TP 2

1 Échauffement : récursion

1. Le code suivant est incorrect. Quel message d'erreur affiche-t-il ? Corrigez-le.

```
let decompte n =
   match n with
   |0 -> ()
   |_ -> print_int n; decompte (n-1);;

decompte 10;;
```

2. Modifiez (un tout petit peu) le code de `decompte` pour écrire un fonction `compte` qui affiche les nombre de 1 à n dans l'ordre croissant.
3. Que fait la fonction suivante ?

```
let mystere n =
   match n with
    | _ when n>100 ->
        failwith "Trop grand";
        Printf.printf "fantome"
    | _ -> compte n;;
```

4. Modifier le code pour lever une exception "Trop petit" lorsque `mystere` est appelé avec un nombre négatif.

2 Programmation : *drag and drop*

Le but de cet exercice est de programmer une petite application pour déplacer un disque de couleur à l'aide de la souris, selon le principe du *drag and drop* : on clique sur le disque pour le sélectionner, on le déplace avec la souris tant que le bouton est pressé, puis on le "repose" en relâchant le bouton.

On va tout d'abord se familiariser avec la bibliothèque graphique `graphics` d'OCaml. Pour cela, on commence par étudier le petit `dragdrop.ml` programme suivant.

```
open Graphics;;

open_graph " 300x400";;

let rec jeu () =
  let event = wait_next_event [Button_down] in
  let mx = event.mouse_x in
  let my = event.mouse_y in
  if mx < 10 && my < 10 then exit 0;
  Format.printf "%d %d@." mx my;
  jeu ()
;;

jeu ();;
```

On compile ce programme à l'aide de la ligne de commande suivante :

```
ocamlc -o dragdrop graphics.cma dragdrop.ml
```

Explications. Le programme commence par lancer une fenêtre graphique de taille 300×400. Puis, il contient une fonction récursive `jeu` qui attend que l'utilisateur clique avec la souris, affiche les coordonnées de la souris et se rappelle récursivement. La fonction se termine quand l'utilisateur clique dans la zone $[0, 10] \times [0, 10]$.

L'appel de fonction `wait_next_event [...]` attend qu'un des évènements mentionnés dans la liste `[...]` arrive. L'évènement `Button_down` correspond à un clic du bouton de la souris. Il y a d'autres évènements comme `Mouse_motion` (déplacement de la souris) ou `Button_up` (lorsqu'on relâche le bouton de la souris). On peut attendre un évènement parmi une liste d'évènements, par exemple `Mouse_motion; Button_down` attend *soit* que la souris se déplace, *soit* un clic.

La valeur renvoyée par la fonction `wait_next_event` donne l'état de la souris et du clavier. Par exemple, les valeurs `event.mouse_x` et `event.mouse_y` sont les coordonnées (de type `int`) de la souris, `event.button` est un booléen à vrai si le bouton est appuyé.

1. Modifier le programme précédent pour afficher les coordonnées de la souris à chaque mouvement.
2. Modifier le programme pour sortir uniquement quand l'utilisateur clique avec la souris.

2.1 La fonction *push*

La première fonction à écrire consiste à détecter un clic de souris au sein du cercle. Pour cela, écrire une fonction récursive

```
val push : int -> int -> unit
```

tel que l'appel de fonction `push x y` effectue les étapes suivantes :

1. Attendre un clic de souris dans la fenêtre
2. Vérifier que l'on a cliqué dans le cercle de centre `(x,y)` sinon revenir en 1.
3. Colorier le cercle en rouge.

On utilisera la fonction `fill_circle` de la bibliothèque `Graphics` pour dessiner un disque. Un appel `fill_circle x y r` dessine un cercle centré en $(\texttt{x}, \texttt{y})$ de rayon `r`. La fonction `set_color` permet de changer la couleur pour dessiner. Pour cela, il suffit de l'appeler avec une couleur comme `black`, `red`, etc.

Écrire ensuite un programme qui utilise la fonction `push` de la manière suivante :

1. Dessiner un cercle noir de rayon 10 et placé en $(100, 100)$.
2. Appeler la fonction `push 100 100`.
3. Attendre un clic de souris dans la fenêtre.
4. Quitter.

2.2 Faire du *drag and drop*

Pour terminer le programme `dragdrop.ml`, écrire une fonction de déplacement

```
val drag : int -> int -> int -> int -> unit
```

mutuellement récursive avec `push` tel que l'appel `drag x y mx my` effectue les étapes suivantes :

1. Attendre que la souris soit déplacée ou le bouton relâché.
2. Rappeler `push` si le bouton est relâché.
3. Sinon calculer les nouvelles coordonnées du cercle.
4. Retourner en 1

Enfin, modifier la fonction `push` de manière à ce que l'étape 3 de son algorithme soit remplacée par un appel à la fonction `drag`.

3 Un casse brique sans briques

Le but de cet exercice est de créer un petit jeu de casse briques qui consiste simplement à faire rebondir une balle dans un cadre à l'aide d'une raquette.

3.1 Le cadre

On commence par définir dans un fichier `cb.ml` les constantes `left`, `right`, `down` et `up` de type `float` avec les valeurs `0.`, `300.`, `0.` et `500.` respectivement. Ces valeurs représentent les limites du cadre.

Écrire ensuite un programme qui ouvre une fenêtre graphique dont les dimensions correspondent à ces 4 constantes. On pourra utiliser pour cela la fonction `int_of_float` pour convertir un flottant en entier.

Afin d'obtenir un affichage graphique de qualité, on ajoutera l'appel `auto_synchronize false` juste après l'ouverture de la fenêtre graphique

3.2 La balle

La balle est représentée par une paire de nombres flottants qui correspondent à sa position. Elle sera dessinée par un cercle dont le rayon est défini dans par la constante `ball`. Ajouter à votre programme la déclaration de la constante `ball` en fixant une valeur pour le rayon de la balle (`5` par exemple). Ensuite, en utilisant la fonction `fill_circle`, écrire une fonction `draw_ball` de type

```
val draw_ball : float -> float -> unit
```

qui affiche la balle comme un cercle de couleur noire.

3.3 Mouvement de la balle

La direction et la vitesse de la balle sont donnés par un vecteur vitesse. Ce vecteur est représentée par une paire de nombres flottants. On utilisera la fonction `Random.float` pour fixer une valeur aléatoire à ce vecteur (comprise dans l'intervalle $[0; 1] \times [0; 1]$). Écrire une fonction `new_position` de type

```
val new_position : (float * float) -> (float * float) -> (float * float)
```

qui, à partir de la position de la balle et de son vecteur vitesse, calcule les nouvelles coordonnées de la balle.

3.4 La raquette

La raquette est représentée par un rectangle dont la longueur est définie dans la constante `paddle`. Cette raquette sera déplacée à l'aide de la souris. La position de la raquette est celle de son point le plus à gauche. L'épaisseur de la raquette est définie à l'aide d'une constante `thick`.

À l'aide de la fonction `fill_rect`, écrire une fonction `draw_paddle : int -> unit` tel que `draw_paddle x` dessine la raquette comme un rectangle noir de longueur `paddle` (et d'épaisseur `4` par exemple) à la position donnée par la paire `(x, down)`. À l'aide de la fonction `mouse_pos`, écrire une fonction `position_paddle : unit -> int` qui, en fonction de la position de la souris, renvoie l'abscisse gauche de la raquette.

3.5 Rebonds

Le vecteur vitesse de la balle est modifiée quand celle-ci rebondit sur un bord (gauche ou droit) du cadre, en haut du cadre ou sur la raquette. Écrire une fonction `bounce` de type

```
val bounce : (float * float) -> (float * float) -> int -> (float * float)
```

telle que `bounce (x, y) (vx, vy) p` renvoie le nouveau vecteur vitesse de la balle en fonction de la position `(x, y)` de la balle, de son vecteur vitesse courant `(vx, vy)` et de la position de la raquette `p`.

3.6 Boucle principale du jeu

La boucle principale du jeu est réalisée à l'aide d'une fonction récursive `game` de type

```
val game = (float * float) -> (float * float) -> unit
```

qui prend en argument les coordonnées courantes de la balle ainsi que son vecteur vitesse. L'algorithme de cette fonction consiste à (récursivement) :

1. Effacer la fenêtre graphique à l'aide de la fonction `clear_graph ()`

2. Afficher la balle en `(x, y)`
3. Tester si la balle est sortie du jeu
4. Calculer la nouvelle position de la balle (éventuellement après rebond).

Afin de ralentir le jeu (les machines actuelles sont très rapides), on utilise des appels à la fonction suivante.

```
let rec wait n =
  if n=0 then Graphics.synchronize () else wait (n-1)
```

Écrire la fonction `game` et terminer votre programme par un appel à cette fonction (en fixant les coordonnées de départ de la balle).