Programmation : Correction TP 12 noté (S04, 1er avril 2005)

[C] Morpion

1. Écriture des fonctions prédéfinies :

   (a) Fonction de création et d'initialisation à vide d'une matrice carrée :

```
int** cree_matrice(int taille) {
  int i,j;
  int** mat = (int **) malloc(taille*sizeof(int*));
  for (i=0; i < taille; i++) {
    mat[i] = (int *) malloc(taille*sizeof(int));
    for (j=0; j < taille; j++)
      mat[i][j] = VIDE;
  }
  return mat;
}
```

   (b) Fonction qui place un symbole dans la grille. Si la case est déjà utilisée ou si les coordonnées entrées au clavier par l'utilisateur sont fausses, on rappelle récursivement la fonction :

```
void place_un_symbole(int** mat, int taille, int joueur) {
  /* prend le numero de la ligne et de la colonne en ligne de commande */
  int ligne, colonne;
  printf("Numeros de ligne et de colonne separes par un espace: ");
  scanf("%d %d", &ligne, &colonne);

  if (0 <= ligne && ligne < taille &&
      0 <= colonne && colonne < taille &&
      mat[ligne][colonne] == VIDE)
    mat[ligne][colonne] = joueur;
  else {
    printf("Coup non valide. Recommence.\n");
    place_un_symbole(mat, taille, joueur);
  }
}
```

   (c) Fonction d'affichage de la grille de jeu :

```
void affiche_grille(int** mat, int taille) {
  int i,j;
  for (i=0; i < taille; i++) {
    for (j=0; j < taille; j++) {
      char c = (mat[i][j] == CROIX ? 'x' : (mat[i][j] == ROND ? 'o' : ' '));
      printf("%c", c);
    }
    printf("\n");
  }
}
```

(d) Fonction retournant `TRUE` si la grille est gagnante, `FALSE` sinon. Un grille est gagnante quand la valeur absolue de la somme des cases d'une ligne, d'une colonne ou d'un diagonale est égale à la taille de la matrice :

```
int est_gagnant (int** mat, int taille) {
  int i,j;
  int res_diag1 = 0;
  int res_diag2 = 0;
  /* test sur les lignes et colonnes */
  for (i=0; i < taille; i++) {
    int res_ligne = 0;
    int res_col = 0 ;
    for (j=0; j < taille; j++) {
      res_ligne = res_ligne + mat[i][j];
      res_col = res_col + mat[j][i];
    }
    if ((abs(res_ligne) == taille) || (abs(res_col) == taille))
      return TRUE;
  }
  /* test sur les diagonales */
  for (i=0; i < taille; i++) {
    res_diag1 = res_diag1 + mat[i][i];
    res_diag2 = res_diag2 + mat[i][taille-i-1];
  }
  if ((abs(res_diag1) == taille) || (abs(res_diag2) == taille))
    return TRUE;

  /* Si tout a echoue, la grille n'est pas gagnante */
  return FALSE;
}
```

2. Fonction principale. On boucle tant que personne n'a gagné et tant que la grille n'est pas pleine (le nombre maximum de coups à jouer est le nombre de cases de la grille) :

```
void play(int t){
  int **m = cree_matrice(t);
  int player = CROIX, nb = 0;

  while (!est_gagnant(m, t) && nb < t*t){
    affiche_grille(m, t);
    place_un_symbole(m, t, player);
    /* changement de joueur */
    player = player * (-1);
    nb++;
  }
  affiche_grille(m, t);
}

int main (int argc, char** argv) {
  int n = atoi(argv[1]);
  play(n);
  return 0;
}
```

[OCaml] *Quadtrees*

1. Définition du type `quadtree`. Un *quadtree* est soit une image noire, soit une image blanche, soit est composé de quatre autres *quadtrees*.

```
#type quadtree = N | B | Q of quadtree*quadtree*quadtree*quadtree (* NE, SE, SW, NW *);;

#let q = Q(N,B,Q(B,N,N,B),B);;
```

2. Fonction qui calcule la proportion de noir dans un *quadtree* :

```
#let rec ratio = fun q ->
#  match q with
#    N -> 1.
#  | B -> 0.
#  | Q (a,b,c,d) -> (ratio a +. ratio b +. ratio c +. ratio d) /. 4.;;
val ratio : quadtree -> float = <fun>
```

3. Fonction qui dessine un *quadtree*.

```
##load "graphics.cma";;

#let draw = fun q ->
#  let rec draw_rec = fun q x y n ->
#    match q with
#      N -> Graphics.set_color Graphics.black; Graphics.fill_rect x y n n
#    | B -> Graphics.set_color Graphics.white; Graphics.fill_rect x y n n
#    | Q (ne, se, sw, nw) ->
#        let n2 = n / 2 in
#        draw_rec ne (x+n2) (y+n2) n2;
#        draw_rec se (x+n2) y n2;
#        draw_rec sw x y n2;
#        draw_rec nw x (y+n2) n2 in
#  Graphics.open_graph " 1024x1024";
#  draw_rec q 0 0 1024;;
val draw : quadtree -> unit = <fun>
```

4. Fonction qui fait pivoter un *quadtree* de 90 degrés dans le sens des aiguilles d'une montre :

```
#let rec rotation = fun q ->
#  match q with
#    N | B -> q
#  | Q (a, b, c, d) -> Q (rotation d, rotation a, rotation b, rotation c);;
val rotation : quadtree -> quadtree = <fun>
```

5. Fonction qui renvoie l'intersection de deux *quadtrees* :

```
#let rec intersection = fun q1 q2 ->
#  match q1, q2 with
#    N, q | q, N -> q
#  | Q (a,b,c,d), Q (a',b',c',d') ->
#     Q (intersection a a', intersection b b', intersection c c', intersection d d')
#  | _ -> B;;
val intersection : quadtree -> quadtree -> quadtree = <fun>
```