Université Paris Diderot
L3 Informatique

Programmation Fonctionnelle
Année 2016-2017

TP n°5

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 affichage 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

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.

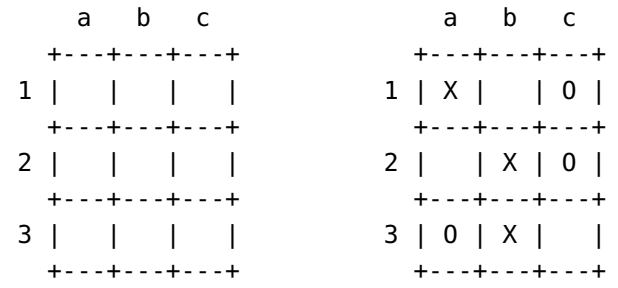

FIGURE 1 – Exemples d'affichage à l'écran d'une grille.

Fonctions pour le roulement du jeu

Exercice 3. Définir les exceptions `MoveForbidden` et `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.

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`.

Interaction avec l'utilisateur

Exercice 6. Définir une fonction `tour: grille -> joueur -> grille` qui prend en entrée une grille `g` et un joueur `j`, affiche d'abord la grille `g` à l'écran, puis demande à l'utilisateur de rentrer une ligne et une colonne pour mettre le pion associé au joueur `j`. La fonction renvoie enfin la grille modifiée en consequence. La fonction doit traiter l'exception `MoveForbidden` et demander à l'utilisateur de rentrer la ligne et la colonne.

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.

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.

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.

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. ($\star$) Modifier le programme en rendant un joueur automatique et qui ne perd jamais.