Tortue graphique et fractales

A rendre le jour de la soutenance (vendredi 16 janvier 2015)

Le projet est à réaliser en OCaml individuellement. Il sera accompagné d'un dossier contenant impérativement la description des choix faits, la description des types et des fonctions. Pour chaque fonction, on donnera impérativement l'interface complète (dans le code en commentaire et dans le rapport pour les fonctions présentées).

Même si le sujet est décomposé en questions, en général chaque question se résoud par l'écriture d'une ou plusieurs fonctions intermédiaires. Celles-ci doivent comporter une interface également.

Le dossier fournira également des cas de tests accompagnés des résultats attendus et retournés.

Le dispositif graphique virtuel appelé *tortue* a fait la célébrité du langage de programmation Logo (`http://en.wikipedia.org/wiki/Logo_programming_language`), en particulier pour l'apprentissage de la programmation. Le but de ce projet est d'implanter une tortue en Caml, puis s'en servir pour tracer des courbes *fractales* (`http://en.wikipedia.org/wiki/Fractal`).

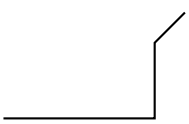
Afin d'effectuer les dessins, un module `Dessin` est fourni, qui propose un type Caml pour modéliser les segments de droite :

```
type segment =
  {
    x1 : float;
    y1 : float;
    x2 : float;
    y2 : float;
  }
```

Executez la commande `#load "graphics.cma";;`, puis la commande `#load "dessin.cmo";;` [1], une fenêtre d'affichage graphique s'ouvre à l'écran. Une fonction `trace_ligne` est fournie, qui prend deux arguments : une couleur et une liste de segments, et qui trace ces segments sur la fenêtre graphique, dans la couleur spécifiée. Les couleurs seront nommées en anglais `black`, `red`, `blue`, `green`, `yellow`, `cyan` et `magenta`. Cette fonction prend soin d'adapter l'échelle de son tracé pour que la liste des segments apparaisse dans la fenêtre graphique.

1 Tortue graphique

La tortue graphique est un dispositif virtuel qui trace des lignes à l'écran. On l'utilise en lui donnant des ordres de deux types : avancer d'un certain nombre de pas (en traçant le chemin parcouru), et tourner d'un certain angle. Ainsi, pour une tortue initialement dirigée vers la droite, la succession des ordres avancer 100, tourner 90, avancer 50, tourner -45, avancer 30 produit le dessin suivant :



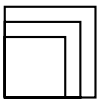
[1] dessin.ml se trouve dans le répertoire `http://dpt-info.u-strasbg.fr/~magaud/IPF/dessin.ml`, pour le compiler faire `ocamlc -c dessin.ml`

On crée un premier type `etat_tortue`, qui décrit l'état courant d'une tortue : un triplet de deux réels et un entier, qui définissent son abcisse, son ordonnée, et sa direction par un angle (en degrés) par rapport à l'axe des x. Afin de modéliser les dessins à l'écran de manière fonctionnelle, on introduit le domaine des *configurations graphiques* : une configuration graphique décrit à la fois l'état courant de la tortue, et l'ensemble des segments déjà tracés à l'écran. Les deux actions de la tortue (avancer et tourner) sont alors spécifiées de manière fonctionnelle par :

- avancer (c : configuration graphique, n : entier) g→ configuration graphique
  { donne la configuration obtenue à partir de c si la tortue avance de n pas }

- tourner (c : configuration graphique, d : entier) → configuration graphique
  { donne la configuration obtenue à partir de c si la tortue tourne de d degrés vers la gauche (un d négatif tourne en fait à droite) }

1. Compléter le type enregistrement Caml `configuration` codant les configurations graphiques. Coder la fonction `creer_config` qui produit la configuration où rien n'est encore tracé, et où la tortue se trouve en (0,0) avec un angle donné.
2. Coder les fonctions `avancer` et `tourner` (utiliser les fonctions `conv_angle`, `sin` et `cos`). Tester en particulier sur l'exemple de dessin simple ci-dessus, déjà donné dans le fichier.
3. Spécifier puis coder une fonction qui dans une configuration donnée trace un carré pour une longueur de côté donné, et retourne la configuration obtenue. Tester cette fonction en traçant le dessin suivant :



2 La courbe de Von Koch

Notre première utilisation avancée de la tortue est pour tracer la courbe de Von Koch. La courbe de Von Koch de niveau 0 est une ligne horizontale, la courbe de niveau 1 est la ligne à 4 segments suivante :

et de façon générale, la courbe de Von Koch de niveau n est obtenue en remplaçant dans la courbe de niveau $n-1$ chaque segment par le motif ci-dessus.

1. Coder la fonction `von_koch` qui étant donné un entier n trace la courbe de Von Koch de niveau n (dans une configuration donnée).
2. Coder la fonction `flocon` qui enchaîne trois courbes de Von Koch de niveau donné, avec une rotation de 120 degrés vers la droite entre chacune d'elles.

3 L-systèmes

Un système de Lindenmayer, ou L-système, est une description compacte d'une courbe fractale, par la donnée de *règles de remplacement*. Une courbe est décrite par une séquence de caractères, où

- `'F'` signifie avancer d'un pas ;
- `'+'` signifie tourner à gauche (d'un angle fixé) ;
- `'-'` signifie tourner à droite (du même angle fixé) ;

Ainsi, la courbe de Von Koch de niveau 1 est décrite par `F+F--F+F`, avec un angle de 60 degrés.

1. Coder la fonction

   action_tortue (a : entier, c : configuration, car : caractère) → configuration
   { calcule la configuration obtenue à partir de c en effectuant
     l'action indiquée par le caractère car et l'angle a,
   (la configuration reste inchangée si car est différent de `'F'`,`'+'` et `'-'`. }

   puis la fonction

   tracer_sequence (a : entier, c : configuration, s : sequence de caractère) → configuration
   { calcule la configuration obtenue à partir de c en effectuant
     les actions indiquées par la séquence de caractères s et l'angle a }

Pour engendrer une courbe fractale par un L-système, on produit sa description en caractères par des règles de remplacement d'un caractère par une séquence de caractères. Ainsi, la description de la courbe de Von Koch de niveau n est obtenue à partir de `F` en appliquant n fois la règle de remplacement de `F` par `F+F--F+F`. Un codage du domaine des règles est fourni par

```
type regle =
    {
      remplacer : char;
      par : char list;
    }
```

1. Coder la fonction `creer_regles` qui à une séquence de couples (caractère, chaîne de caractères) associe l'ensemble de règles correspondant. Par exemple :

   ```
   creer_regles [ ('A',"BF+") ; ('B',"-FA") ]
   ```

   donne

   ```
       [
         { remplacer = 'A' ; par = [ 'B' ; 'F' ; '+' ] } ;
         { remplacer = 'B' ; par = [ '-' ; 'F' ; 'A' ] } ;
       ]
   ```

   On utilisera la fonction `lettres_d_un_mot` fournie.

2. Coder la fonction `remplace_car` qui étant donné une séquence de règles, et un caractère, donne la séquence obtenue par remplacement de ce caractère (inchangée si aucune règle ne concerne ce caractère). Par exemple, avec les règles R ci-dessus

   `remplace_car R 'A'` donne `[ 'B' ; 'F' ; '+' ]`

   et

   `remplace_car R 'C'` donne `[ 'C' ]`

3. Coder la fonction `deplier_une_fois` qui applique des règles sur chacun des caractères d'une séquence. Par exemple

   ```
   deplier_une_fois R ['A' ; 'F' ; 'B' ]
   ```

   donne

   ```
   [ 'B' ; 'F' ; '+' ; 'F' ; '-' ; 'F' ; 'A']
   ```

4. Coder la fonction `deplier` qui applique un nombre donné de fois les remplacements. Par exemple

```
deplier 2 R ['A' ; 'F' ; 'B' ]
```

donne

```
[ '-' ; 'F' ; 'A' ; 'F' ; '+' ; 'F' ; '-' ; 'F' ; 'B' ; 'F' ; '+' ]
```

5. Coder la fonction `tracer`, qui produit la liste de segments d'une courbe décrite par un ensemble de règles, un mot de de départ, un nombre d'itérations, un angle pour les rotations de la tortue, et un angle de départ.

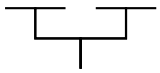
   Tester avec quelques exemples de fractales : voir par exemple `http://en.wikipedia.org/wiki/Lindenmayer_system` .

4 Tortue avec mémoire

On va maintenant ajouter une fonctionnalité à notre dispositif de tortue graphique : une mémoire. On désire ajouter deux fonctions : une dont le rôle est de mémoriser l'état courant de la tortue, et une autre pour replacer la tortue dans un état précédemment mémorisé. Plus généralement, on veut permettre de mémoriser successivement un nombre arbitraire d'états, et alors la fonction de replacement mettra la tortue dans le dernier état mémorisé, puis l'avant-dernier, etc.

Faites une souvegarde de votre fichier avant de continuer !

1. Changer la définition des configurations, afin de rajouter la mémoire de la tortue.
2. Modifier les fonctions `creer_config`, `avancer` et `tourner`, puis coder des fonctions `memoriser` et `replacer`, qui respectivement enregistre l'état de la tortue et replace la tortue dans un état précédemment mémorisé.
3. Tester en réalisant le dessin suivant (arbre généalogique à 2 niveaux) en partant du bas :



4. Coder une fonction qui pour un entier n donné dessine un arbre généalogique à n niveaux.

Pour notre description des actions de la tortue par des caractères, on ajoute :

- `'['` signifie mémoriser l'état ;
- `']'` signifie replacer la tortue dans l'état précédemment mémorisé.

1. Compléter la fonction `action_tortue`.
2. Vérifier que vos tests existants fonctionnent encore, puis tester les derniers exemples du fichier.

5 Pour en savoir plus sur les L-systèmes

Livre : A. Lindenmeyer et P. Prusinkiewicz. The algorithmic beauty of plants, Springer Verlag, 1990.

Sur le web :
`http://en.wikipedia.org/wiki/Lindenmayer_system`
`http://www.math.okstate.edu/mathdept/dynamics/lecnotes/node12.html`
`http://mathforum.org/advanced/robertd/lsys2d.html`
`http://www.cs.unh.edu/~charpov/Programming/L-systems/`