Révisions

OPTION INFORMATIQUE - TP nº 4.6 - Olivier Reynet

À la fin de ce chapitre, je sais :

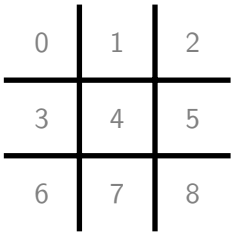
- ☞ manipuler une structure de donnée sous la forme d'un vecteur de bits
- ☞ utiliser les fonction d'itération sur les listes
- ☞ implémenter l'algorithme minimax

A Modélisation d'un jeu de morpion

On se donne les types OCaml suivants :

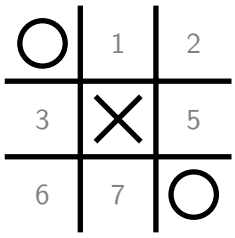
```
type ttt_player = PCross | PNought;;
type ttt_cell = Empty | Cross | Nought;;
type ttt_board = {cross : int; nought : int};;
```

Ils représentent les joueurs (croix ou rond), les cellules (vides, avec une croix ou un rond) et l'aire de jeu séparée en deux camps (croix et ronds). L'aire de jeu est numérotée de bas en haut, de 0 à huit. L'occupation du jeu est implémentée par des entiers nommés **bitboards** : chaque bit du bitboard $b_8 b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0$ représente une case occupée si le bit est à 1 et inoccupée si le bit est à zéro. Un bitboard est donc un vecteur de neuf bits et se note en hexadécimal.



$\rightarrow b_8 b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0$

■ **Exemple 1 — Exemple de modèle de partie morpion.** La partie



est représentée par `let board = {cross = 0x010; nought = 0x101}`

A1. Écrire une fonction de signature `pos_to_bb : int -> int` qui convertit un numéro représentant une case de 0 à 8 en un bitboard. Par exemple, `pos_to_bb 8 ; ;` renvoie `256`.

A2. Écrire une fonction de signature `create_bb : int list -> int` qui prend en paramètre la liste des numéros des cases occupées et qui renvoie le bitboard associé. Par exemple, `create_bb [2 ;4 ;6]` renvoie `84`.

A3. Écrire la fonction précédente à l'aide de la fonction d'itération `List.fold_left`.

A4. Écrire une fonction de signature `get_cell : ttt_board -> int -> t_cell` qui renvoie le contenu de la cellule de numéro k dans une partie. Sur la partie de l'exemple 1, l'expression `get_cell board 4` renvoie `Cross`.

B Utilitaires

B1. Écrire une fonction de signature `show : ttt_board -> unit` qui imprime la partie de morpion sur la console.

B2. Écrire une fonction de signature `bb_free_moves : ttt_board -> int` qui renvoie le bitboard représentant toutes les positions libres du jeu.

B3. Écrire une fonction de signature `free_moves : ttt_board -> int list` qui renvoie les numéros des case des positions libres du jeu sous la forme d'une liste. Par exemple, pour la partie de l'exemple 1, `free_moves board` renvoie `[7 ; 6 ; 5 ; 3 ; 2 ; 1]`.

B4. Créer la liste `winning_bb` des bitboards de tous les motifs gagnants du jeu.

B5. Écrire une fonction de complexité constante et de signature `is_winning_bb : int -> bool` qui teste si un bitboard est gagnant.

B6. Écrire une fonction de signature `winner : ttt_board -> t_player option` qui renvoie le joueur vainqueur d'une partie ou `None` s'il n'y en a pas.

B7. Écrire une fonction de signature `pos_eval : ttt_board -> int` qui évalue la qualité d'une position, c'est-à-dire la différence entre le nombre de triplets gagnants encore accessibles à `PCross` et le nombre de triplets gagnants encore accessibles à `PNought`. Cette fonction constituera l'heuristique pour l'implémentation de minimax.

C Implémentation de Minimax

C1. Écrire une fonction de signature `minimax : ttt_board -> int -> t_player -> int list -> int list * int` qui calcule la position à jouer pour un joueur d'après l'algorithme minimax. On choisira la convention suivante : `PCross` est J_{max} et `PNought` J_{min}.