Dessiner avec scheme

Lien vers la bibliothèque graphique de racket:
http://docs.racket-lang.org/draw/index.html

Note : comme dans tous les systèmes de dessin, en scheme, le point de coordonnées (0,0) est en haut à gauche, et le point de coordonnées (xmax,ymax) est en bas à droite !

L'objectif du TP est de vous faire découvrir la bibliothèque graphique de scheme (ou plus exactement de racket). Pour dessiner, il faut d'abord créer deux objets : l'espace dans lequel on va dessiner et un contexte (les couleurs, les crayons, etc …). Pour l'espace, nous choisissons un bitmap, créé avec la fonction make-bitmap, qui prend en argument la hauteur et la largeur du bitmap :

```
(define target (make-bitmap 30 30))
```

Ensuite, nous créons un contexte de dessin pour notre bitmap:

```
(define dc (new bitmap-dc% [bitmap target]))
```

Pour dessiner, il faut ensuite envoyer (send) des commande au contexte. Par exemple, pour dessiner un rectangle à partir du point (0,10) (coin supérieur gauche), de longueur 30 et de hauteur 20 :

```
(send dc draw-rectangle 0 10 30 20)
```

Si vous avez essayé d'évaluer ces commandes, vous vous êtes rendu compte que cela ne fonctionne pas ! C'est parce qu'il faut commencer par inclure la bibliothèque graphique. Cela se fait en utilisant la fonction require (l'équivalent du #include de C) :

```
(require racket/draw)
```

Maintenant, cela s'évalue sans erreur, mais cela ne produit rien ! Il faut en effet indiquer une sortie pour notre bitmap. Il y a plusieurs possibilités : le mettre dans un fichier, l'afficher dans une fenêtre ou créer un snip pour qu'il s'affiche dans l'éditeur. C'est cette dernière que nous allons utiliser maintenant :

```
(make-object image-snip% target)
```

Cela ne marche pas, parce qu'il faut inclure la bibliothèque de racket qui contient la définition des snips :

```
(require racket/snip)
```

Tout fonctionne correctement maintenant, et vous pouvez voir, dans l'éditeur une image :

Comme dans tous systèmes de dessin qui se respectent, on peut changer la couleur de fond (set-brush) et la couleur des traits (set-pen) :

```
(send dc set-brush "green" 'solid)
(send dc set-pen "red" 1 'solid)
```

Et si on dessine un rectangle à l'intérieur du premier :

```
(send dc draw-rectangle 2 12 26 16)
```

On obtient :

Vous pouvez regarder la doc des contextes de dessin pour explorer l'ensemble des fonctions de dessin disponibles.

Pour finir cette introduction, on peut noter qu'il n'est pas nécessaire de nommer le bitmap, puisqu'on dessine sur le contexte, et que de toutes façon il existe un moyen de récupérer le bitmap associé à un contexte par la commande get-bitmap. La séquence :

```
(define dc2 (new bitmap-dc% [bitmap (make-bitmap 30 30)]))
(send dc2 draw-rectangle 0 10 30 20)
(send dc2 set-brush "green" 'solid)
(send dc2 set-pen "red" 1 'solid)
(send dc2 draw-rectangle 2 12 26 16)
(make-object image-snip% (send dc2 get-bitmap))
```

devrait conduire au même résultat.
Il est également possible de mettre ces commandes dans une fonction, ce qui à l'avantage de créer un contexte privé :

```
(define (dessin1)
  (define dc (new bitmap-dc% [bitmap (make-bitmap 30 30)]))
  (send dc draw-rectangle 0 10 30 10)
  (send dc draw-line 0 0   30 30)
  (send dc draw-line 0 30 30 0)(make-object image-snip% (send dc get-bitmap)))
```

Une fois notre fonction dessin1 définie, il ne reste plus qu'à l'appeler :

```
(dessin1)
```

Même si le style de programmation contient un peu d'impératif, cela reste du scheme. Par exemple, si on souhaite obtenir cette image :

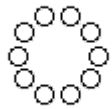
Il faudra le faire de cette façon :

```
(define (dessin2)
  (define dc (new bitmap-dc% [bitmap (make-bitmap 100 100)]))
  (define (boucle i)
    (cond ((= i 10) )
          (else (send dc draw-rectangle (* i 5) (* i 5) (* i 5) (* i 5))
                (boucle (add1 i)))))
  (boucle 0)
  (make-object image-snip% (send dc get-bitmap))
)
(dessin2)
```

Maintenant, si on veut enregistrer l'image dans un fichier plutôt que de l'afficher, il faudra envoyer la commande save-file au bitmap :

```
(define (dessin3)
  (define dc (new bitmap-dc% [bitmap (make-bitmap 100 100)]))
  (define (boucle i)
    (cond ((= i 10) )
          (else (send dc draw-rectangle (* i 5) (* i 5) (* i 5) (* i 5))
                (boucle (add1 i)))))
  (boucle 0)
  (send (send dc get-bitmap) save-file "dessin3.png" 'png)
)
(dessin3)
```

Essayez de produire ce dessin :



Pour cela, vous trouverez peut-être plus pratique de vous définir une fonction affichant un cercle de centre (x,y) et de rayon r :

```
(define (draw-circle x y r dc)
  (send dc draw-ellipse (- x r) (- y r) (* 2 r) (* 2 r)))
```

Et enfin, une belle spirale :

Vous allez maintenant dessiner une courbe, la courbe de koch, dont on peut trouver une définition ici : http://fr.wikipedia.org/wiki/Flocon_de_Koch. C'est une courbe définie récursivement, par transformation des segments qui la compose. Pour la petite histoire, cette courbe est une fractale, elle a la propriété d'avoir une dimension entre 1 (dimension d'une ligne) et une dimension 2 (dimension d'un plan). Pour l'algorithme de création de la courbe, vous pourrez vous inspirer de celui ci : http://algor.chez.com/math/koch.htm (en ne considérant qu'un seul coté du flocon). Le résultat à obtenir est (avec une profondeur de 5):

Notez que vous aurez besoin de définir des variables locales (les coordonnées des points c, d et e), pour cela il est nécessaire d'utiliser let* plutôt que let parce que le point e est défini à partir des point c et d (cf doc)

Pour finir, scheme propose une bibliothèque assez complète de tracés de courbes (fonctions, séries de points, surface 3D, …). Par exemple, essayez ceci :

```
(require plot)
(plot  (function (lambda (x) (/ (sqr x) 2)) 1 100))
```