DEVOIR MAISON D'OPTION INFORMATIQUE
Triangle de Sierpinski déformé

L'objectif de ce DM est de tracer en OCaml la figure fractale du triangle de Sierpinski, en introduisant une légère déformation aléatoire. Dans un second temps, on remplit les trous de la figure, ce qui permet d'obtenir une image ressemblant à la triangulation d'une surface présentant des reliefs.

1 Tracé du triangle de Sierpinski

Nous allons utiliser la bibliothèque graphique d'OCaml pour réaliser les tracés. Votre fichier doit ainsi commencer par les lignes suivantes :

```
#load "graphics.cma";;
#load "unix.cma";;

Graphics.open_graph " 1200x800";;

let attendre f = ignore (Unix.select [] [] [] f ) ;;

let largeur = Graphics.size_x();;
let hauteur = Graphics.size_y();;

type point = int * int;;
```

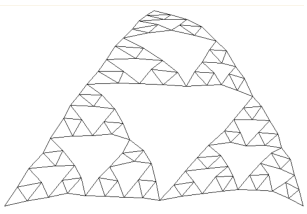
Un point de l'image est représenté par un couple de coordonnées **entières** :

```
type point = int * int;;
```

1. Écrire une fonction `tracer_triangle : point -> point -> point -> unit` traçant le triangle reliant les 3 points. On utilisera les fonctions suivantes :
   - `Graphics.moveto : int -> int -> unit`, qui déplace le point actuel aux coordonnées indiquées sans tracer de trait ;
   - `Graphics.lineto : int -> int -> unit`, qui déplace le point actuel aux coordonnées indiquées en traçant un trait.
2. Écrire une fonction `distance : point -> point -> int` renvoyant la partie entière de la distance euclidienne entre deux points. On pourra utiliser les fonctions suivantes :
   - `int_of_float : float -> int`
   - `float_of_int : int -> float`

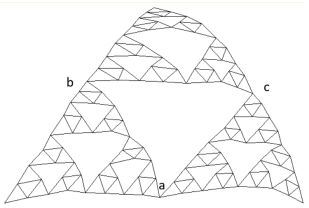
   Ainsi que l'opérateur flottant `**`.
3. Écrire une fonction `milieu : point -> point -> point` renvoyant le milieu entre deux points $(x_1, y_1), (x_2, y_2)$, auquel on ajoute une petite perturbation en abscisse a_x et en ordonnée a_y égales à $\frac{d}{10\, r(1,6)}$, où d est la distance entre les points, et $r(1,6)$ un tirage uniforme (indépendant pour a_x et a_y) d'un entier entre 1 et 6. On pourra utiliser la fonction `Random.int : int -> int`.
4. Écrire une fonction `triangle_sierpinski : int -> point -> point -> point -> unit` prenant en entrée un entier n et trois points, et traçant le triangle de Sierpinski à n niveaux délimité par les trois points.

   Ainsi, l'appel `triangle_sierpinski 4 (10,10) (largeur - 10, 10) (largeur/2, hauteur - 10)` doit afficher ce type d'image :



2 Complétion des trous

Pour écrire une fonction complétant un trou du triangle de Sierpinski, on ne peut pas recalculer les milieux, car il n'y aurait aucune garantie de retomber sur les mêmes perturbations aléatoires. Nous aurons donc besoin de lui donner en entrée l'ensemble des points déjà construits sur les arêtes du triangle à compléter. Plus précisément, pour chaque arête, dont on a choisi une orientation, on veut stocker la séquence des points déjà construits sauf le premier. Ainsi, dans l'image précédente :



en choisissant d'orienter les arêtes dans le sens horaire, l'arête a-b contient 8 points, a étant exclu et b étant le dernier ; l'arête b-c contient également 8 points, b étant exclu et c étant le dernier ; et l'arête c-a contient 8 points, c étant exclu et a étant le dernier.
On remarque que les arêtes des trous ainsi définies contiennent toujours un nombre de points qui est une puissance de 2, et nous allons représenter efficacement de telles arêtes par des arbres binaires complets, dont les feuilles seront les points, dans l'ordre qui a été précisé :

```
type arbre =
  | Feuille of point
  | Noeud of arbre * arbre
;;

type triangle = arbre * arbre * arbre
(* trois arêtes cycliquement consécutives A->B, B->C, C->A *)
;;
```

Une telle structure permet notamment d'obtenir efficacement une arête à partir de ses deux moitiés, et réciproquement.

1. Écrire une fonction `dernier_point : arbre -> point` prenant en entrée une arête sous forme d'arbre, et renvoyant son dernier point.
2. Écrire deux fonctions **mutuellement récursives** :
   - `triangle_sup : int -> point -> point -> point -> triangle` reprenant la base de la fonction `triangle_sierpinski`, en complétant les trous présents, et renvoyant les trois arêtes au bord du triangle, **orientées dans le sens trigonométrique** ;
   - `triangle_inf : int -> triangle -> unit` prenant un niveau et 3 arêtes **orientées dans le sens horaire**, et complétant le trou (qui est donc un triangle pointe vers le bas) délimité par ces trois arêtes (en traçant les triangles adéquats à l'intérieur).

   Indication : Faire un schéma !

   L'appel `triangle_sup 4 (10,10) (largeur - 10, 10) (largeur/2, hauteur - 10)` doit ainsi afficher ce type d'image :