

DM5 : saute mouton

Thèmes

- Traits impératifs de OCaml
- Arguments en ligne de commande
- Entrées-sorties.
- Backtracking ;

Document

Dans l'archive se trouvent :

- Le sujet du DM4 (le présent document)
- Un exemple de fichier de configuration,
- Un exemple de fichier HTML de sortie comportant une mini-page de style CSS.

1 Présentation

On se propose de réaliser une recherche automatique du jeu de saute mouton par backtracking.

On considère $2n$ moutons, n noirs et n blancs.

La configuration initiale avec $n = 10$ est celle de la figure 1. On veut obtenir la configuration de la figure 2.

FIGURE 1 – Situation de départ : les moutons noirs à droite, les blancs à gauche

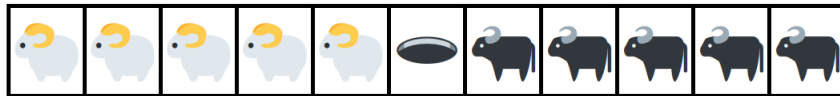
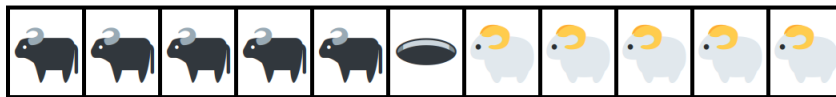


FIGURE 2 – Situation d'arrivée : les moutons blancs à droite, les noirs à gauche



On modélise le plateau du jeu par un tableau Ocaml de `cases` :

```

1 | type case = N|B|T;; (*mouton noir, mouton blanc, trou*)
2 | type plateau = case array;; (*plateau de jeu*)

```

Les règles du jeu sont les suivantes :

1. Un seul mouton se déplace à la fois ;
2. Un mouton peut avancer d'une case si celle-ci est vide ou sauter au-dessus d'un mouton de couleur opposée s'il y a une case vide derrière ;
3. Un mouton ne peut pas reculer ;
4. Un mouton ne peut pas aller plus loin que le bord droit ou le bord gauche.

2 Implantation

Q.1 Implanter `init` de type `init : int -> plateau` qui retourne la configuration initiale à n moutons blancs.

```

1 | # init(5);;
2 | - : plateau = [|B; B; B; B; B; T; N; N; N; N; N|]

```

De même, écrire `final` qui donne la configuration de sortie attendue (les noirs à gauche, les blancs à droite).

Q.2 Écrire la fonction `affichePlateau` de type `plateau -> unit` qui affiche le contenu d'un plateau de jeu. Par exemple

```

1 | # let tab = init (5) in
2 |   affichePlateau tab;;
3 |   B;B;B;B;B;T;N;N;N;N;N;
4 | - : unit = ()

```

On n'attend qu'un affichage sans fioriture à des fins de tests. Vous pourrez donner libre cours à votre créativité lors de la sortie **HTML** avec feuille de style **CSS** un peu plus bas.

Q.3 Donner la fonction `calculCandidats` de type `plateau -> (int * int) list` qui calcule quels moutons peuvent être déplacés dans le tableau `t`.

```

1 | # let tab = init (5) in calculCandidats(tab);;
2 | - : (int * int) list = [(4, 5); (6, 5)]
3 | # let (tab : plateau) = [|T;B;N|] in calculCandidats(tab);;
4 | - : (int * int) list = [(2, 0)]

```

Elle retourne une liste de couples de positions : (position du mouton, position du trou).

Cette fonction indique juste quels déplacements sont possibles mais n'effectue pas lesdits déplacements.

Q.4 Écrire la fonction `forward` de type `plateau -> int * int -> (int * int) list` telle que `forward p c h` modifie le plateau `p` en tenant compte du candidat proposé `c` et ajoute `c` à l'historique des coups joués (la liste traitée comme une pile fonctionnelle `h`). La fonction fait donc les mises à jour nécessaires pour effectuer un « pas en avant » dans le backtracking. Elle renvoie le nouvel historique des coups joués.

Voici un exemple de déroulement depuis la configuration initiale avec affichage des plateaux de jeu, deux coups joués et renvoi de l'historique.

```

1 | # let tab = init (5) in
2 | affichePlateau tab;
3 | let h = forward tab (4,5) [] in
4 | affichePlateau tab;
5 | let h' = forward tab (3,4) h in
6 | affichePlateau tab;
7 | h';;
8 | B;B;B;B;B;T;N;N;N;N;N;
9 | B;B;B;B;B;T;B;N;N;N;N;N;
10| B;B;B;T;B;B;N;N;N;N;N;
11| - : (int * int) list = [(3, 4); (4, 5)]

```

Q.5 Écrire la fonction `backward` de type `plateau -> (int * int) list -> (int * int) list`

telle que `backward p h` modifie le plateau `p` en tenant compte du sommet de l'historique des coups joués `h`. La fonction fait donc les mises à jour nécessaires pour effectuer un « pas en arrière » dans le backtracking. Elle remet le jeu dans l'état où il se trouvait avant le dernier coup joué et renvoie l'état de la pile après dépilement.

Voici un exemple de déroulement avec un pas en avant qui mène à une impasse puis un pas en arrière pour revenir à la configuration précédente :

```

1 | # let tab = [|B;T;B;N;N|] in
2 | affichePlateau tab;
3 | let h = forward tab (0,1) [(1,2)] in (*pas en avant*)
4 | affichePlateau tab; (*blocage constaté*)
5 | let h' = backward tab h in (*pas en arrière*)
6 | affichePlateau tab;
7 | h';;
8 | B;T;B;N;N;
9 | T;B;B;N;N;
10| B;T;B;N;N;
11| - : (int * int) list = [(1, 2)]

```

Q.6 Écrire la fonction récursive `solve` de type

`plateau -> plateau -> (int * int) list -> bool * (int * int) list` qui réalise le backtracking. Elle prend en paramètres le plateau de jeu dans son état courant, la configuration finale attendue et l'historique des coups joués jusqu'ici. Elle renvoie un booléen indiquant si la recherche a donné satisfaction et la pile des coups joués pour y parvenir.

Le déroulement de l'algorithme est le suivant :

- La fonction calcule les candidats pour la configuration courante.
- Tant qu'il y a un candidat à tester, on fait un pas en avant et on vérifie si ce choix donne satisfaction. Si le candidat ne permet pas d'aboutir, on fait un pas en arrière et on essaye un autre candidat.
- S'il n'y a pas de candidat où si tous ont été testés sans succès, il faut indiquer à l'appel récursif précédent que la recherche n'a rien donné.

Exemple 1 Pas de solution trouvée :

```

1 | let current = [|T;B;B;N;N|] and fin = [|N;N;T;B;B|]
2 | in solve current fin [];;
3 | T;B;B;N;N;
4 | - : bool * (int * int) list = (false, [])

```

Exemple 2 Une solution trouvée

```
1 | let b, h = let debut = init 2 and fin = final 2 in
2 | solve debut fin [];;
3 | val b : bool = true
4 | val h : (int * int) list =
5 |   [(2, 3); (3, 1); (1, 0); (0, 2); (2, 4);
6 |     (4, 3); (3, 1); (1, 2)]
```

Pour cet exemple, les différents états du plateau de jeu sont les suivants :

```
1 | B;B;T;N;N;
2 | B;T;B;N;N;
3 | T;B;B;N;N;
4 | B;N;B;T;N;
5 | B;N;T;B;N;
6 | T;N;B;B;N;
7 | N;T;B;B;N;
8 | B;N;N;B;T;
9 | B;N;N;T;B;
10 | B;N;B;N;T;
11 | B;N;T;N;B;
12 | T;N;B;N;B;
13 | N;T;B;N;B;
14 | N;N;B;T;B;
15 | N;N;T;B;B;
```

3 Sortie HTML

Pour se documenter sur **HTML** consulter w3c html et pour **CSS** aller lire w3c css.

On ne demande pas de fichier sophistiqué.

Voici donc l'aspect « fête de Noël » de ce devoir. Nous voulons récupérer en ligne de commandes le nom de deux fichiers.

- Le premier contient des consignes de configuration ;
- Le second est le fichier HTML dans lequel seront affichés les différents plateaux du jeu depuis la configuration initiale indiquée dans le premier fichier.

À partir de

- la configuration initiale ;
- des symboles de représentation des deux sortes de moutons et du trou ;
- et de l'historique des coups joués pour aboutir à la solution,

on veut afficher l'évolution du plateau de jeu dans un tableau **HTML** « élégant » avec une feuille de style minimale (ou très élaborée, c'est vous qui voyez).

Dans un fichier **config.file** écrivons maintenant quatre lignes

```
nombre de moutons : 2
mouton blanc : &#128017;
trou : &#128371;&#65039;
mouton noir : &#128017;
```

Dans ce fichier on trouve

- le nombre de moutons blancs (2 ici);
- un symbole d’emoji donné par son codeHTML pour représenter un mouton blanc;
- un autre pour représenter le trou;
- et un dernier pour un mouton noir¹.

Q.7 Écrire la fonction `infos` de type `string -> int * string * string * string`

```
1 | # infos "config.file";;
2 | - : int * string * string * string =
3 | (2, "⬛️", "⬜️⬛️⬜️⬛️", "⬛️")
```

On peut utiliser avec profit la fonction `String.split_on_char` du module `String`, pratique pour décomposer une chaîne en la découpant avant et après chaque occurrence d’un caractère donné en paramètre, et `int_of_string` pour transformer une chaîne de caractères comme `"56"` en l’entier `56`.

Q.8 Écrire la fonction `ligne` de type `plateau -> string -> string -> string -> string`

de sorte que `ligne p b n t` construit une ligne du tableau `HTML` résultat en utilisant le plateau donné en argument, le symbole de représentation des moutons blancs, celui pour les moutons noirs et celui du trou.

```
1 | # ligne (init 2) "⬛️" "⬜️" "⬛️⬜️⬛️⬜️" ;;
2 | - : string =
3 | "<tr class= \"bordure-on\">
4 | <td class= \"bordure-on\">⬛️</td>
5 | <td class= \"bordure-on\">⬛️</td>
6 | <td class= \"bordure-on\">⬛️⬜️⬛️⬜️</td>
7 | <td class= \"bordure-on\">⬜️</td>
8 | <td class= \"bordure-on\">⬜️</td>
9 | </tr>\n"
```

Nous utilisons les chaînes de caractères immutables de type `string` et leurs concaténations :

```
1 | # "toto et ^"gogo";;
2 | - : string = "toto et gogo"
3 | #
```

Il serait cependant plus judicieux d’utiliser le type `bytes` des chaînes de caractères mutables mais les formats utilisés sont assez petits pour que ce choix ne soit pas indispensable.

Remarque. Pour ma part, j’ai créé une fonction auxiliaire qui crée une ligne vide et insère donc un espace vertical entre deux plateaux. Mais c’est vous qui êtes maître du rendu final.

Q.9 Écrire la fonction `sortie` de type `string -> string -> unit` qui prend en paramètre le nom d’un fichier de configuration (comme `config.file`) et celui d’un fichier `HTML` de sortie (par exemple `sortie.html`). La fonction trouve le nombre de moutons dans le fichier de configuration et lance le jeu.

La fonction utilise ensuite l’historique des coups joués pour construire un fichier `HTML` avec sa feuille de style intégrée et dont le contenu est un tableau représentant les différentes étapes du passage de la configuration initiale à la terminale.

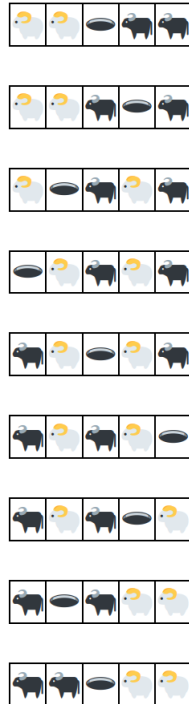
1. En fait, c’est un buffle d’eau; je n’ai pas trouvé de symbole pour le mouton noir mais vous avez le droit de faire mieux que moi. Une façon d’y parvenir est d’insérer des images plutôt que du code HTML.

Par exemple :

```
1 | # sortie "config.file" "sortie.html";;  
2 | - : unit = ()
```

produit un fichier **sortie.html** dont l'apparence est donnée figure 3.

FIGURE 3 – Exemple de déroulement du jeu pour 4 moutons en tout.



Q.10 L'ultime question de ce devoir consiste à récupérer en ligne de commandes le nom du fichier de configuration et celui du fichier de sortie.

```
$ rm sortie.html  
$ ls sortie.html  
ls: impossible d'accéder à'sortie.html': Aucun fichier ou dossier de ce type  
$ ocamlpopt -o moutons.exe moutons.ml  
$ ./moutons.exe config.file sortie.html  
$ head -n 4 sortie.html # affiche les 4 premières lignes du fichier  
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"  
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">  
  
<html>
```