DS 2 du 5 avril 2023 - 2 heures

Calculatrice, téléphone et documents interdits

Les programmes non commentés ou dont le fonctionnement n'est pas expliqué ne seront pas relus. Une présentation raisonnablement aérée (avec une identation convenable des structures de choix ou de répétition), ainsi que des résultats soulignés ou encadrés, sont des éléments de l'évaluation... C'est vous qui voyez...

Exercice 1

1. Écrire une fonction récursive `colle : 'a list -> 'a list list -> 'a list list` qui associe à une liste d'éléments (l'alphabet) et à une liste de listes constituées d'éléments de l'alphabet, la liste de listes obtenues en ajoutant en tête l'un des éléments de l'alphabet. On souhaite obtenir par exemple

```
# colle [1;2;3;4] [[5;6];[7];[]];;
- : int list list = [[1; 5; 6]; [1; 7]; [1];
[2; 5; 6]; [2; 7]; [2]; [3; 5; 6]; [3; 7]; [3];
[4; 5; 6]; [4; 7]; [4]]
```

2. Écrire une fonction récursive `mots : 'a list -> int -> 'a list list` qui à une liste `alphabet` et un entier `n`, associe la liste de listes correspondant à tous les mots de longueur `n` construits avec les éléments de l'alphabet. On souhaite obtenir par exemple :

```
mots [1;2] 4;;
- : int list list = [[1; 1; 1; 1]; [1; 1; 1; 2];
       [1; 1; 2; 1]; [1; 1; 2; 2]; [1; 2; 1; 1];
       [1; 2; 1; 2]; [1; 2; 2; 1]; [1; 2; 2; 2];
       [2; 1; 1; 1]; [2; 1; 1; 2]; [2; 1; 2; 1];
       [2; 1; 2; 2]; [2; 2; 1; 1]; [2; 2; 1; 2];
       [2; 2; 2; 1]; [2; 2; 2; 2]]
```

Exercice 2 [Connecteur de Sheffer] Le connecteur de Sheffer `nand`, noté $|$, est le connecteur binaire de la logique propositionnelle défini par $\boxed{F|G = \neg F \vee \neg G}$ où F et G sont des formules propositionnelles.
On considère des formules propositionnelles construites avec des variables propositionnelles $a, b, c, \ldots$, les connecteurs binaires $\vee$(disjonction), $\wedge$(conjonction), $\rightarrow$(implication), le connecteur unaire de négation et le connecteur de Sheffer.

L'objet de l'exercice est de montrer que pour toute formule F de la logique propositionnelle, il existe une formule F^*, ne contenant que des variables propositionnelles et le connecteur de Sheffer, telle que F et F^* soient logiquement équivalentes

1. Construisez la table de vérité d'une formule $x|y$, puis de $x|x$.
2. Montrer que l'on peut exprimer $\neg x$ et $x \vee y$ à l'aide du connecteur de Sheffer.
3. Montrer qu'il existe une formule sémantiquement équivalente à $x \rightarrow y$ ne contenant que les variables propositionnelles x et y et deux occurrences du connecteur de Sheffer.
4. De même, montrez qu'il existe une formule équivalente à $x \wedge y$ ne contenant que les variables propositionnelles x et y et des occurrences du connecteur de Sheffer.
5. Des questions précédentes, déduire un algorithme permettant de transformer toute formule F de la logique propositionnelle en une formule F^* ne contenant que des occurrences du connecteur de Sheffer et des variables propositionnelles présentes dans F. Appliquer cet algorithme à la formule $x \wedge (y \vee z)$
6. On note $\sigma(F)$, la taille de F. *On dit qu'une formule F est équilibrée lorsqu'elle est sans connecteur de négation et qu'elle vérifie :*
   - *Si F est une variable propositionnelle, elle est équilibrée ;*
   - *Si $F = G \star H$ avec $\star \in \{\vee, \wedge, \rightarrow, |\}$ alors G et H sont équilibrées et $\sigma(G) = \sigma(H)$.*

Montrer que si F est équilibrée alors il existe $k \in \mathbb{N}$ tel que $\sigma(G) = 2^k - 1$ et préciser ce que représente k ?

7. On note $\sigma^*(n)$ la taille au pire de la transformée F^* d'une formule F équilibrée de taille n.
   a. Justifier que $\sigma^*(0) = 0$ et que $\forall n \in \mathbb{N}^*,\ \sigma^*(n) = 4\sigma^*(n-1) + 3$
   b. En déduire $\sigma^*(n)$ pour tout entier naturel n.

Exercice 3 [Roller Splat]

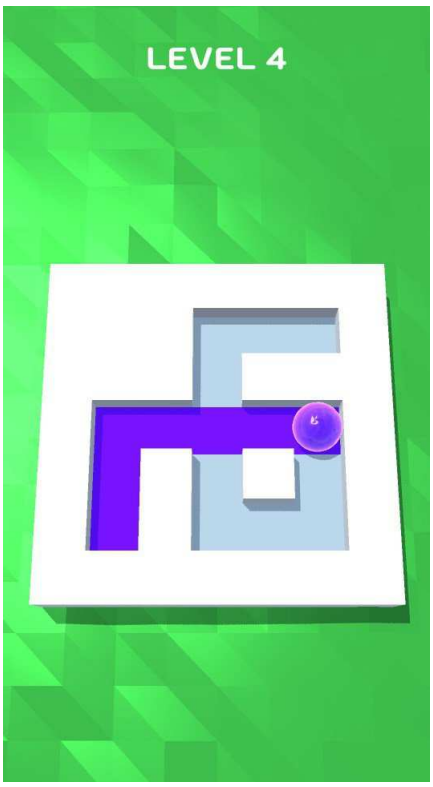
Le but du jeu Roller Splat est de repeindre un plateau de jeu à l'aide d'une bille de peinture. Lorsque l'on choisit une direction

0 : Haut, 1 : Droite, 2 : Bas, 3 : Gauche,

la bille avance jusqu'à toucher un mur. Dans cet exercice, on modélise le plateau de jeu par une matrice (tableau de tableaux) avec la convention suivante :

0 case vide, 1 : mur, 2 : case peinte, 3 : case où se trouve la bille.

Par exemple, le plateau ci-contre est modélisé par la déclaration :




```
let plateau = [|
      [|1; 1; 1; 1; 1; 1; 1|];
      [|1; 1; 1; 0; 0; 0; 1|];
      [|1; 1; 1; 0; 1; 1; 1|];
      [|1; 2; 2; 2; 2; 3; 1|];
      [|1; 2; 1; 0; 1; 0; 1|];
      [|1; 2; 1; 0; 0; 0; 1|];
      [|1; 1; 1; 1; 1; 1; 1|] |];;
```

A noter enfin que l'on peut repasser sur une case peinte et que le pourtour du plateau de jeu est toujours entouré de murs.

1. Écrire une fonction `dim : 'a array array -> int * int` qui reçoit une matrice et renvoie le couple (l, c) correspondant au nombre de lignes et de colonnes de ce tableau.
2. Écrire une fonction `gagne : int array array -> bool` qui reçoit une matrice et renvoie un booléen indiquant si la partie est gagnée ou non.
3. Écrire une fonction `case : int array array -> int * int` qui reçoit une matrice et renvoie un couple de la forme (ligne, colonne) indiquant où la bille se peinture se trouve.
4. Pour simplifier la suite du code, on souhaite réaliser la table de correspondance suivante :

Français	Numérique	Vecteur $(\Delta_{ligne}, \Delta_{colonne})$
Haut	0	$(-1, 0)$
Droite	1	$(0, 1)$
Bas	2	$(1, 0)$
Gauche	3	$(0, -1)$

   a. Écrire une fonction `dir2num : int * int -> int` qui converti un couple de direction en la valeur numérique correspondante de $\{0, 1, 2, 3\}$ en **utilisant un filtrage**.
   b. Écrire une fonction `num2dir : int -> int * int` qui produit l'effet inverse, on pourra utiliser une table de correspondance à l'aide d'un tableau de couples d'entiers ne manière à n'utiliser ni `if`, ni filtrage. On appelle ce tableau **une table de hachage**.

5. Écrire une fonction `depPossible : int array array -> int * int -> int -> bool` qui reçoit la matrice du plateau de jeu, une case sous la forme d'un couple (ligne, colonne) ainsi qu'une direction de l'ensemble $\{0, 1, 2, 3\}$ et renvoie un booléen, indiquant si le déplacement depuis la case donnée et dans la direction donnée est possible (i.e : il n'y a pas de mur).
6. Écrire une fonction `nouvelle : int array array -> int -> unit` qui reçoit la matrice représentant le plateau et la direction souhaitée et modifie le plateau. Cette fonction est une fonction d'impression, elle modifie le plateau, mais ne renvoie "rien".

Correction 1

Correction 3

1. en utilisant `length` :

```
let dim p = Array.length p, Array.length p.(0);;
```

2. On parcours tout le tableau, si on trouve un 0, c'est que ce n'est pas gagné :

```
let gagne p = let g = ref true and l,c = dim p in
   for i = 0 to l-1 do
      for j = 0 to c-1 do
         if p.(i).(j) = 0 then g := false
      done
   done;
   !g
;;
```

3. Même principe pour trouver la balle, mais on cherche un 3 :

```
let case p = let rep = ref (0,0) and l,c = dim p in
   for i = 0 to l-1 do
      for j = 0 to c-1 do
         if p.(i).(j) = 3 then rep:= (i,j)
      done
   done;
   !rep
;;
```

4. Les 2 fonctions :

```
let dir2num = function
        | (-1,0) -> 0
        | (0,1) -> 1
        | (1,0) -> 2
        | _ -> 3
;;

let num2dir n = let table = [|(-1,0);(0,1);(1,0);(0,-1)|] in table.(n);;
```

5. on teste la présence d'un mur :

```
let depPossible p case d =
        let l,c = case and dl, dc = num2dir d in
                p.(l+dl).(c+dc) <> 1
;;
```

6. On avance jusqu'à toucher un mur

```
let nouvelle p d =
        let c = ref (case p) and dl, dc = num2dir d in
        while depPossible p !c d do
                p.(fst !c).(snd !c) <- 2;
                c:= fst !c + dl, snd !c + dc
        done;
        p.(fst !c).(snd !c) <- 3
;;
```