université
PARIS-SACLAY

Introduction à la programmation fonctionnelle
L2 Informatique

Devoir à la maison

1 Modalités

Le devoir est **individuel**. Il est à rendre au plus tard :

le lundi 11 janvier 2021, 17h00

Le seul fichier à rendre est le fichier **yams.ml** que vous aurez complété avec votre solution. Les autres fichiers ne doivent êtres ni modifiés ni rendus. Le dépôt se fera uniquement sur l'espace eCampus du cours, dans l'activité de rendu prévue à cet effet. Le devoir peut être déposé sous forme de « brouillon », mais le dépôt final devra être validé avant la date limite. Attention, **une fois validé**, votre soumission en pourra plus être modifiée.

Le plagiat sera sévèrement sanctionné, conformément au règlement des études premier cycle de l'Université. Les questions génériques (de compréhension de l'énoncé, ou de problème pour utiliser le code) peuvent être posée sur le forum du cours, sur la page eCampus (afin que les réponses puissent profiter à tout le monde).

Les questions personnelles sur votre code, en particulier montrant des exemples de votre code, doivent être posée individuellement par email (`kn@lri.fr`) afin de ne pas influencer les autres étudiants.

La suite de l'énoncé se décompose en trois parties. La section 2 présente le cadre général du sujet (le jeu de Yams) et les règles du jeu. La section 3 explique la structure des fichiers fournis et vous donne toutes les commandes pour tester votre programme facilement et le comparer à la solution donnée dans l'archive. Enfin, la section 4 contient les questions.

Le projet a été conçu pour être fait sous l'interface Jupiterhub utilisée en TP (en particulier, le programme de référence et le corrigé ont été compilés sous cet environnement). Si, pour des raisons impérieuses, vous ne pouvez pas utiliser l'environnement Jupiter, vous pouvez contacter **kn@lri.fr** pour obtenir une version compilée du corrigé pour votre plateforme de travail.

2 Le jeu de Yam's

Le but du devoir est de coder des fonctions utilitaires permettant de jouer à une variante du jeu de Yam's pour un joueur. Ce jeu (aussi appelé Yahtzee) est un jeu de hasard raisonné utilisant 5 dés à 6 faces [1]

2.1 Description du jeu

Dans la variante à un joueur, ce dernier dispose de 13 tours pour faire un maximum de points. À chaque tour, le joueur peu lancer un certain nombre de dés pour former des combinaisons. Les différentes combinaisons, inspirées du poker, permettent de gagner des points. Au premier lancé, le joueur lance les 5 dés. Au second lancé, il peut choisir de ne relancer qu'un certain nombre de dés et conserver les autres. Il peut procéder ensuite à un troisième lancé sur le même principe. Après ce troisième lancé, on regarde si les dés forment une des combinaisons gagnantes et ont inscrit alors un certain nombre de points correspondants. Une fois une combinaison trouvé (par exemple le carré, avec 4 dés identiques), elle ne peut plus être utilisée pour marquer des points. Pour marquer un maximum de points, il faut donc essayer de faire toutes les combinaisons possibles.

1. Un jeu de hasard raisonné est un jeu dans le quel le hasard intervient mais n'est pas le seul élément permettant la victoire. De nombreux jeux rentre dans cette catégorie : Poker, Tarot (et tout jeu dans lequel des cartes sont battues et distribuées aléatoirement), Monopoly (ou tout jeu utilisant des dés mais nécessitant une stratégie). Ils sont différents des jeux de hasard pur (jeu de l'oie, roulette) et des jeux sans hasard (échecs, go, …).

2.2 Grille de points

La grille de points se présente en deux partie, la partie supérieure et la partie inférieure :

Partie supérieure :

	nombre de points
total de 1 (total de 1)	
total de 2 (2× somme des 2)	
total de 3 (3× somme des 3)	
total de 4 (4× somme des 4)	
total de 5 (5× somme des 5)	
total de 6 (6× somme des 6)	
Bonus $\geq$ 62 (35)	
Total supérieur	

Partie inférieure :

	nombre de points
Chance (somme des dés)	
Brelan (somme des dés)	
Carré (somme des dés)	
Full House (25)	
Petite Suite (30)	
Grande Suite (40)	
Yam's (50)	
Total inférieur	

Total	

Avant de présenter la façon de compter les points, on détaille les combinaisons de la grille inférieure :

brelan : trois dés identiques, par exemple ⚂ ⚃ ⚂ ⚂ ⚅.

carré : quatres dés identiques, par exemple ⚄ ⚄ ⚄ ⚄ ⚅.

full house : trois dés identiques et deux autres identiques (et distincts), par exemple : ⚀ ⚀ ⚀ ⚇ ⚇. Attention, les cinq dés ne peuvent pas être identiques.

petite suite : quatres dés qui se suivent, par exemple : ⚂ ⚁ ⚃ ⚅ ⚄ (ici, les dés 1,2,3,4 sont présents).

grande suite : cinq dés qui se suivent, par exemple : ⚂ ⚁ ⚃ ⚇ ⚄ (ici, les dés 1,2,3,4,5 sont présents).

yam's : cinq dés identiques, par exemple ⚀ ⚀ ⚀ ⚀ ⚀.

chance : n'importe quels dés.

2.3 Déroulement d'un tour et comptage des points

Supposons que nous soyons sur une grille vierge (premier tour).

— Premier lancé : ⚀ ⚂ ⚀ ⚂ ⚃. On obtient deux paires. On choisit de conserver tous les dés sauf le dernier.

— Deuxième lancé ⚄. On choisit de relancer encore ce dé.

— Troisième lancé ⚀.

Les dés que l'on considère pour ce tour sont : ⚀ ⚂ ⚀ ⚂ ⚀. On calcule donc **toutes les combinaisons valides pour la grille supérieure et pour la grille inférieure**. On conserve ensuite pour chaque grille la combinaison la plus forte et on inscrit les points [2]. Ici :

Grille supérieure : on peut compter :

— trois ⚀ : 3 points

— deux ⚂ : $2 \times (2 + 2) = 8$ points

On inscrit donc 8 points dans la deuxième ligne de la grille supérieure, le total supérieur est maintenant de 8 points.

Grille inférieure : on peut compter :

2. Cette façon de compter les points n'est pas tout à fait standard, mais rend le code plus simple

— brelan ⚀ ⚀ ⚀ : somme de tous les dés : $1+1+1+2+2=7$ points.
— full house ⚀ ⚀ ⚀ ⚄ ⚄ 25 points.

On incrit donc 25 points dans la case full house de la grille inférieure et le total inférieur est maintenant de 25 points.

Le premier tour est terminé, on peut maintenant passer au second tour.

Supposons qu'après les trois lancés du second tour, on obtienne : ⚄ ⚄ ⚄ ⚅ ⚅. Même si se motif est un full, les points ont déjà été marqués pour cette combinaison, elle n'est donc plus accessible. On peut cependant marquer les points grâce à un brelan (on a trois ⚄). Le total pour un brelan est la somme des dés donc $2+2+2+3+3=24$ points. De même pour la grill supérieure, on a déjà rempli la case des 2. On ne peut donc pas marquer $2\times(2+2+2)=12$ points, mais on peut marquer $3\times(3+3)=18$ points. La case des trois sera alors inaccessible par la suite.

Enfin, supposons qu'au tour trois on obtienne ⚀ ⚀ ⚃ ⚃. Pour la partie supérieure les points possibles sont :
— $1+1+1=3$ points
— $4\times 4=16$ points
— $5\times 5=25$ points

On marque donc 25 points (et on condamne la case des 5).

Pour la partie inférieure, le brelan n'est plus disponible. On peut utiliser la case chance et marquer la somme des dés : $1+1+1+4+5=12$ points. À l'issu du troisième tour, la grille est donc :

Partie supérieure :

	nombre de points	
total de 1 (total de 1)		
total de 2 (2× somme des 2)	8	(obtenu au tour 1)
total de 3 (3× somme des 3)	18	(obtenu au tour 2)
total de 4 (4× somme des 4)		
total de 5 (5× somme des 5)	25	(obtenu au tour 3)
total de 6 (6× somme des 6)		
Bonus ≥ 62 (35)		
Total supérieur		

Partie inférieure :

	nombre de points	
Chance (somme des dés)	12	(obtenu au tour 3)
Brelan (somme des dés)	24	(obtenu au tour 2)
Carré (somme des dés)		
Full House (25)	25	(obtenu au tour 1)
Petite Suite (30)		
Grande Suite (40)		
Yam's (50)		
Total inférieur		

Total	

Si à un moment le total de la partie supérieure atteint ou dépasse 62 points, alors on marque en plus un bonus de 35 points.

La partie se termine après 13 tours. On remarque qu'après les premiers tours il est de plus en plus difficile de marquer des points.

3 Le programme

Le programme ainsi que les fichiers à compléter se trouve dans une archive : `ipf_l2_info_devoir.tar.gz` disponible sur la page du cours. Vous devez :
— télécharger cette archive sur votre ordinateur **sans la décompresser**
— déposer cette archive telle qu'elle dans votre espace jupyterhub
— ouvrir un Terminal jupyterhub, vous placer dans le même répertoire que l'archive et exécuter la commande :
`$ tar xzvf ipf_l2_info_devoir.tar.gz`
Cette commande affichera les fichiers décompressés, ils se trouvent tous dans le répertoire `ipf_l2_info_devoir.tar.gz`

Dans ce répertoire, vous trouverez un fichier exécutable : `corrige_jeu_yams.exe`. Ce dernier contient le code corrigé et compilé (il est donc impossible de voir le code à partir de ce fichier). Vous pouvez l'utiliser pour tester le comportement du jeu. En lançant se programme avec `./corrige_jeu_yams.exe`, vous verrez un affichage comme celui ci :

```
Tour 1/13, grille actuelle:
Grille supérieure:
 1: 0 point(s)
 2: 0 point(s)
 3: 0 point(s)
 4: 0 point(s)
 5: 0 point(s)
 6: 0 point(s)
 Bonus >= 62: 0
 Total supérieur: 0
--
Grille inférieure:
 Chance: 0 point(s)
 Brelan: 0 point(s)
 Carré: 0 point(s)
 Full House: 0 point(s)
 Petite suite: 0 point(s)
 Grande Suite: 0 point(s)
 Yams: 0 point(s)
 Total inférieur: 0
--
Score Total: 0
Lancé 1, dés: 0:1  1:6  2:5  3:6  4:2  Positions à conserver:
```

Le programme affiche la grille de score courante, ainsi que le numéro du tour et celui du lancé au sein du tour. Il affiche ensuite les dés tirés avec une position permettant de les sélectionner et attend que l'utilisateur saisisse les dés à conserver. Ici, on souhaite conserver les deux ⚅ se trouvant en position 1 et 3, on écrit donc : `1,3` suivi de la touche entrée (sans espace, séparés par une virgule). En cas de saisie invalide, on reste sur le lancé courant. Après avoir saisi `1,3` l'affichage devient (on ommet le début qui est identique) :

```
...
Score Total: 0
Lancé 2, dés: 0:1  1:6  2:2  3:6  4:4  Positions à conserver:
```

On conserve encore le 1 et le 3 : `1,3`. L'affichage devient :

```
Combinaison finale: 0:6  1:6  2:3  3:6  4:6
Grille supérieure:
 1: 0 point(s)
 2: 0 point(s)
 3: 0 point(s)
 4: 0 point(s)
 5: 0 point(s)
 6: 144 point(s)
 Bonus >= 62: 35
 Total supérieur: 179
--
Grille inférieure:
 Chance: 0 point(s)
 Brelan: 0 point(s)
 Carré: 27 point(s)
 Full House: 0 point(s)
 Petite suite: 0 point(s)
 Grande Suite: 0 point(s)
 Yams: 0 point(s)
 Total inférieur: 27
--
Score Total: 206
Tour terminé, appuyer sur [Enter]
```

On peut facilement vérifier que la grille obtenue tiens compte des règles précédemment expliquées. On peut poursuivre jusqu'au 13ème tour ou interrompre le programme avec CTRL-C.

3.1 Structure du programme

L'archive contient les fichiers suivants :

yams_type.ml Un fichier contenant uniquement la définition de type OCaml et la grille initiale :

```
1  (* Définition des deux types auxiliaires, ne pas modifier ce fichier *)
2  type combinaison = Chance  | Brelan | Carre | Full | Psuite | Gsuite | Yams
3  ;;
4
5  type grille =
6  {
7    sup : (int * int) list;
8    inf : (combinaison * int) list;
9  }
10 ;;
11
12 (* La grille initiale *)
13 let grille_init = {
14   sup = [ (1,0); (2,0); (3,0); (4,0); (5,0); (6,0)];
15   inf = [  (Chance, 0);
16            (Brelan, 0);
17            (Carre, 0);
18            (Full, 0);
19            (Psuite, 0);
20            (Gsuite, 0);
21            (Yams, 0) ]
22 }
```

Comme on le voit, la grille est représentée par un produit nommé avec deux champs (partie supérieure et inférieure). La partie supérieure est une liste de couples d'entiers, indiquant le numéro du dé et le score associé (initialement 0). Le bonus et le total de cette grille seront recalculée automatiquement et ne sont pas stockés. La grille inférieure est représenté par une liste de couples combinaison, entiers. Une combinaison est un type somme représentant l'une des 7 combinaisons possibles. La grille de départ est stocké dans une variable globale `grille_init`.

yams_type.cmi, yams_type.cmo, corrige_yams.cmi, corrige_yams.cmo : des fichiers compilés (binaires) contenant le code du corrigé. Il ne faut ni effacer ni modifier ces fichiers. Si cela vous arrive, il faudra les récupérer depuis l'archive originale. Attention, si vous décompressez de nouveau l'archive, les fichiers existants seront écrasés. Il convient donc de faire une copie de votre fichier `yams.ml` contenant vos réponses avant de décompresser de nouveau l'archive, sinon vous risquez de tout perdre !

yams.ml : le fichier que vous devez compléter contenant la logique du jeu (détection des combinaisons, affichage de la grille, mise à jour du score).

jeu_yams.ml : le progrmame principal qui utilise votre fichier `yams.ml`. Il gère les saisies de l'utilisateur, l'initialisation, …. Vous pouvez le lire, mais ne devez pas le modifier.

Makefile : un fichier de configuration pour la commande `make` qui va compiler votre programme.

3.2 Compilation et tests

Pour compiler votre programme, vous ne devez pas appeler `ocamlc` directement mais utiliser la commande `make` dans le répertoire où se trouve votre code :

- la commande `make` écrite seule dans le terminal, compile votre fichier `yams.ml` et produit une fichier exécutable `jeu_yams.exe`. Vous pouvez tester votre programme en le lançant (`./jeu_yams.exe`) et le comparer au programme de référence (`./corrige_jeu_yams.exe`).
- la commande `make test` lance un interpréteur OCaml avec les fonctions de votre fichier `yams.ml` chargées à l'intérieur. Vous pouvez donc directement appeler vos fonctions pour les tester. Par exemple, pour tester la fonction `lance_de : unit -> int` de la question 1 :

```
$ make test
ocamlc -c yams.ml
rlwrap ocaml -noinit yams_type.cmo corrige_yams.cmo yams.cmo -open Yams_type -open Yams
        OCaml version 4.08.1
# lance_de;;
- : unit -> int = <fun>
# lance_de ();;
- : int = 1
# lance_de ();;
- : int = 2
# lance_de ();;
- : int = 5
# lance_de ();;
- : int = 6
# lance_de ();;
- : int = 4
#
```

Le programme compile votre fichier et lance l'interprète OCaml (ces commandes sont affichées dans le terminal). Vous pouvez ensuite saisir des expressions OCaml. Vous pouvez quitter l'interpréteur avec CTRL-D.

3.3 Utilisation du corrigé

Afin que le devoir ne soit pas un exercice à tirroir (où rater la première question implique de rater la suite), un fichier binaire `corrige_yams.cmo` est fourni. Il contient le corrigé de toutes les fonctions. Vous ne pouvez évidemment pas retrouver le code source correspondant, mais vous pouvez l'utiliser. Le début du fichier `yams.ml` a compléter est comme suit :

```
1  (* La ligne ci-dessous permet d'inclure le fichier yams_type.ml,
2      ne pas la modifier *)
3  open Yams_type
4
5  (* Q1 (1 point) *)
6  let lance_de () =
7    (* À REMPLACER PAR VOTRE CODE *)
8    Corrige_yams.lance_de ()
9
10 ;;
11
12 (* Q2 (1.5 point)*)
13 let lance_n_des n =
14   (* À REMPLACER PAR VOTRE CODE *)
15   Corrige_yams.lance_n_des n
16 ;;
17
18
19 (* Q3 (1 point) *)
20 let somme des =
21   (* À REMPLACER PAR VOTRE CODE *)
22   Corrige_yams.somme des
23 ;;
```

À la ligne 3, l'instruction `open Yams_type` permet d'inclure les définitions du fichiers `yams_type.ml` à cet endroit. Vous pouvez donc considérer que les types `combinaison`, `grille` et la variable globale `grille_init` sont visibles ici.

La fonction `lance_de ()` (ainsi que toutes les suivantes) appelle directement la fonction du corrigé du même nom. Vous devez évidemment remplacer le corps de la fonction par votre propre code, mais, si vous êtes bloqué pour une fonction vous pouvez

— mettre votre code en commentaire. Il sera tout de même corrigé.
— revenir au code initial qui appelle le corrigé.

Cela vous permettra de ne pas rester bloquer sur une question et d'avancer aux questions suivantes sans être pénalisés pour les suivantes (évidemment si vous ne répondez pas du tout à une question et laissez le code initial, vous aurez 0 point).

4 Questions

Il y a 13 fonctions à compléter, chacune rapporte un certain nombre de points indiqué dans le fichier `yams.ml`. Le barème est indicatif et pourra être modifié. Si une question est marquée ⋄, cela signifie qu'un code très semblable a été vu en TP. Vous êtes donc invité à relire les corrigés de TPs disponibles.

0. Modifier les chaînes de caractères se trouvant dans les variables globales `nom`, `prenom` et `num_etud` situées au début du fichier `yams.ml`.
1. ⋄ Compléter la fonction `lance_de` pour qu'elle renvoie un entier alétatoire compris entre 1 et 6.
2. Compléter la fonction `lance_n_des` qui prend en argument un entier `n` et renvoie une liste contenant `n` entiers aléatoires tous compris entre 1 et 6.
3. ⋄ Compléter la fonction `somme` qui renvoie la somme des éléments d'une liste.
4. Compléter la fonction `garde` qui prend en argument deux listes. Une liste de positions `pos` comprises entre 0 et $n-1$ et une liste de `des` de n entiers compris entre 1 et 6. La fonction renvoie une liste où les entiers dont la position est dans la liste `pos` ont été conservés et les autres remplacé par un lancé de dé. Par exemple :

   ```
   # garde [0;4] [1;3;1;5;6];;
   - : int list = [1; 6; 4; 3; 6]
   ```

   Ici on souhaite garder les dés en position 0 et 4 (le 1 et le 6), les dés en position 1,2,3 sont remplacés par de nouvelles valeurs. On pourra utiliser les fonction

   `List.mapi : (int -> 'a -> 'b) -> 'a list -> 'b list` similaire à `List.map` mais où la fonction d'ordre supérieur reçoit en plus la position de l'élément courant dans la liste (à partir de 0)

   `List.mem : 'a -> 'a list -> bool` qui renvoie vrai si et seulement si un élément appartient à une liste.

   Avec ces fonctions, pour chaqu élément de `des`, si sa position est dans `pos` le conserver, sinon le remplacer par un lancé de dé.
5. Compléter la fonction `string_of_comb` qui renvoie une chaîne de caractères représentant une valeur du type `combinaison` (voir le fichier `yams_type.ml`).
6. Compléter la fonction `affiche_grille`. Cette dernière prend en argument une valeur du type `grille` et effectue un affichage des deux grilles comme celui indiqué dans les exemples de la section 3 :

   ```
   Grille supérieure:
    1: 0 point(s)
    2: 0 point(s)
    3: 0 point(s)
    4: 0 point(s)
    5: 0 point(s)
    6: 0 point(s)
    Bonus >= 62: 0
   Total supérieur: 0
   --
   Grille inférieure:
    Chance: 0 point(s)
    Brelan: 0 point(s)
    Carré: 0 point(s)
    Full House: 0 point(s)
    Petite suite: 0 point(s)
    Grande Suite: 0 point(s)
    Yams: 0 point(s)
   Total inférieur: 0
   --
   ```

7. Compléter la fonction `maj_grille`, qui prend en argument une clé `k`, une valeur entière `v` et une liste ocamlin-lineg de paires clés et valeurs.
   — si la paire `(k, 0)` appartient à `g` alors elle est remplacée par `(k, v)`
   — si la paire `(k, n)` appartient à `g` avec `n` différent de 0, une erreur est levée (avec `failwith`).
   — si `k` n'appartient pas à `g`, une erreur est levée (avec `failwith`).
   Cette fonction sera utilisée pour mettre à jour le score. Par exemple :

```
let inf = [(Chance, 0); (Brelan, 0); (Carre, 0); (Full, 0); (Psuite, 0); (Gsuite, 0);
   (Yams, 0)];;
val inf : (Yams_type.combinaison * int) list =
  [(Chance, 0); (Brelan, 0); (Carre, 0); (Full, 0); (Psuite, 0); (Gsuite, 0);
   (Yams, 0)]
# let inf2 = maj_grille Full 25 inf;;
val inf2 : (Yams_type.combinaison * int) list =
  [(Chance, 0); (Brelan, 0); (Carre, 0); (Full, 25); (Psuite, 0);
   (Gsuite, 0); (Yams, 0)]
# let sup = [ (1,0); (2,0); (3,0); (4,0); (5,0);(6,0)];;
val sup : (int * int) list = [(1, 0); (2, 0); (3, 0); (4, 0); (5, 0); (6, 0)]
# let sup2 = maj_grille 2 8 sup;;
val sup2 : (int * int) list =
  [(1, 0); (2, 8); (3, 0); (4, 0); (5, 0); (6, 0)]
# let sup3 = maj_grille 42 8 sup2;;
Exception: Failure "Clé invalide".
```

8. ⋄ Compléter la fonction `calc_mult`. Cette fonction prend en arguement une liste `des` de dés (entiers compris entre 1 et 6) et renvoie une liste de paires d'entiers (d, m) où d est dé et m le nombre de fois où d apparait dans la liste originale (sa multiplicité). La liste résultante est triée par multiplicité croissante. On pourra :
   — trier `des` par ordre croissant
   — parcourir la liste triée récursivement avec un accumulateur pour produire la liste de paires
   — retrier la liste de paire en fonction de la seconde composante.
   Par exemple :

```
  calc_mult [1;1;6;1;4];;
- : (int * int) list = [(4, 1); (6, 1); (1, 3)]
```

9. Compléter la fonction `est_psuite` qui prend en argument une liste `des` de taille 5 et qui teste si cette dernière contient une petite suite.

10. Compléter la fonction `score_comb`. Cette dernière prend en argument une valeur de type combinaison `c` et une liste ~~de paires (d, m) telle que renvoyée par `calc_mult`~~ de dés et qui renvoie le nombre de points pour la combinaison `c` et cette liste. La fonction sera de la forme :

```
let score_comb c des =
    match c, calc_mult des with
      Yams, [ (_, 5) ] -> 50 (* on a bien un yams *)
  | Gsuite, [ (1,1); (2,1); (3,1); (4,1); (5,1)]
  | Gsuite, [ (2,1); (3,1); (4,1); (5,1);(6,1)] -> 40
  | ...
```

11. Compléter la fonction `score_des` qui prend en argument un numéro `num` de dé et une liste de dé et qui renvoie le nombre de points pour la grille supérieure associée au dés `num`. Par exemple

```
score_des 4 [ 1;5;4;6;4];;
- : int = 32
```

    La fonction ci-dessus effectue : $4 \times (4 + 4)$.

12. Compléter la fonction `maximum`. Cette dernière prend en argument une liste de paires (v, s). Si la liste est vide, alors la fonction lève une erreur avec `failwith`. Sinon la fonction renvoie la paire pour laquelle s est le plus grand dans la liste.

13. Compléter la fonction `score_max`. Cette fonction est telle que : `score_max fscore scores des` :
   — filtre la liste `scores` de paires (v, d) pour ne garder que les paires pour lesquelles $d = 0$
   — pour chaque élément $(k, 0)$ de cette liste, renvoie la paire $(k,$ `fscore k des`$)$ permettant de calculer le score pour la valeur k avec les dés `des`.
   — utilise `maximum` pour renvoyer le plus grand score obtenu dans la liste précédente

— utilise `maj_score` pour mettre à jour le score correspondant dans la liste `scores`
Par exemple :

```
# score_max score_comb  [(Chance, 0); (Brelan, 0); (Carre, 0);
   (Full, 0); (Psuite, 0); (Gsuite, 0); (Yams,0) ] [1;1;1;3;3];;
- : (Yams_type.combinaison * int) list =
[(Chance, 0); (Brelan, 0); (Carre, 0); (Full, 25); (Psuite, 0); (Gsuite, 0); (Yams, 0)]
# score_max score_des  [ (1,0); (2,0); (3,0); (4,0); (5,0);(6,0)] [1;1;1;3;3];;
- : (int * int) list = [(1, 0); (2, 0); (3, 18); (4, 0); (5, 0); (6, 0)]
```

14. Compléter la fonction `maj_score g des` prenant en argument une valeur de type `grille` et une liste de dés et qui renvoie la nouvelle grille de score. La fonction `score_max` peut être utilisée comme dans l'exemple précédent pour calculer le score de la grille supérieure et inférieure.