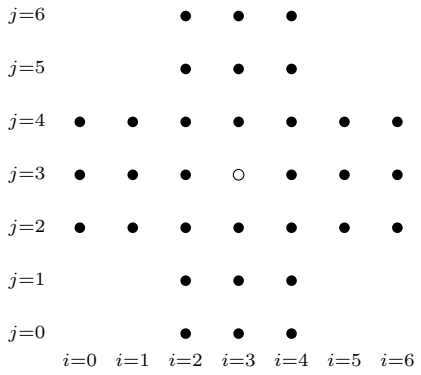
MPSI option info TP du 23 avril

Le jeu du solitaire résolu par Backtracking

Stéphane Legros Année scolaire 2024-2025

Nous nous intéressons dans ce TP à la version de base du jeu de solitaire : un plateau contenant des billes est initialement dans la position décrite ci-dessous, où les ronds noirs (resp. blancs) représentent les billes encore en jeu (resp. les cases vides).



À chaque coup de la partie, une bille située dans une case A peut sauter une seconde bille située dans une case B, à condition que B soit voisine de A et que la case C symétrique de A par rapport à B soit vide. La bille qui était en A se retrouve alors en C et la bille qui était en B sort du jeu. La partie est gagnée quand il ne reste plus qu'une bille sur le plateau. Une solution consiste donc en la donnée d'une suite de coups, permettant d'aboutir à un plateau contenant une unique bille. Pour coder ce jeu, nous représenterons le tableau comme un couple formé par un entier, égal au nombre de cases pleines, et d'une matrice de taille 7×7 : les trous sont associés à la valeur 1 ou 0, selon qu'ils contiennent ou non une bille, et les cases "hors jeu" sont associées à la valeur 2. Ainsi, la position dite de départ est définie en Caml par l'instruction :

```
let M = 32, [|[|2;2;1;1;1;2;2|];[|2;2;1;1;1;2;2|];[|1;1;1;1;1;1;1|];[|1;1;1;0;1;1;1|];
             [|1;1;1;1;1;1;1|];[|2;2;1;1;1;2;2|];[|2;2;1;1;1;2;2|]|];;
```

Un couple de la forme n, $\begin{pmatrix} 2 & 2 & ? & ? & ? & 2 & 2 \\ 2 & 2 & ? & ? & ? & 2 & 2 \\ ? & ? & ? & ? & ? & ? & ? \\ ? & ? & ? & ? & ? & ? & ? \\ ? & ? & ? & ? & ? & ? & ? \\ 2 & 2 & ? & ? & ? & 2 & 2 \\ 2 & 2 & ? & ? & ? & 2 & 2 \end{pmatrix}$ où les ? sont des 0 ou des 1 et où n est le nombre de 1 est appelée une `position`. Un `site` est un couple (i, j) avec $0 \leq i, j \leq 6$ et un *coup* est un couple (S, d) où S est un site et d une direction, i.e. un des quatre caractères g, d, h ou b. Une partie sera ainsi une liste de coups.

L'affichage de la matrice ne correspond pas à celle de la grille : veiller à bien utiliser le graphique du haut de la page pour bien prendre en compte les déplacements. Par exemple, aller vers la droite se fait en augmentant i.

Vous trouverez à l'adresse `https://cahier-de-prepa.fr/mp*-corneille` le fichier `solitaire.ml`, qui contient la définition des types utilisés, de quelques positions et le code d'une fonction `dessin: position -> UNIT`, qui vous permettront de tester vos précédures.

Nous utilisons dans ce TP une méthode de recherche exhaustive d'une solution : à chaque étape du calcul, nous cherchons récursivement les solutions en testant l'un après l'autre tous les coups possibles. La méthode la plus naturelle consiste à calculer toutes les solutions en écrivant une fonction `solutions: position -> partie list`, l'appel `solutions (n,M)` conduisant aux calculs suivants :

- si la position est gagnante, i.e. si $n = 1$, nous renvoyons une liste contenant la liste vide : il existe une seule partie gagnante, qui consiste à ne jouer aucun coup ;
- sinon, nous calculons tous les coups possibles $C_1, C_2, \ldots, C_k$ à partir de la position (n, M). Deux cas sont alors à distinguer :
  a) si $k = 0$, nous renvoyons la liste vide car la position est perdante : il n'y a pas de partie gagnante à partir de M.
  b) sinon, nous créons une pile vide $Pile$; pour chaque C_i, nous appliquons récursivement la fonction `solutions` à la position $(n-1, M_i)$ obtenue à partir de M en jouant le coup C_i. Cela donne une liste de parties $[P_1; P_2; \ldots; P_m]$, qui sont les parties gagnantes depuis la position M_i : nous ajoutons à $Pile$ les parties $C_i :: P_1$, $C_i :: P_2$, ...,$C_i :: P_m$ qui sont gagnantes depuis la position M. Il reste ensuite à retourner la liste associée à $Pile$.

1) Écrire une fonction `coups_possibles` qui, appliquée à une position (n, M), renvoie la liste des coups possibles depuis la position M.

2) Écrire une fonction `nouvelle_position` qui, appliquée à une position (n, M) et à un coup possible C, renvoie la position obtenue en jouant C depuis n, M.

3) Écrire la fonction `solutions`.

4) Écrire une fonction `solution` telle que l'appel `solution (n,M)` renvoie, si elle existe, une solution depuis la position (n, M). Cette fonction renverra un message d'erreur quand il n'existe pas de solution. Nous utiliserons l'analyse suivante :

- si la position est gagnante, i.e. si $n = 1$, nous renvoyons la liste vide ;
- sinon, nous calculons la liste $L = [C_1; C_2; \ldots; C_k]$ des coups possibles à partir de la position (n, M) et nous appliquons à L une fonction auxiliaire récursive `essai` : si L est vide, elle renvoie un message d'erreur ; sinon, elle essaie d'appliquer la fonction `solution` à la position obtenue en jouant C_1 depuis (n, M) : si cela ne renvoie pas de message d'erreur, il suffit de concaténer C_1 à la solution renvoyée. Sinon, on rattrape l'erreur en appliquant `essai` à la queue de L.

Remarque : il est possible de rattraper l'erreur `"pas de solution"` par le biais de la syntaxe :

```
try [INSTRUCTIONS 1] with Failure "pas de solution" -> [INSTRUCTIONS 2] (*)
```

Le fonctionnement est simple : si le bloc `[INSTRUCTIONS 1]` ne provoque pas l'erreur `"pas de solution"`, la ligne précédente est équivalente à `[INSTRUCTION 1]`. Sinon, elle est équivalente à `[INSTRUCTION 2]`.

5) Écrire une fonction `solution2 : position -> position -> partie` qui, appliquée à une position M_0 et à une position M_1 renvoie, si elle existe, une partie permettant de passer de la position initiale M_0 à la position finale M_1.