Programmation : TP 12 noté (1er avril 2005)

Objectifs : [C, OCaml] Contrôle de connaissances.

L'utilisation des cours et des TPs précédents est autorisée.

Comme d'habitude, vous rendrez par mail les fichiers (`.ml` et `.c`) à `barnier@recherche.enac.fr`. Le code produit sera bien sur **aéré**, **lisible**, **commenté** et les exemples d'exécutions seront fournis en commentaire (vous n'oublierez pas de tester vos différentes fonctions avec des exemples).

[C] Morpion [10pt]

Le morpion est un jeu qui se joue à 2 en utilisant une grille carrée de taille $n * n$. Chaque joueur place, à tour de rôle, un signe lui appartenant dans une case vide de la grille (typiquement une croix ou un rond). Le premier réussissant à aligner n symboles lui appartenant (sur une ligne, une colonne ou une diagonale) gagne la partie. Si la grille est remplie sans qu'aucun des joueurs n'ait réussi à aligner n symboles, la partie est déclarée nulle.

Afin de représenter le problème, les croix, les ronds et les cases vide seront remplacés par les valeurs 1, 0 et -1 et la grille par une matrice carrée.

En utilisant la structure du programme fourni dans `/usr/local/serveur/PROG/C/morpion.c`, écrire un programme permettant de jouer au morpion.

1. Complétez et testez les différentes fonctions :
   (a) [1pt] `int** cree_matrice_carre(int)` prend la taille de la matrice en argument et retourne une matrice carrée.
   (b) [2pt] `void place_un_symbole (int**, int, int)` prend en paramètre la grille de jeu, sa taille ainsi que le symbole (ici un entier) du joueur et le place dans la grille. On vérifiera s'il est possible de placer le symbole à l'emplacement demandé.
   (c) [2pt] `void affiche_grille(int**, int)` prend en paramètre la grille de jeu ainsi que sa taille et affiche celle-ci à l'écran.
   (d) [3pt] `int est_gagnant (int**, int)` retourne `TRUE` ou `FALSE` suivant qu'une grille est gagnante ou non.
2. [2pt] Écrivez une fonction qui effectue la boucle principale et qui termine si un des joueurs gagne ou lorsque la grille est pleine.

[OCaml] [10pt]

On considère une image carrée noire et blanche divisée récursivement en quatre quarts (nord-ouest, nord-est, sud-est, sud-ouest). Celle-ci peut alors être représentée par un *quadtree* (voir figure 1).

1. [1pt] Définir le type `quadtree` permettant de représenter ce type d'image. Représenter le *quadtree* de la figure 1 avec ce type.
2. [2pt] Écrire une fonction `ratio: quadtree -> float` qui calcule la proportion de noir dans un *quadtree*. Par exemple, un *quadtree* dont les quatre fils sont noirs (donc l'image est intégralement noire) a un ratio de 1 ; le *quadtree* de la figure 1 a un ratio de 0.375.

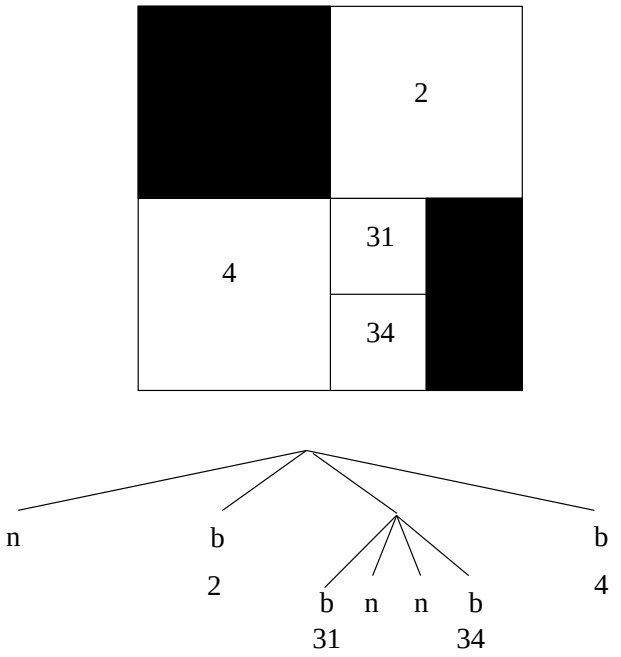

FIG. 1 – Une image et un exemple de codage possible par un *quadtree* (NB : les numéros ne font pas partie du codage)

3. [3pt] Écrire une fonction `draw: quadtree -> unit` qui dessine un *quadtree*. Utiliser la librairie `Graphics` (voir documentation en ligne). Pour compiler, rajouter `graphics.cma` dans la ligne de commande et/ou utiliser le *toplevel* en lançant `ledit ocaml graphics.cma`. Utiliser les fonctions `set_color` et `fill_rect` pour dessiner les carrés ainsi que `open_graph` (un exemple d'utilisation de ces fonctions est donné dans un fichier `/usr/local/serveur/PROG/Ocaml/exemple_graphics.ml`).
4. [2pt] Écrire une fonction `rotation: quadtree -> quadtree` qui fait pivoter un *quadtree* de 90 degrés dans le sens des aiguilles d'une montre.
5. [2pt] Écrire une fonction `intersection : quadtree -> quadtree -> quadtree` qui renvoie l'intersection de deux *quadtrees*. Un exemple d'intersection est donné figure 2.

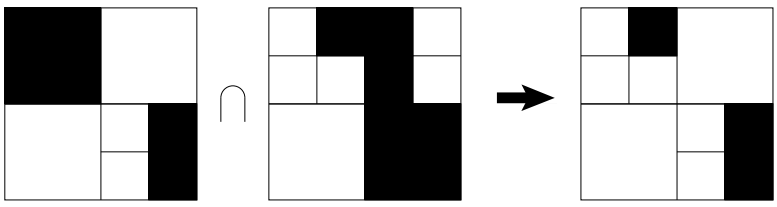

FIG. 2 – Un exemple d'intersection