Université Paris Diderot
L3 Informatique

Programmation Fonctionnelle
Année 2016-2017

TP n°5 - Correction

Unit, exceptions et entrées/sorties

Le but de ce TP est de mettre en ouvre le jeu du *morpion* (aussi connu comme tic-tac-toe) :

Vous pouvez trouver les règles du jeu sur wikipedia : `http://fr.wikipedia.org/wiki/Tic-tac-toe`

Quelques fonctions prédéfinies

Dans les exercices qui suivent, vous pourrez utiliser les fonctions prédéfinies suivantes pour réaliser des affichages à l'écran :

- `print_string : string -> unit` pour afficher une chaîne de caractères ;
- `print_int : int -> unit` pour afficher un entier ;
- `print_newline : unit -> unit` pour passer à la ligne – cette fonction prend en argument la constante `()` de type `unit`, un appel de cette fonction s'écrit donc `print_newline ()`.
- `read_int: unit -> int` pour lire un entier
- `read_line: unit -> string` pour lire une chaîne de caractères

La fonction suivante sera également utile pour manipuler des chaînes de caractères :

- `String.get : string -> int -> char` retourne le caractère en position `n` dans une chaine de caractère `s`.

1 Définition des types et a□chage de leur valeurs

Exercice 1 (definition des types)**.** Définir les types suivantes :

- `joueur` : qui est soit une croix, soit un rond
- `case` : qui est soit un joueur, soit un symbole représentant la case vide
- `ligne` : un triplet de cases
- `grille` : un triplet de lignes

Correction :

```
type joueur = Croix | Rond
type case = P of joueur | Vide
type ligne = case*case*case
type grille = ligne*ligne*ligne ;;
```

Exercice 2 (affichage d'une grille)**.** Définir la grille `init` initiale, où toutes les cases sont vides. Définir les fonctions (en faisant attention à ce qu'elles aient le type indiqué) :

- `stringCase: case -> string` qui prend en entrée une case et renvoie une chaîne de caractères qui lui correspond (`X` (resp. `O` ou espace) si la case est une croix (resp. un rond ou vide)),
- `afficheCase : case -> unit` qui affiche la chaîne précédente,

— `stringLigne : ligne -> string` qui prend en entrée une ligne et renvoie la chaîne correspondante,
— `afficheLigne : ligne -> unit` qui affiche la chaîne précédente,
— `afficheGrille : grille -> unit` qui affiche la grille donnée en entrée. La figure 1 ci-dessous donne des examples de grilles affichées.

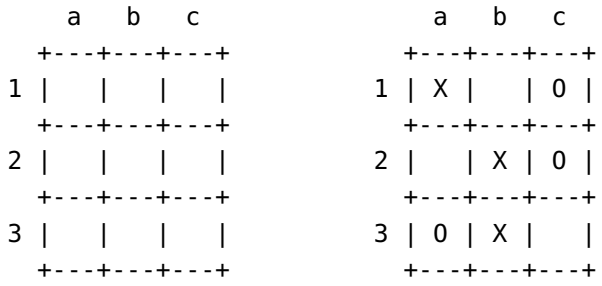

Figure 1 – Exemples d'affichage à l'écran d'une grille.

Correction :

```
(* grille initiale *)
let init = ((Vide,Vide,Vide),(Vide,Vide,Vide),(Vide,Vide,Vide));;
(* Fonction qui renvoie une chaine correspondant a une case *)
let stringCase = function
      P Croix -> "X"
    | P Rond -> "O"
    | Vide -> " " ;;

(* Fonction qui affiche une case *)
let afficheCase x =
    print_string (stringCase x);;

(* Fonction qui renvoie une chaine corrspondant a une ligne *)
let stringLigne = function
   (p1,p2,p3) -> "| "^(stringCase p1)^" | "^(stringCase p2)^
                 " | "^(stringCase p3)^" |\n";;

(* Fonction qui affiche une ligne *)
let afficheLigne l = print_string (stringLigne l);;

(* Fonction qui affiche la grille *)
let afficheGrille = fun (l1, l2, l3) ->
    print_string "\n     a   b   c\n" ;
    print_string "   +---+---+---+\n" ;
    print_string " 1 " ;
        afficheLigne l1;
    print_string "   +---+---+---+\n" ;
    print_string " 2 " ;
        afficheLigne l2;
        print_string "   +---+---+---+\n" ;
    print_string " 3 " ;
        afficheLigne l3;
    print_string "   +---+---+---+\n" ;;
```

Fonctions pour le roulement du jeu

Exercice 3. Définir les exceptions `MoveForbidden` et `NoMoreMove`.
Correction :

```
(* Les exceptions *)
exception MoveForbidden;;
exception NoMoreMove;;
```

Exercice 4. Définir la fonction `move : int * char -> grille -> joueur -> grille` pour placer dans une case de la grille le pion associé à un jouer. Cette function doit lever l'exception `MoveForbidden` si la case à changer n'est pas vide, ou si la pair `int * char` donnée en entrée ne corresponde pas à une case de la grille.
Correction :

```
(* Fonction pour changer une case de la grille *)
let move (l,c) ((l1,l2,l3): grille) j : grille =
    let move_ligne c (p1,p2,p3) j =
        match c with
        | 'a' -> if (p1=Vide) then (P j, p2, p3) else raise  MoveForbidden
        | 'b' -> if (p2=Vide) then (p1, P j, p3) else raise  MoveForbidden
        | 'c' -> if (p3=Vide) then (p1, p2, P j) else raise  MoveForbidden
        | c -> raise  MoveForbidden
    in match l with
    | 1 -> ((move_ligne c l1 j),l2,l3)
  | 2 -> (l1, (move_ligne c l2 j),l3)
    | 3 -> (l1,l2, (move_ligne c l3 j))
    | n -> raise MoveForbidden;;
```

Exercice 5. Définir une fonction `aVide: grille -> bool` pour tester si une grille a au moins une case vide. Définir une fonction `gagne : grille -> case` qui prend en entrée une grille et renvoie :

- la case associée à un joueur si la grille est gagnante pour ce joueur,
- ou alors la case vide, s'il reste encore une case vide dans la grille,
- ou alors elle lève l'exception `NoMoreMove`.

Correction :

```
(* Fonction pour tester un Vide *)
let aVide (l1,l2,l3) =
    let aVide_ligne (p1,p2,p3) = (p1=Vide) ||(p2=Vide) || (p3=Vide) in
    aVide_ligne l1 || aVide_ligne l2 || aVide_ligne l3
;;

(* Fonction qui verifie si un joueur a gagne *)
let gagne (((p1,p2,p3),(p4,p5,p6),(p7,p8,p9)):grille) =
  match p1 with
  | P j when (p1=p2 && p2=p3) || (p1=p4 && p4=p7) || (p1=p5 && p5=p9) -> P j
  | _ ->
    match p2 with
    | P j when (p2=p5 && p5=p8) -> P j
    | _ ->
      match p3 with
      | P j when (p3=p6 && p6=p9) || (p3=p5&&p5=p7) -> P j
      | _ ->
    match p4 with
    | P j when p4=p5 && p5=p6 -> P j
    | _ ->
```

```
match p7 with
| P j when p7=p8 && p8=p9 -> P j
| _ ->  if aVide ((p1,p2,p3),(p4,p5,p6),(p7,p8,p9)) then Vide
        else raise NoMoreMove
```

```
        begin
            print_string "Ligne ? ";
            let l=read_int () in
            print_string "Colonne ? ";
            let c=(read_line ()).[0] in
            move (l,c) g j
        end
    with
    | MoveForbidden ->
    (print_endline "Le choix est interdite ! Merci de la refaire\n" ; read_coordonees g j
        )
;;

let tour g j =
    afficheGrille g;
    print_string "Bonjour "; afficheCase (P j) ;print_newline ();
    read_coordonees g j
```

Exercice 7. Écrire la function `morpion: grille -> joueur -> unit` qui étant donné une grille `g` et un joueur `j` commence le jeu sur la grille `g` avec le joueur `j`. Ensuite elle continue, en alternant les coups des deux joueurs et en affichant la grille à chaque tour, jusqu'à ce que soit un joueur gagne, soit il n'y a plus de case à remplir.

Correction :

```
(*negation joueur*)
let autre j = if j=Croix then Rond else Croix

(*fonction principale*)
let rec morpion g j=
    try
        begin
            let x = gagne g in
            match x with
            | P winner -> print_string "Bravo : "; afficheCase (P winner) ; print_endline
                 " a gagne !"
            | Vide -> let g_next = (tour g j) in morpion g_next (autre j)
        end
    with
    | NoMoreMove -> print_endline ("Aucun gagnant, desole !")
;;
```

Exercice 8. Mettre toutes les fonctions ainsi définies dans un fichier `morpion.ml`. Completer le fichier pour que l'exécution du code compilé exécute le jeu du morpion à partir de la grille initiale et du joueur associé à la croix. Tester le code.

Correction :

```
morpion init Croix;;
```

Exercice 9. Modifier le programme de sorte qu'on puisse sauvegarder une partie et la recharger, si on veut. On suppose qu'on peut sauvegarder une seule partie. On peut donc choisir un nom de fichier unique pour cela. Il faut définir un format de sauvegarde. On pourrait utiliser les fonctions `stringCase` et `stringLigne` définies précédemment.

Correction :

```
let rec sauvegarder j ((l1,l2,l3):grille) =
  let c = open_out "partie" in
```

```
  let jc = if j = Croix then "X\n" else "O\n" in
  output_string c jc;
  output_string c (stringLigne l1);
  output_string c (stringLigne l2);
  output_string c (stringLigne l3);
  close_out c;;

let convertligne s =
  let convertchar c = match c with
     'O' -> P Rond
    |'X' -> P Croix
    | _ -> Vide
  in (convertchar s.[2], convertchar s.[6], convertchar s.[10]);;

let charger nom =
  let c = open_in nom in
  let jc = input_line c and
      s1 = input_line c and
      s2 = input_line c and
      s3 = input_line c in
  let j = if (jc).[0] = 'X' then Croix else Rond in
  close_in c; let g = (convertligne s1, convertligne s2, convertligne s3)
          in afficheGrille g; (j,g)
  ;;

let rec choix j g =
       print_string "Vous voulez sauvegarder (s) la partie courante ou recharger (r) la
           partie sauvegarde'e ou continuer (c) ? ";
       match (read_line ()).[0] with
      's' -> sauvegarder j g; (j,g)
        | 'r' -> charger "partie"
        | _ -> (j,g);;

let rec morpion2 g j=
    try
      begin
        afficheGrille g;
        let x = gagne g in
        match x with
        | P winner -> print_string "Bravo : "; afficheCase (P winner) ; print_endline " a
             gagne !"
        | Vide -> let (j1,g1) = choix j g in let g_next = (tour g1 j1) in morpion2 g_next
             (autre j1)
      end
    with
    | NoMoreMove -> print_endline ("Aucun gagnant, desole !")
;;

morpion2 init Croix;;
```

Exercice 10. Modifier le programme de sorte qu'il puisse traiter les exceptions soulevées par les functions de lecture au clavier (une exception est soulevée quand on entre un caractère au lieu d'un entier après un `read_int`).

Exercice 11. (*?*) Modifier le programme en rendant un joueur automatique et qui ne perd jamais.