CHAPITRE XII

GRAPHIQUE

Instructions de base

La directive, avant tout nécessaire, est < #open "graphics";; > (attention, avec le # au début).
Les axes sont dans le sens usuel, l'abscisse allant de gauche à droite de 0 à 480, l'ordonnée de bas en haut de 0 à 280. Pour placer et tracer un segment les mots réservés sont moveto et lineto.

Les couleurs black = 0 blue = 255 green = 65280 cyan = 65535 red = 16711680 magenta = 16711935 yellow = 16776960 white = 16777215 sont déjà définies, et on peut poser par exemple :

```
let rose = 16737946 and mandarine = 16737792 and orange = 16724736
and jaunefaible = 16763904 and saumon = 16750950 and violet = 13369599
and indigo = 6684927 and turquoise = 65484 and vertjaune = 10092288;;
```

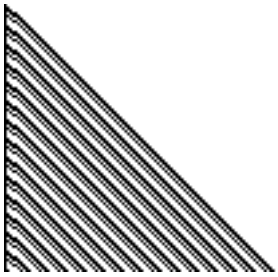
moveto x y;;	Place le point courant aux coordonnées x, y
lineto x y;;	Trace un segment du point courant au point spécifié qui devient le point courant.
set_color black;;	Fixe la couleur courante (par défaut noir).
set_line_width 3;;	Largeur 3 des lignes (1 par défaut).
current_point ();;	Renvoie le couple (x, y)
point_color x y;;	Renvoie la couleur du point.
draw_rect x y l h;;	Dessine un rectangle de coin inférieur gauche (x, y), largeur l et hauteur h.
fill_rect x y l h;;	Dessine le rectangle spécifié dans la couleur courante.
draw_circle x y r;;	Dessine le cercle de centre x, y) et de rayon r.
fill_circle x y r;;	Dessine le disque spécifié dans la couleur courante.
draw_arc x y rx ry a1 a2;;	Arc de a1 à a2 de l'éllipse centrée en (x, y) de demi-axes rx et ry.
draw_ellipse x y rx ry;;	Dessine une éllipse .....
fill_ellipse x y rx ry;;	Remplit avec la couleur courante l'éllipse .....
fill_poly tp;;	Remplit le polygone dont les sommets sont des couples d'entiers.
set_font ;;	Fixe l'alphabet choisi, par exemple ci-dessous "venice"
set_text_size ;;	Fixe la taille des écritures de 7 à 24 ou plus.
draw_string " ";;	Ecrit un texte, par exemple : #let message= "Titre" in moveto 0 0; draw_string message ;;
key_pressed () ;;	Renvoie le booléen "vrai" dès qu'une touche est pressée.
read_key ();;	Attend un caractère au clavier.
open_grapf"";;	Ouvre une fenêtre graphique.
clear_graph ();	Efface la fenêtre graphique.
close_graph ();;	Ferme la fenêtre graphique.

1. Dessin d'un cadre

Ce petit exemple montre ce que l'on peut faire avec les couleurs, polices etc. On définit ici (ce n'est pas la coutume) une constante "ecrire" dont l'éxecution provoque le dessin ci-dessous dès le chargement des deux définitions.

```
let cadre = open_graph" "; moveto 5 5; lineto 400 5; lineto 400 275;
        lineto 5 275; lineto 5 5; moveto 20 10;
        draw_string " Presser une touche ...";
        read_key ();
        close_graph ();;

let ecrire = open_graph" "; set_color magenta; moveto 5 5; lineto 400 5;
        lineto 400 40; lineto 5 40; lineto 5 5;
        moveto 100 15; set_text_size 18; set_color blue; set_font "venice";
        draw_string "Presser une touche ...";
        read_key ();
        close_graph ();;
```

2. Exemple d'utilisation de la lecture de couleur, les règles de StephenWolfram



On dessine (aligné à gauche) un triangle partant d'un point noir sur une ligne 1, deux points la ligne 2 en dessous et trois points noirs sur la ligne 3. Puis, n points sur la ligne n, suivant la "règle 110" simple, les trois points au dessus sont tous noirs ou tous blancs ou noir-blanc-blanc alors le point reste blanc, sinon on le trace noir et on termine la ligne par un point noir. Ci-contre la règle "essai" :

```
#open "graphics";;
let point x y =  set_color black; moveto x y; lineto x y;;

let regle110 u v w =
        mem (u, v, w) [(black, black, black); (white, white, white); (black, white, white)] ;;

let essai1 u v w =
        mem (u, v, w) [(black, white, black); (white, black, white);
                       (white, white, black); (black, black, white)] ;;
let essai2 u v w =
        mem (u, v, w) [(black, white, white); (white, black, black);
                       (white, black, white); (white, white, black)] ;;

let triangle n regle = open_graph" "; point 0 n; point 0 (n-1); point 1 (n-1);
        point 0 (n-2); point 1 (n-2); point 2 (n-2);
        for y = n-3 downto 1 do
                for x = 0 to n-y do
                if regle (point_color x (y+1)) (point_color x (y+2)) (point_color x (y+3))
                then point x y
                done;
        point (n-y+1) y
        done;
        read_key (); close_graph ();;

| triangle 100 regle110;;
```

3. Tracé de courbe

Il faut bien montrer un ou deux exemples de belles courbes. Dans aucun langage de programmation cela ne pose de problème théorique si on s'en tient à des tracés point par point. Par contre il y a toujours un problème d'échelle généralement fastidieux, doublé pour Caml d'un problème permanent de conversion entre entiers et réels.

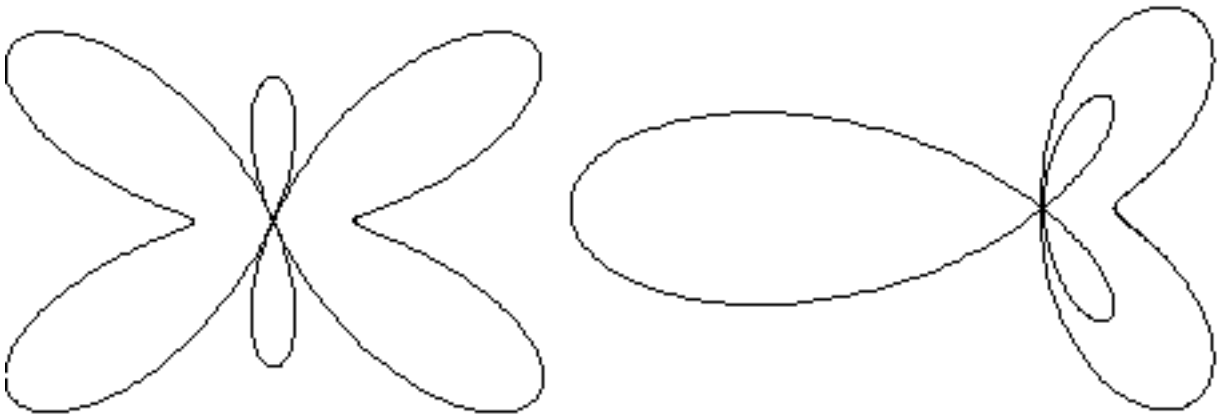
On recommande souvent de paramétrer l'affichage par les dimensions du rectangle l et h où on veut afficher. Alors il faut redéfinir une instruction "point" pour placer dans ce rectangle, un pixel correspondant au point de coordonnées réelles (x, y) dans [a, b]*[c, d]. Mais nous verrons dans des exemples ultérieurs que moyennant une étude du problème des bornes à chaque fois, on peut aller de pixel en pixel.

```
let round x = if x >=. 0. then int_of_float (x +. 0.5) else -(int_of_float (-. x +. 0.5));; (* arrondi *)

let phi x a b l = round (float_of_int l *. (x -. a) /. (b -. a))
and psi y c d h = round (float_of_int h *. (y -. c) /. (d -. c));;

let point x y a b c d l h = moveto (phi x a b l) (psi y c d h)
and trait x y a b c d l h = lineto (phi x a b l) (psi y c d h);;
```

On donne juste l'exemple du tracé d'une courbe en polaire d'équation $\rho = ro(\theta)$ pour $0 < \theta < t_{max}$ en faisant avancer θ d'un "pas". On dessine par exemple, le papillon et la torpille en coordonnées polaires :

$$r = e^{\cos 2t} - 2\cos 4t + \sin^5(t / 12) \quad \text{puis} \quad r = e^{\cos 2t} - 2\cos 3t + \sin^5(t / 20)$$

```
let trace ro a b c d l h pas tmax = open_graph" ";
        let t = ref 0. and r = ro 0. in
        point (r *. (cos !t)) (r *. sin !t) a b c d l h;
        while !t < tmax do
                let r = ro !t in trait (r*.(cos !t)) (r*.sin !t) a b c d l h;
                t := !t +. pas
        done;
        read_key (); close_graph ();;

let rec puiss x n = if n = 0 then 1. else x*.(puiss x (n-1));;
```

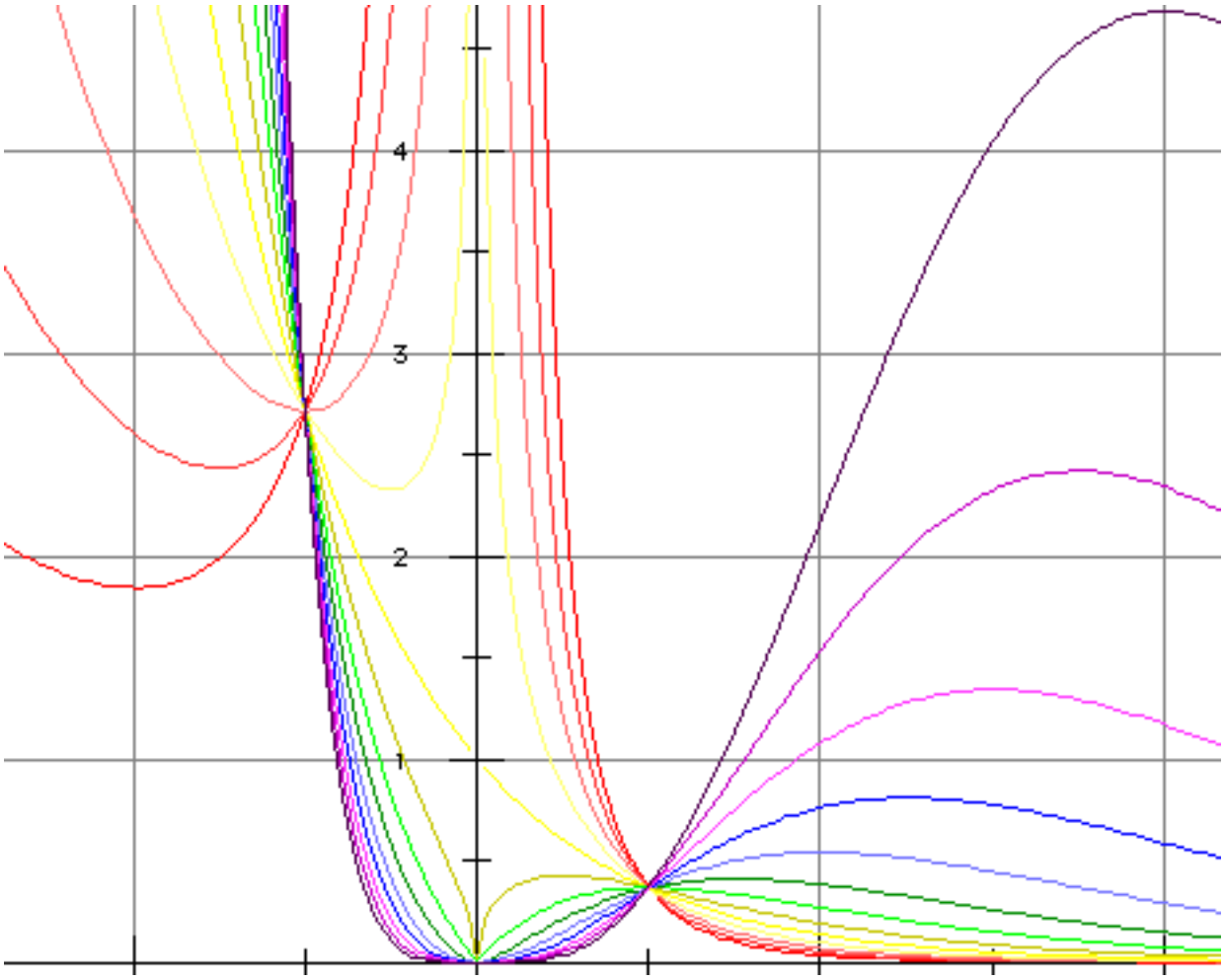


```
trace (fun t -> exp (cos (2. *. t)) -. 2.*.(cos (4. *. t)) +.
                (puiss (sin (t /. 12.)) 5) ) (-3.) 3. (-3.) 3. 250 200 0.01 6.28 ;;

trace (fun t -> exp (cos (2.*.t)) -. 2.*.(cos (3. *. t)) +.
                (puiss (sin (t /. 20.)) 5) ) (-5.) 3. (-3.) 3. 300 200 0.01 6.28 ;;
```

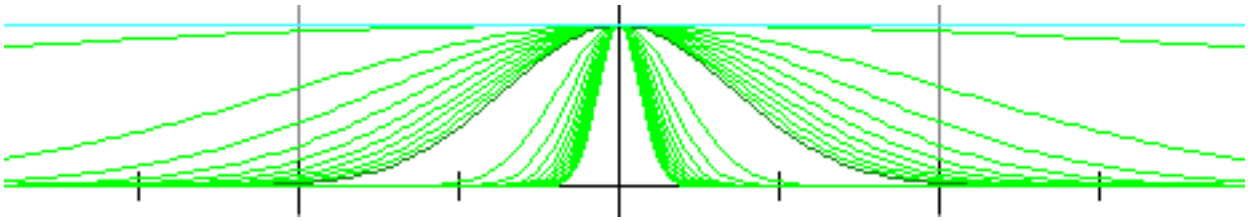
4. Tracé d'une famille de courbes sur un même repère

Avec les mêmes fonctions préliminaires, ce que permet un langage fonctionnel, c'est que la famille de fonctions x -> $f_m(x)$ sera une fonction de deux arguments : x et m, dont on donnera un intervalle [a, b] pour x, [c, d] pour l'image y, et les bornes m_{min}, m_{max}, ainsi que le pas pour le paramètre m.
Ainsi, ci dessous, la famille de fonctions y = exp(-x+mln|x|) pour m de -2 à 4 de 0,5 en 0,5.

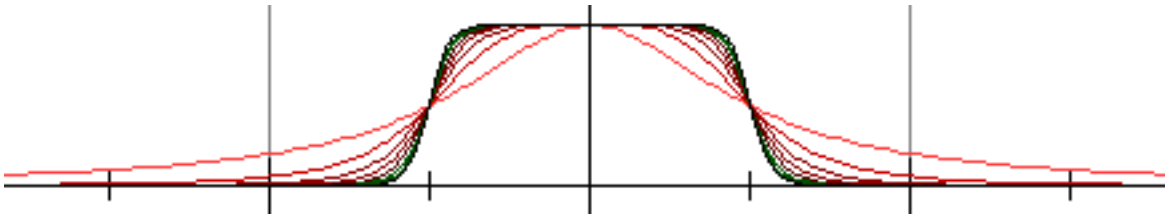
```
let round x = if x >=. 0. then int_of_float (x +. 0.5) else -(int_of_float (-. x +. 0.5));; (* arrondi *)

let phi x a b l = round (float_of_int l *. (x -. a) /. (b -. a))
and psi y c d h = round (float_of_int h *. (y -. c) /. (d -. c));;

let point x y a b c d l h = moveto (phi x a b l) (psi y c d h)
and trait x y a b c d l h = lineto (phi x a b l) (psi y c d h);;

let famille f a b c d l h mmin mmax pas = open_graph" ";
        let m = ref mmin in
        while !m <= mmax do
                let x = ref a in point a  (f a !m) a b c d l h;
                while !x < b do trait !x  (f !x !m) a b c d l h; x := !x  +. 0.05
                done;
                m := !m +. pas
        done;
        read_key (); close_graph ();;
```




```
famille (fun x m -> exp (-.x +. m *. log (abs_float x))) (-1.5) 2.5 0. 5. 400 400 (-2.) 4. 0.5;;
```



famille (fun x m -> exp (-. m*. x *. x)) (-4.) 4. 0. 1. 400 50 0.05 10. 0.4;; (* y = exp (- mx^2) *)

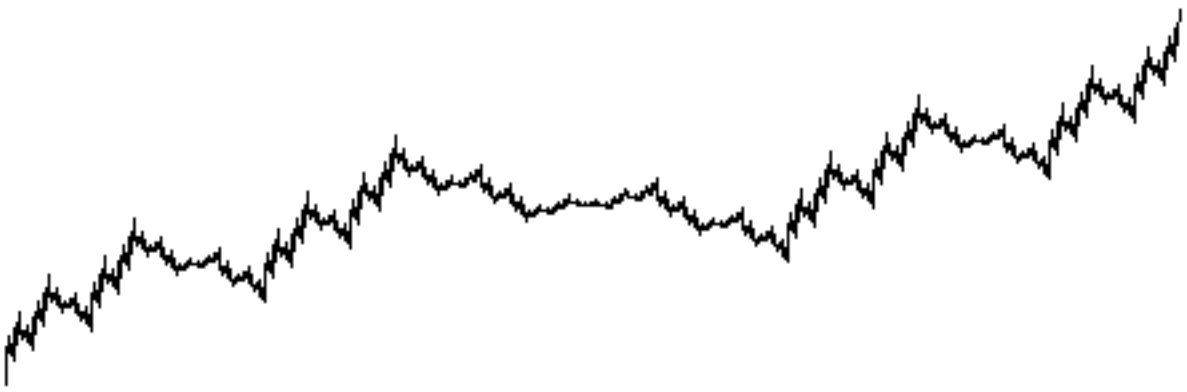


famille (fun x m -> 1. /. (1. +. power (x *. x) m)) (-4.) 4. 0. 1. 400 50 0.1 10. 0.5;;
(* les courbes y = 1 / (1 + x^{2m}) montrent une convergence non uniforme de cette suite de fonctions vers une fonction discontinue *)

5. Escalier de Cantor

Il s'agit d'un contre-exemple célèbre de fonction de [0, 1] dans [0, 1] qui est continue en étant dérivable en aucun point.

```
let rec diable n a b c d = if n = 0 then (moveto a b; lineto c d)
                           else (diable (n - 1) a b ((2*a + c) / 3) ((b + 2*d) / 3);
                           diable (n - 1) ((2*a + c) / 3) ((b + 2*d) / 3) ((a + 2*c) / 3) ((2*b + d) / 3);
                           diable (n - 1) ((a + 2*c) / 3) ((2*b + d) / 3) c d);;

let escalier = open_graph""; diable 7 10 10 450 150 ; read_key (); close_graph ();;
```



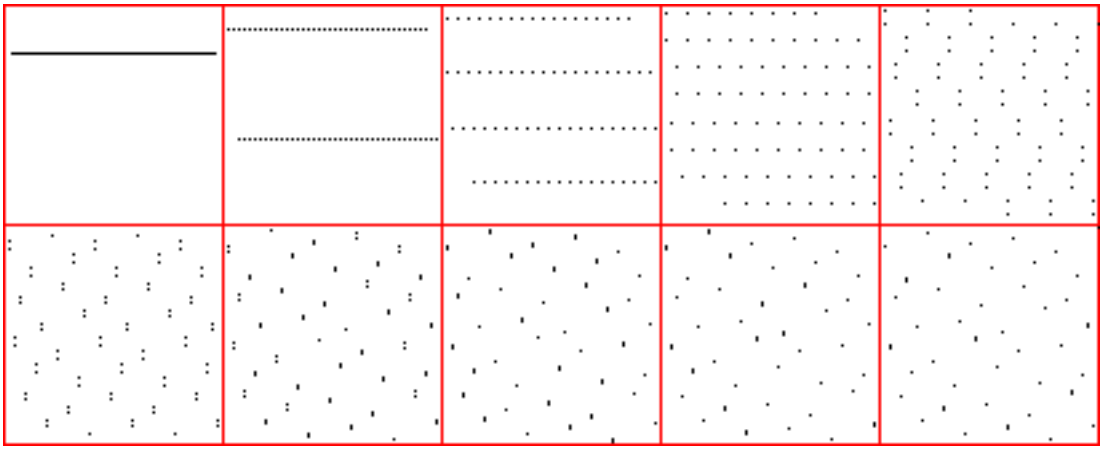
6. Transformation du boulanger

Dans un carré, les points voient leur abscisse doublée et leur ordonnée divisée par deux, puis, ceux qui sont ainsi sortis du carré, sont translatés dans la moitié supérieure du carré.
c'est donc "transfo" ci-dessous et sa réciproque est :

(x', y') = si 1/2 < y' alors ((x'+1)/2, 2y'-1) sinon (x'/2, 2y').

Très vite, en quelques itérations, la répartition devient uniforme, mais la simulation est difficile à cause de la division entière.

```
#open "graphics";;

let voir d =
let transfo (x, y) = (let x' = 2 * x and y' = y / 2 in if x' > d then (x' - d, y' + d / 2) else (x', y'))
and td = make_vect (d - 5) (3, 64)          (* dimension des carrés d < 100 *)
and dessin a b t = moveto a b; set_color red; lineto (a + d) b; lineto (a + d) (b + d);
        lineto a (b + d); lineto a b; set_color black;
        for i = 0 to (vect_length t) - 1 do
                let (u, v) = t.(i) in (moveto (a + u) (b + v); lineto (a + u) (b + v)) done

        in open_graph" ";
        for k = 0 to (vect_length td) - 1 do td.(k) <- (i+k, j) where (i, j) = td.(k) done;
        for k = 0 to 14 do (dessin (d * k mod (5 * d)) (180 - d * (k / 5)) td;
                        for i=0 to (vect_length td) - 1 do td.(i) <- transfo td.(i) done)
        done;
        moveto 20 270; draw_string "Presser une touche ..."; read_key (); close_graph () ;;

    voir 82;; (* on a reproduit les 10 premiers dessins : *)
```

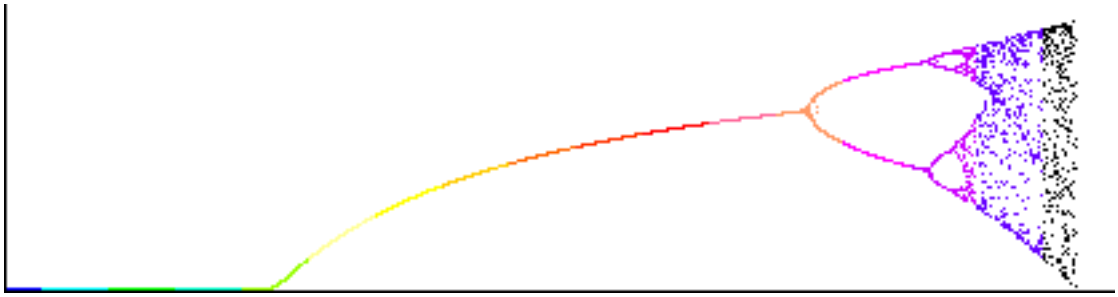


7. Problème chaotique de Coullet-Tresser

On étudie les suites u est définies par récurrence par la relation $u_{n+1} = au_n(1 - u_n)$ avec un premier terme u_0 dans [0, 1], suivant le coefficient a. On définit une fonction "limite", plaçant le 100° point de la suite en ordonnée en fonction de a en abscisse (les coordonnées logarithmiques ne sont pas nécéssaires et aller au delà de 100 ne change pas l'aspect général).

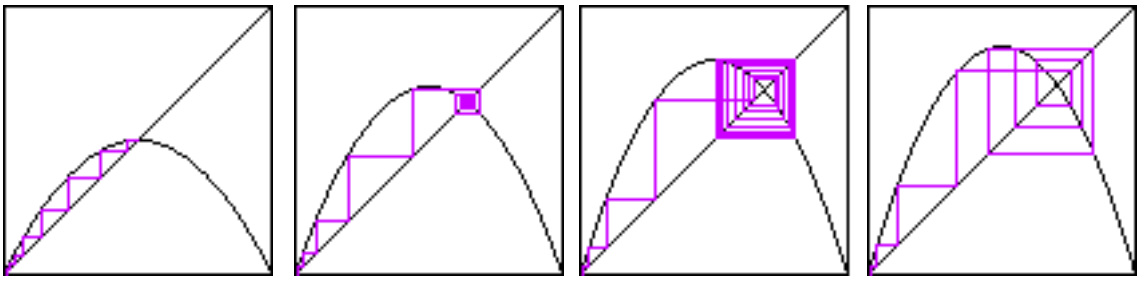
```
let round x = if x >=. 0. then int_of_float (x +. 0.5) else -(int_of_float (-. x +. 0.5));; (* arrondi *)

#open "graphics";;

let coul = [|255; 65535; 65280; 65484; 10092288; 16777100; 16776960;
        16763904; 16737792 ; 16724736; 16711680; 16737945; 16750950;
        16711935; 13369599; 6684927; black|];;

let point u v = moveto u v; lineto u v;;

let limite dep a = let p = ref dep in for i = 1 to 100 do p := a *. (!p)*.(1. -. (!p)) ; print_float !p done;
        point (round (100. *. a)) (round (100. *. (!p)));;

let trace n = open_graph"";
        for i = 1 to n do let a = random__float 4. in
                set_color coul.(round(4.*.a));
                limite (random__float 1.) a done;
        read_key (); close_graph ();;
```

L'observation sur n = 10 000 études de convergence pour a entre 0 et 4, montre que ces suites tendent d'abord vers 0, puis elles sont convergentes vers une limite non nulle jusqu'à a = 3, enfin elles oscillent sur deux points adhérents, puis vers a = 3, 5 sur 4 points adhérents, sur 8 à 3,56, sur 16 à 3,57 enfin deviennent vraiment chaotiques lorsque a s'approche de 4.
Ces changements se font pour différentes valeurs de a, croissant avec le taux 4,669201609102 (de Feigenbaum), général aux études sur le chaos.

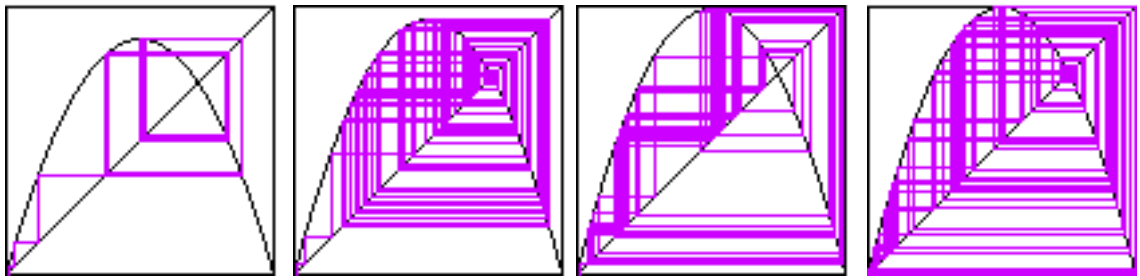


Maintenant, on regarde classiquement les 100 premières valeurs de la suite pour différentes valeurs de a où elle est convergente, ou bien admet deux valeurs d'adhérence (a = 0.8) ou encore quatre pour a = 088, puis la suite devient plus en plus absconse.

```
let feig a = open_graph""; let x = ref 0.01 and d = 100 in let dr = float_of_int d

and suivant x = 4. *. a *. x *. (1. -. x) in
        let phi u = round (dr *. (suivant (float_of_int u /.dr))) in
                set_color black; moveto d d;
                lineto 0 0; lineto d 0; lineto d d; lineto 0 d; lineto 0 0;
                for i = 1 to d do lineto i (phi i) done;
                set_color 13369599;
                moveto (round (dr *. (!x))) 0;
        for i = 1 to 100 do let y = suivant !x  in lineto (round (!x *. dr)) (round (y  *. dr)) ;
        lineto (round (dr  *. y)) (round (dr  *. y)) ; x := y done; read_key (); close_graph ();;
```



Suite de Feigenbaum pour a = 0.5, 0.7, 0.8, 0.85

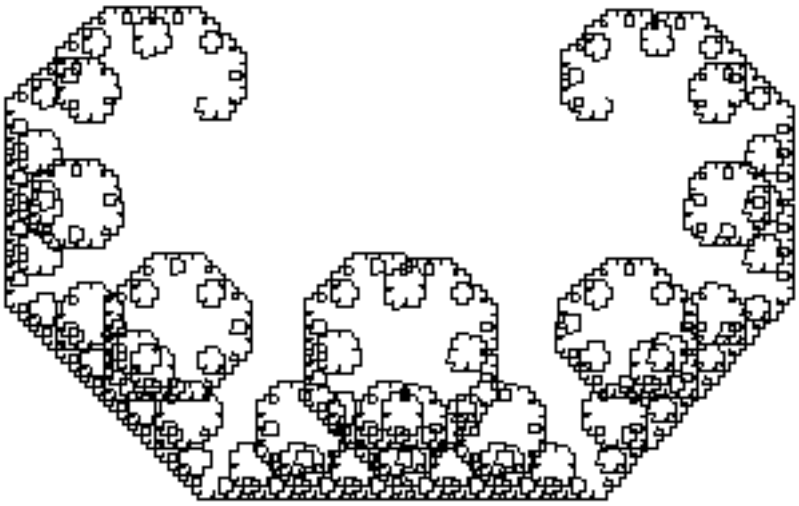


a = 0.88 0.95 0.99 0.999

8. Dragon de Hilbert

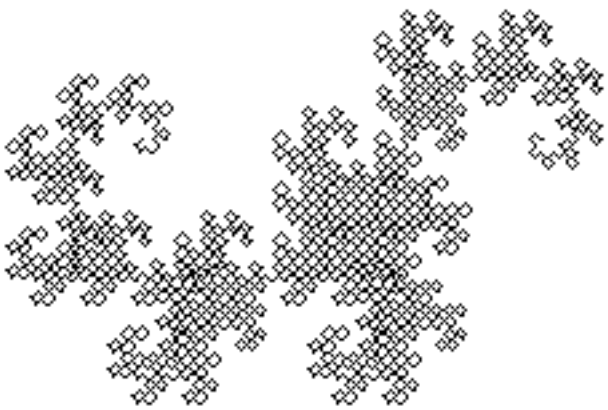
Partant d'un vecteur joignant les points de coordonnées (a, b) et (c, d), on trace le triangle rectangle isocèle à gauche, son sommet est facile à trouver, ce sont les coordonnées (u, v) figurant dans la fonction "dragon". "Dragon"pour la valeur n sur ce segment consiste à tracer "dragon" pour la valeur n - 1 sur les deux côtés de ce triangle. La récursivité s'arrête pour n = 1 où seul le segment considéré doit être tracé.

```
#open "graphics";;

let rec dragon n a b c d =
        if n = 1 then (moveto a b; lineto c d)
                else let u = (a + c + d - b) / 2  and  v = (a + b - c + d) / 2
                        in (dragon (n - 1) a b u v;
                        dragon (n - 1) u v c d);;

let hilbert n = open_graph""; dragon n 150 200 300 200 ; read_key (); close_graph ();;
```



Ci dessus "Hilbert" pour n = 13, mais il suffit d'intervertir les deux dernières coordonnées pour avoir le vrai dragon (ci-dessous exemple de "hilbert 12;;"):

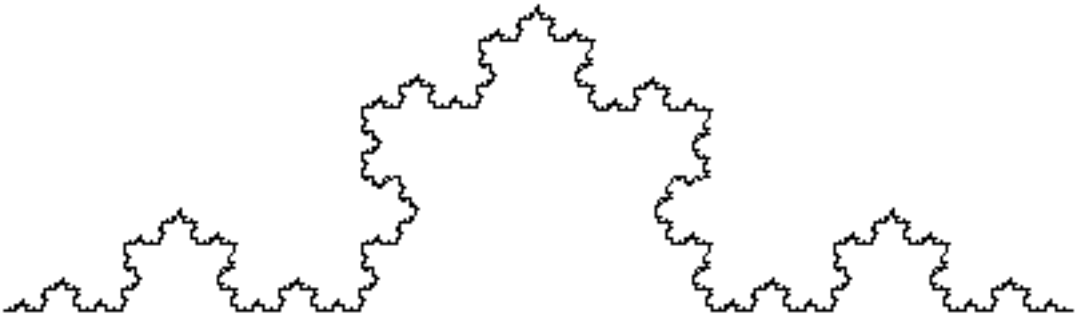
```
let rec vraidragon n a b c d =
        if n = 1 then (moveto a b; lineto c d)
                else let u = (a + c + d - b) / 2 and v = (a + b - c + d) / 2
                        in (vraidragon (n - 1) a b u v;
                        vraidragon (n - 1) c d u v);;

let hilbert n = open_graph""; vraidragon n 150 200 300 200 ; read_key (); close_graph ();;
```



9. Etoile de Von Koch

Pour n = 0, il s'agit d'un segment orienté du point (a, b) vers (c, d). L'étape suivant consiste à diviser ce segment en trois, le second étant remplacé à sa gauche par deux segments mis bout à bout, formant une ligne brisée de 4 segments de même longueur. A l'ordre n+1, cette opération est répétée sur chacun des segments correspondant à l'ordre n.

```
let rec trig n a b c d = if n = 0 then (moveto a b; lineto c d)
        else (trig (n-1) a b ((2*a + c) / 3) ((2*b + d) / 3);          (* 7/2 est environ 2.sqrt(3) *)
        trig (n-1) ((2*a + c) / 3) ((2*b + d) / 3) ((a + c) / 2 - 2*(d -b ) / 7) ((b + d) / 2 + 2*(c - a) / 7);
        trig (n-1) ((a + c) / 2 - 2*(d - b) / 7) ((b + d) / 2 + 2*(c -  a) / 7) ((a + 2*c) / 3) ((b + 2*d) / 3);
        trig (n-1) ((a + 2*c) / 3) ((b + 2*d) / 3) c d);;


let vonkoch n = open_graph""; trig n 10 10 410 10 (* ci-dessous pour n = 7 *);;
```

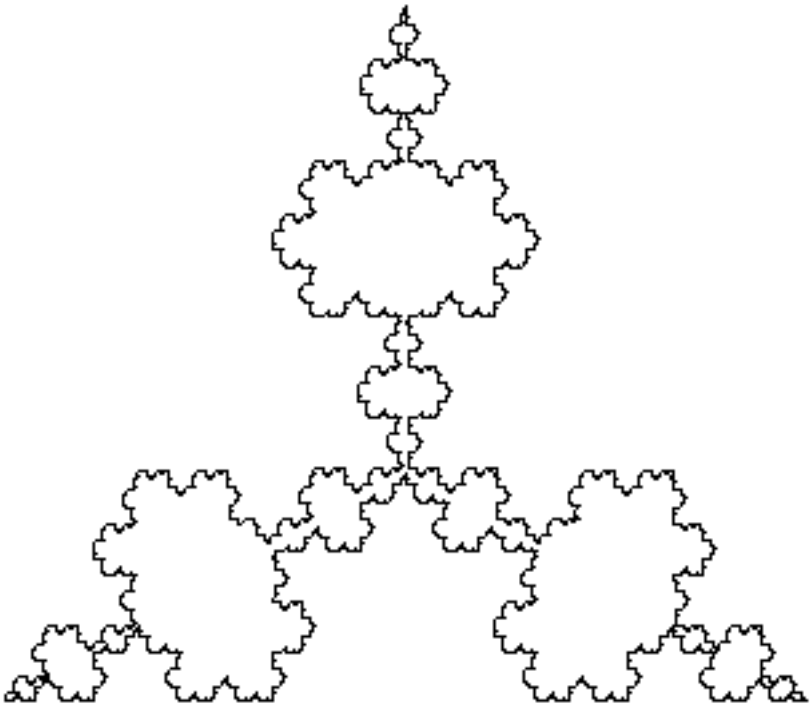


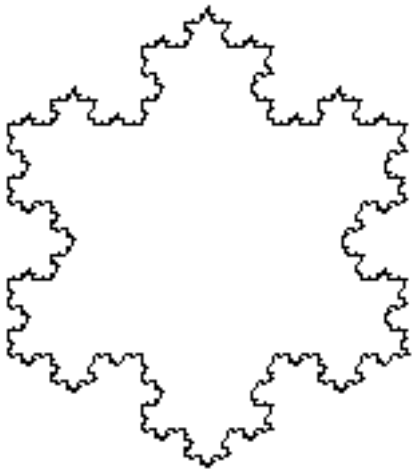
Même chose (3 fois) à l'intérieur d'un triangle équilatéral pour n = 7.

```
let etoileinterieure n = open_graph"";
        trig n 10 10 310 10 ; trig n 310 10 160 270; trig n 160 270 10 10 ;
        read_key (); close_graph ();;
```

Et enfin, avec n = 5 :

```
let  etoilexterieure n =
open_graph"";
trig n 10 50 85 180 ;
trig n 85 180 160 50;
trig n 160 50 10 50 ;
read_key ();
close_graph ();;
```





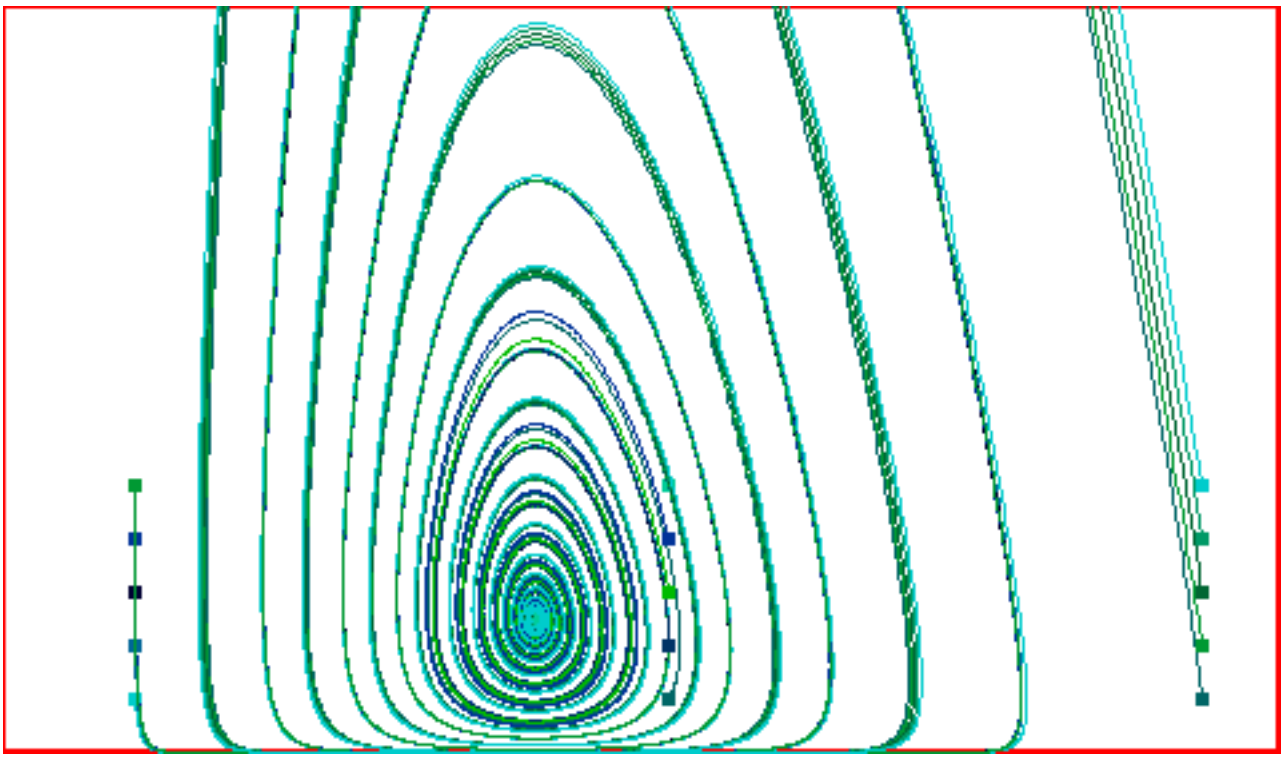
10. Simulation d'un écosystème par des équations de Lokta-Voltera simplifiées

C'est un des premiers systèmes différentiels modèlisant une évolution de deux populations proies - prédateurs. Soient x le nombre de lapins initialement dans un enclos ($0 < x < 500$) et y celui de renards ($0 < y < 10$).
On récurre avec les équations de mise à jour :

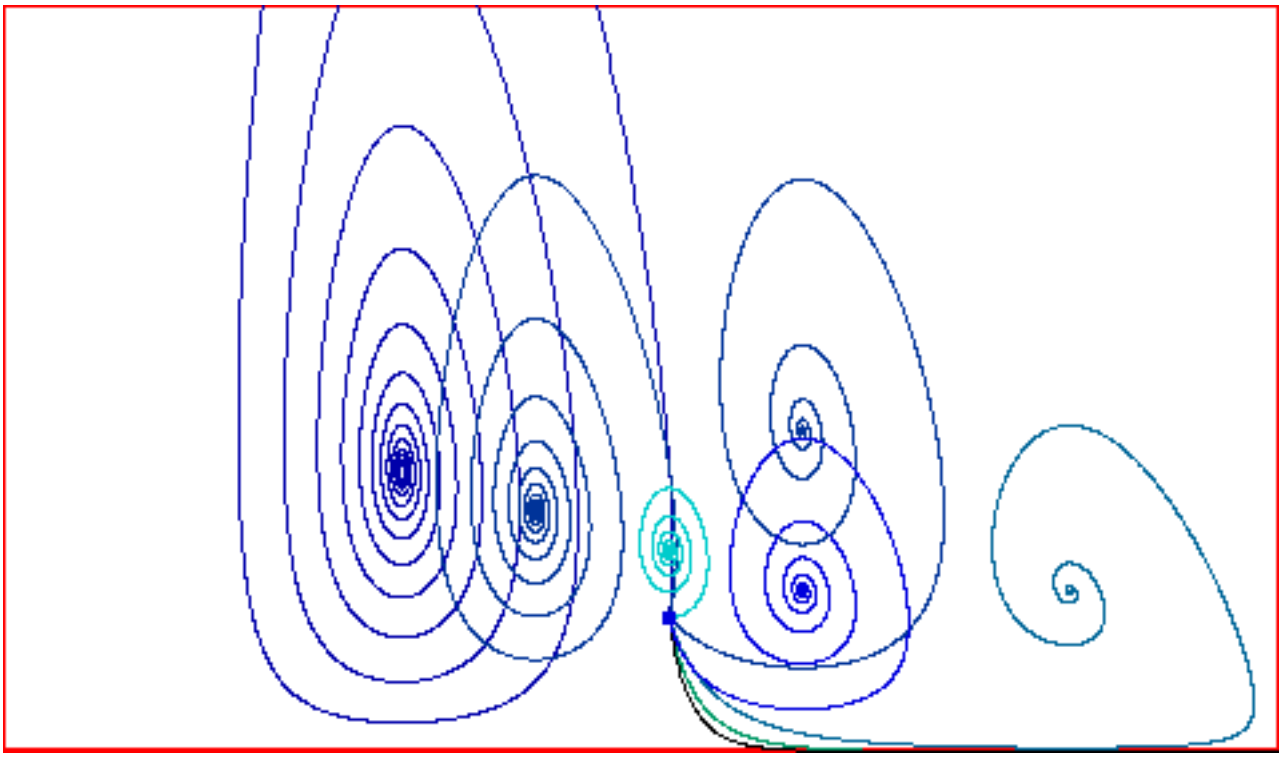
$$x_{n+1} = x_n + gx_n(1 - x_n/500) - ax_ny_n$$
$$\text{et} \quad y_{n+1} = y_n + ax_ny_n - by_n$$

(la fonction "round" est toujours la même).

```
#open "graphics";;

let pnt x y = fill_rect (x - 2) (y - 2) 5 5

and piste x y = 0. < x & x < 500. & 0. < y & y < 300.;;

let dynsys x y g a b nmax = let n = ref 0 and u = ref x and v = ref y and du = ref 2. and dv = ref 2.
        in
        moveto (round x) (round (10. *. y));
        set_color (has 6500 389000);
        pnt (round x) (round (10. *. y));
        while !n < nmax & piste !u !v do
                u :=  !u  +. g *.(!u) *. (1.-.(!u) /. 500.)  -. a *. (!u) *. (!v);
                v := !v  +. a *. (!u) *. (!v) -. b *. (!v);
                lineto (round !u) (round (10. *. (!v)));
                incr n
        done

where has a b = a + random__int (b -  a  + 1);;
```



```
for x = 0 to 2 do for y = 1 to 5 do
    dynsys (50. +. 200. *. (float_of_int x)) (2. *. (float_of_int y)) 0.0025 0.0003 0.06 10000
done done;;
```

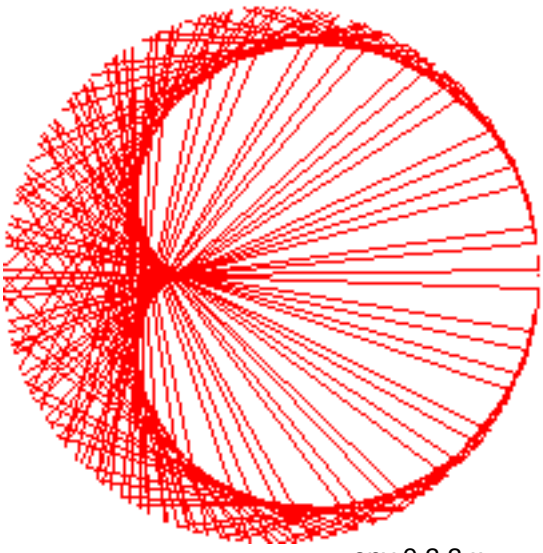


```
for a=1 to 2 do for b=3 to 5 do
    dynsys 250. 5. 0.003 (0.0001 *. (float_of_int a)) (0.01 *. (float_of_int b))10000
done done;;
```

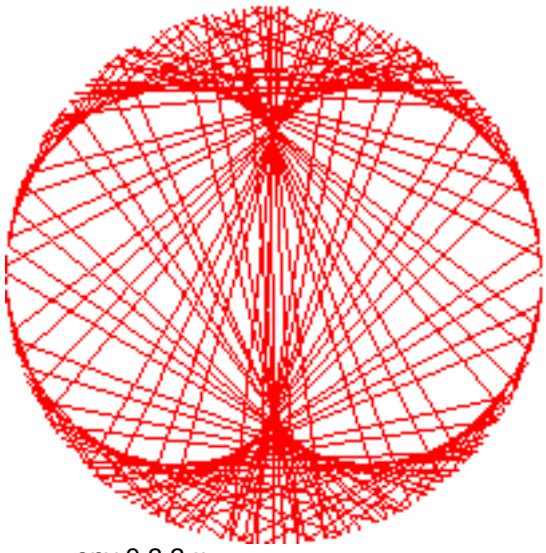
11. Enveloppes de droites

Sur un cercle de rayon r, on joint les points de coordonnées polaires (r, θ) et (r, nθ) en faisant varier l'angle θ de p en p (par exemple p = 5°).
Pour n = 2 on obtient une cardioïde, n = 3 une néphroïde etc...
Pour cet exemple également, on prend les coordonnées réelles de l'écran.

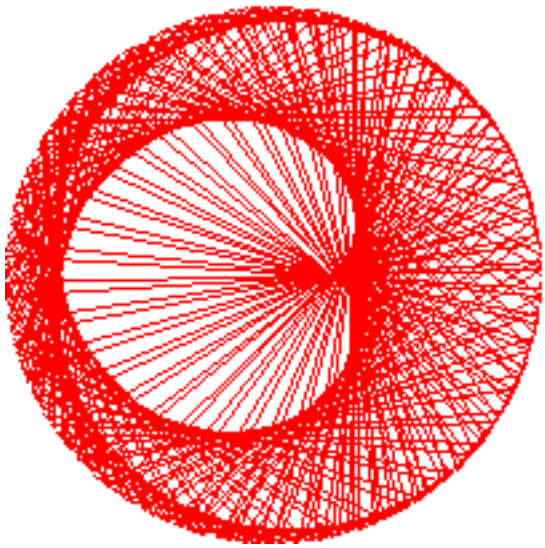
```
#open "graphics";;

let round x = if x >=. 0. then int_of_float(x +. 0.5) else -(int_of_float (-. x +. 0.5)) (* arrondi *) ;;

let place x y  = (* place le point de vraies coord x, y *)
        moveto (100 +  round (100. *. x) ) (100 +  round (100. *. y))

and trace x y = (* trace le segment vers le point de vraies coord x, y *)
        lineto (100 +  round (100. *. x) ) (100 +  round (100. *. y));;

let env p n =     set_color red;
        let a = ref 0. in
        while !a < 8. *. 3.14 do
                place (cos !a) (sin !a) ;
                trace (cos (n *. !a)) (sin (n *. !a));
                a := !a +. p
        done;;


open_graph""; env 0.1 1.5; read_key (); close_graph ();;
```



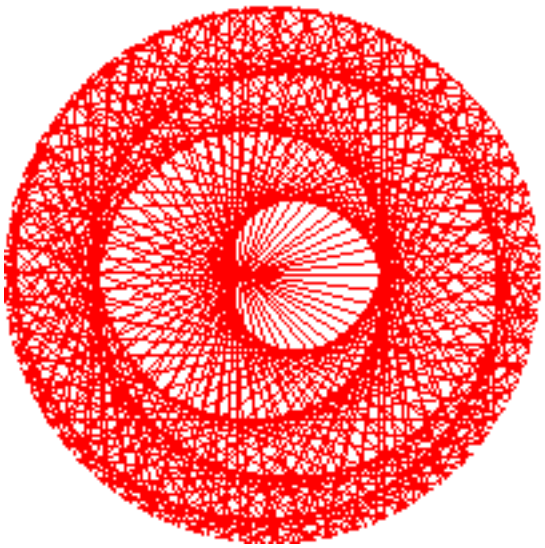
env 0.2 2.;;



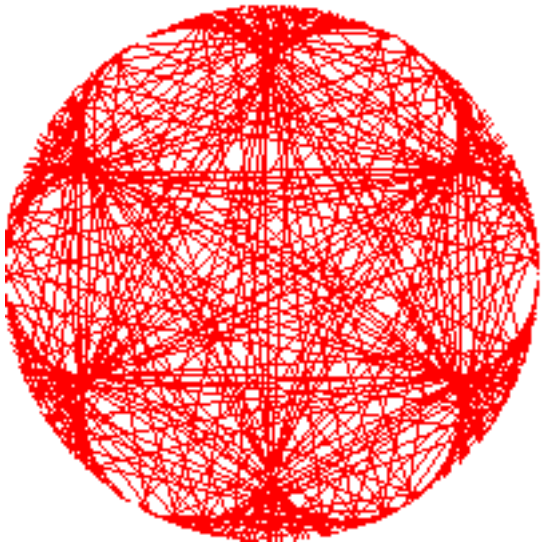
env 0.2 3.;;



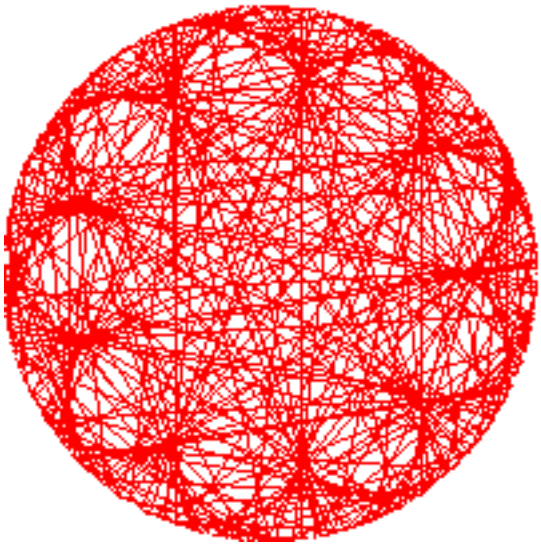
env 0.1 1.5;;



env 0.1 1.2;;



env 0.1 7.;;

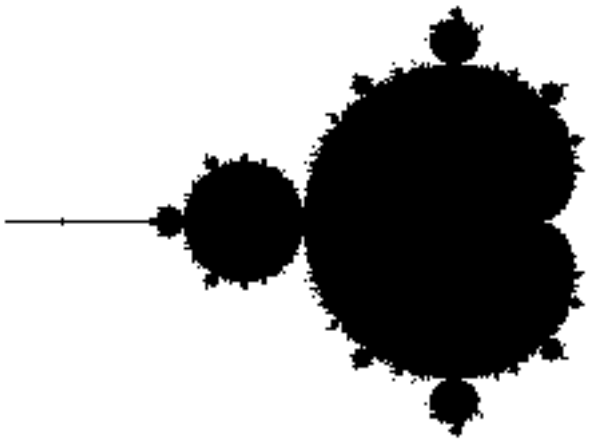


env 0.1 5.5;;

12. Ensemble de Mandelbrot

On cherche l'ensemble des complexes c de $[-2, 1]^2$ tel que l'itération de la suite déterminée par : $z_0 = 0$ et $z_{n+1} = f(z_n) = z_n^2 + c$, ait ses (par exemple) 100 premiers termes dans un cercle de centre O et de rayon 3 (ce qui ne veut pas dire qu'elle converge, ni que cette "convergence" ait lieu pour les autres complexes que 0, c'est l'ensemble des c tels que K_c ensemble de Julia, soit connexe).

```
#open "graphics";;
let point x y = moveto x y; lineto x y
and contract n = (float_of_int n) /. 90. -. 2.;; (* fait la modification linéaire [0, 270] -> [-2, 1] *)

let converge x y = (* indique si la suite z_{n+1} = z_n^2+x+iy converge, le départ étant toujours 0 *)
        let x0 = ref 0. and y0 = ref 0. and x1 = ref 0. and y1 = ref 0. and it = ref 1 in
        while abs_float !x0 +. abs_float !y0 <. 3. && !it < 100 do
                incr it ; x0 := !x1; y0 := !y1; x1:= x +. !x0 *. !x0 -. !y0 *. !y0; y1 := y +. 2. *. !x0 *. !y0
        done;
        !it = 100;;

let mandelbrot () =
        for x = 0 to 270 do for y = 0 to 270 do
                if converge (contract x) (contract y) then point x y
        done done;;

open_graph""; mandelbrot (); read_key (); close_graph ();;
```



13. Ensembles de Julia

Il s'agit des frontières entre les parties du plan où il y a convergence ou divergence pour les mêmes suites du type $z_0 = 0$ et $z_{n+1} = f(z_n) = z_n^2 + c$.
La fonction "point" est la même. On modifie cependant la fonction précédente "converge" de façon à pouvoir partir d'un $z_0 = x_0 + iy_0$. Puis dans le carré $[-2, 2]^2$, on cherche à représenter les points de départ de la suite où il y a convergence.

Voir : B.Mandelbrot *Les objets fractals* Flammarion 1975
J.Crutchfield *Le chaos* Pour la science, février 1987

```
let  contract n = (float_of_int n) /. 60. -. 2.  (* fait la modification linéaire [0, 240] -> [-2, 2] *)
and converge x0 y0 x y = (* indique si la suite z_0 = x_0 + y_0 et z_{n+1} = z_n^2+x+iy converge *)
        let x1 = ref x0 and y1 = ref y0 and x2 = ref x0 and y2 = ref y0 and it = ref 1 in
        while abs_float !x1 +. abs_float !y1 <. 3. && !it < 100 do
        incr it ; x1 := !x2; y1 := !y2; x2 := x +. !x1 *. !x1 -. !y1 *. !y1; y2 :=  y +. 2. *. !x1 *. !y1
        done; !it = 100;;
```

```
let julia xc yc = for x = 0 to 240 do for y = 0 to 240 do
                                if converge (contract x) (contract y) xc yc then point x y
                  done done;;
```

```
open_graph""; set_color  blue; julia 0.31 0.04; read_key (); close_graph ();;
```

```
julia 0.3 0.025;
```

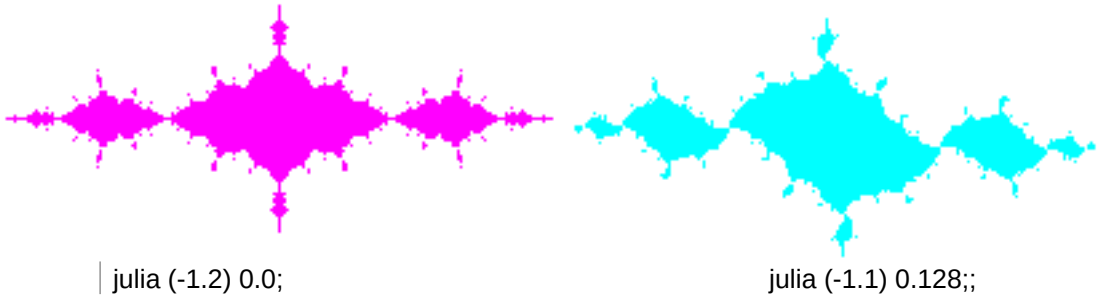
```
julia (-0.8) (-0.1);;
```

```
julia 0.31 0.04;;
```

```
julia (-0.1225) 0.74493;;
```

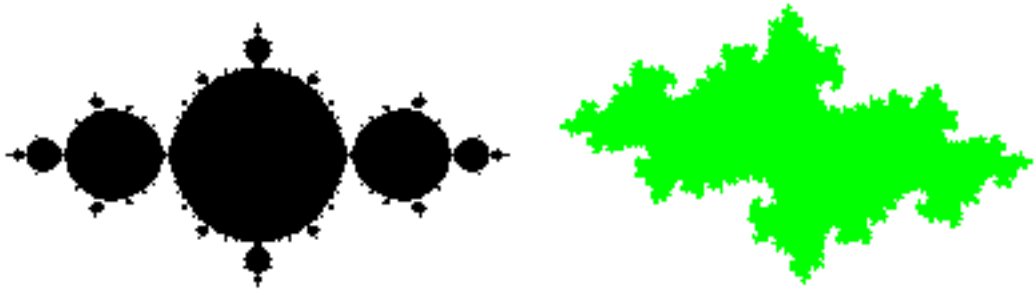
```
julia 0.32 0. 43;;
```

```
julia (-0.1134) 0.8606;;
```



```
julia (-1.2) 0.0;
```

```
julia (-1.1) 0.128;;
```



```
julia (-0.9) 0.0;;
```

```
julia (-0.7) 0.2;;
```

Pour en avoir systématiquement, il suffit d'une double boucle avec pas mal de temps pour n'en retenir que quelques-unes.

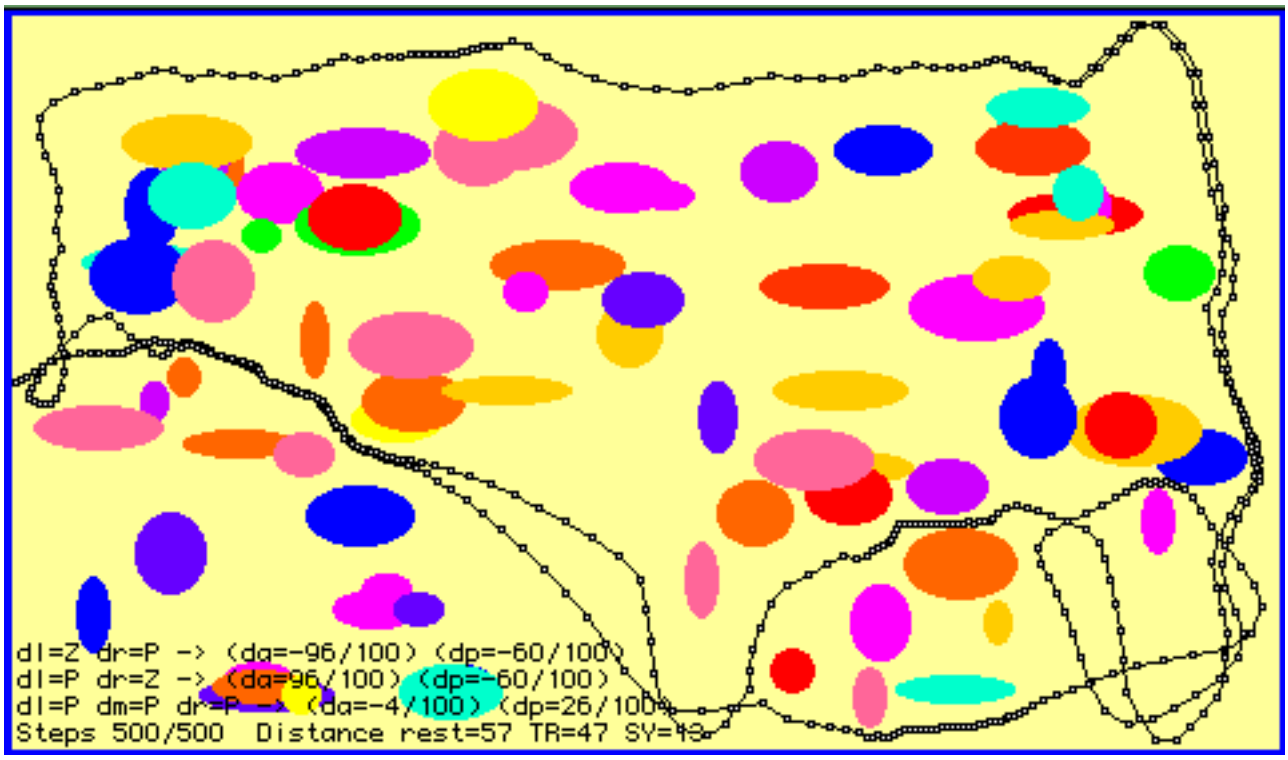
14. Parcours d'obstacles au moyen de règles floues

Un robot disposant de 3 capteurs, frontal et deux autres disposés à ± ouv reconnait les obstacles jusqu'à une distance dm.

```
let rec nth (a :: q) = fun 0 -> a | n -> nth q (n - 1)
and ret x = fun [] -> [] | (a :: ll) -> (if x = a then ll else a :: (ret x ll));;
let hasard ll = nth ll (random__int (list_length ll))
and has a b = a + random__int (b -  a + 1);;
```

Pour le contrôleur flou (problème VIII 3), on définit une règle comme 3 prédicats (qualifiant les trois distances mesurées à chaque instant) et deux réels, les conclusions concernant le changement de direction et de vitesse.

```
type pred = ANY | N | Z | P | NB | NS | ZE | PS | PB;;
type regle = {c1 : float; c2 : float; h1 : pred; h2 : pred; h3 : pred};;

let opp = fun NB -> PB | NS -> PS | N -> P
and pente p = if mem p [NB; PB; NS; PS; ZE] then 0.5 else 1.;;

let rec sommet p = if p = PS then 0.5
        else if p = PB or p = P then 1.
        else if p = ZE or p = Z then 0.
        else -. (sommet (opp p));;

let max x y = if x <. y then y else x and min x y = if x <. y then x else y ;;

let rec mu p x = if mem p [N; NB; NS] then mu (opp p) (-. x)
        (* valeur du prédicat p pour un x dans [-1, 1] *)
                else if (mem p [P; PB] & 1. <.  x) or (p = ANY) then 1.
                else max 0. ((pe -. abs_float (x -. (sommet p))) /. pe) where pe = pente p ;;
let sat r x y z = min (mu r.h1 x) (min (mu r.h2 y) (mu r.h3 z));;
        (* renvoie le degré de satisfaction de la règle r par (x, y, z) *)

let fuzzy lr x y z = fuzzy' 0. 0. 0. lr (* sc somme des coefficients de satisfaction des règles *)
        where rec fuzzy' s1 s2 sc =
                fun     | [] -> if sc = 0. then (0., 0.) else (s1 /. sc, s2 /. sc)
                        | (r :: q) -> let m = sat r x y z
                                        in fuzzy' (s1 +. m*.r.c1) (s2 +. m*.r.c2) (sc +. m) q;;
let reg p q r u v = {c1 = u ; c2 = v; h1 = p; h2 = q; h3 = r};;
        (* forme une règle, exemple donné sur la figure ou encore  :*)

let rec nbsymbol = fun [] ->0 | (r :: l) ->2 + f(r.h1) + f(r.h2) + f(r.h3) + nbsymbol l
        where f = fun ANY ->0 | _ ->1;;

let sym r = {c1 = -. r.c1; c2 = r.c2; h1 = r.h3; h2 = r.h2; h3 = r.h1};;
let rec complete = fun [] -> []
        | (r :: l r) -> if r.h1 = r.h3 then r ::  (complete lr) else r :: (sym r) :: (complete lr);;

#open "graphics";;
let rec visu y = fun [] -> () | (r :: q) -> moveto 5 y; draw_string ((f "dl=" r.h1)^ (f " dm=" r.h2)^
(f " dr=" r.h3) ^ " -> (da=" ^ (string_of_int (round (100.*. r.c1))) ^ "/100) (dp=" ^ (string_of_int (round
(100. *.  r.c2))) ^ "/100)"); visu (y  + 10) q where f s = fun ANY -> "" | N -> s^"N" | Z -> s^"Z"| P ->
s^"P";;

let prval (x, y, z) = print_string ("("^(string_of_int x)^", "^(string_of_int y)^", "^(string_of_int z)^") ");;

let rec aff = fun [] -> ()
        | (r :: q) -> print_string ((f "dl=" r.h1) ^ (f " dm=" r.h2) ^ (f " dr=" r.h3) ^ " -> (da=" ^
        (string_of_int (round (100. *. r.c1))) ^ "/100) (dp=" ^ (string_of_int (round (100. *.  r.c2))) ^
        "/100)"); print_newline(); aff q
        where f s = fun ANY -> "" | N -> s^"N" | Z -> s^"Z"| P -> s^"P" | _ -> "";;
```



Exemple de simulation avec une liste de règles lr, laquelle est affichée grâce à "visu".

Simulation d'un evitement d'obstacles, les pas min et max, la dist. max de reconnaissance, ouv. capteurs et virage max, nb d'obstacles et nb de pas (le moniteur est en 256 couleurs).

```
let pi = 3.14159 and  fond = 16777100 and verif = 150;; (* 2 couleurs *)
let (ds, pm, dm) = (2., 15., 35.) and (ouv, am, acc) = (pi /. 4., pi /. 3., 10.)
and (no, km) = (100, 200);;

let efface () = set_color fond; fill_rect 0 0 (size_x ()) (size_y ());;
let cadre () = set_color blue; set_line_width 3; moveto 0 2; lineto (size_x () - 2) 2;
        lineto (size_x () - 2) (size_y ()); lineto 0 (size_y ()); lineto 0 2; set_line_width 1;;

let initobs n = for i = 1 to n do set_color (hasard [blue; red; green; yellow; magenta;
                    16737945; 16724736; 16737792; 16763904; 13369599; 6684927; 65484]);
        fill_ellipse (has 30 450) (has 20 250) (has 5 25) (has 5 15) done;;

let point x y = fill_rect (x - 1) (y - 1) 3 3; set_color white; fill_rect x y 1 1;;

let piste x y = let b = 0 < x & x < (size_x ()) & 0 < y & y < (size_y ())
        in if b then (let c = point_color x y in c < verif or fond - 1 < c ) else false ;;
let dist x0 y0 a = let d = ref 0. and (x, y) = (ref x0, ref y0)
        (* distance du point courant à un l'obstacle dans la direction a  *)
        in
while (!d <. dm) & piste !x !y do x := !x + (round (ds *. cos a)); y := !y + (round (ds *. sin a));
                                d := !d +.ds done;
        !d /. dm;; (* le résultat est donné relativement dans [0, 1] *)
```

Simulation donnant la performance, à savoir le triplet formé par la distance restante, la tenue de route et le nombre de symboles significatifs décrivant le jeu de règles (DR, TR, SY). Le seul paramètre de la fonction "simul" est un jeu de règles tel que r1, r2 ou r3. Ce programme est issu d'un projet plus vaste dans lequel les règles sont apprises par optimisation des trois critères énoncés.

La tenue de route est un simple coefficient entre 0 et 100, indiquant que plus on tourne, plus on ralentit.

```
let troute da pas = let a = abs_float da and p = (pas -. ds) /.(pm -. ds) (* tenue de route *)
        in abs_float (mu Z a -. mu P p) +. abs_float (mu P a -. mu Z p);;

let simul lr = open_graph""; efface(); cadre(); initobs no; set_color black; visu 15 lr;
        let (dr, tr, k) = (ref 0., ref 0., ref 1) and pas0, pas1 = ref ds, ref ds
        and (x0, y0, x1, y1, dir0, dir1) = (ref 4, ref (size_y() /2), ref 4, ref (size_y() /2), ref 0., ref 0.)
        in while (!k < km) & piste !x1 !y1  & !x0 < size_x() - 15 do
                (*  prise de données et calcul par commande floue *)
                let  (da, dp)  =  fuzzy  lr             (dist !x1 !y1 (!dir1 +. ouv))
                                                (dist !x1 !y1 !dir1)
                                                (dist !x1 !y1 (!dir1 -. ouv)) (* mesure des 3 distances *)
                in moveto !x0 !y0; set_color black; lineto !x1 !y1; point !x0 !y0;
                        x0 := !x1; y0 := !y1; dir0 := !dir1;
                dir1 := !dir1 +. (da*.am); (* en radian *)
                pas1 := max ds (min pm (!pas0 +. acc*.dp)) ;
                x1 := ! x1 + (round (!pas1 *. (cos !dir1)));
                y1 := !y1 + (round (!pas1 *. (sin !dir1)));
                moveto !x1 !y1;
                tr := !tr +. troute da !pas1;
                pas0 := !pas1; dr := !dr +. !pas1 ; incr k
        done;
        set_color black; point !x0 !y0;
        let DR, TR, SY = size_x() - !x0, round(50.*. !tr) / !k, nbsymbol lr
        in moveto 5 5; set_color black; draw_string ("Steps " ^ (string_of_int !k) ^ "/" ^
                (string_of_int km) ^ "  Distance rest=" ^ (string_of_int DR) ^ " TR=" ^
                (string_of_int TR) ^ " SY=" ^ (string_of_int SY));
 read_key (); close_graph (); (DR, TR, SY);;
```

Exemple de règles :

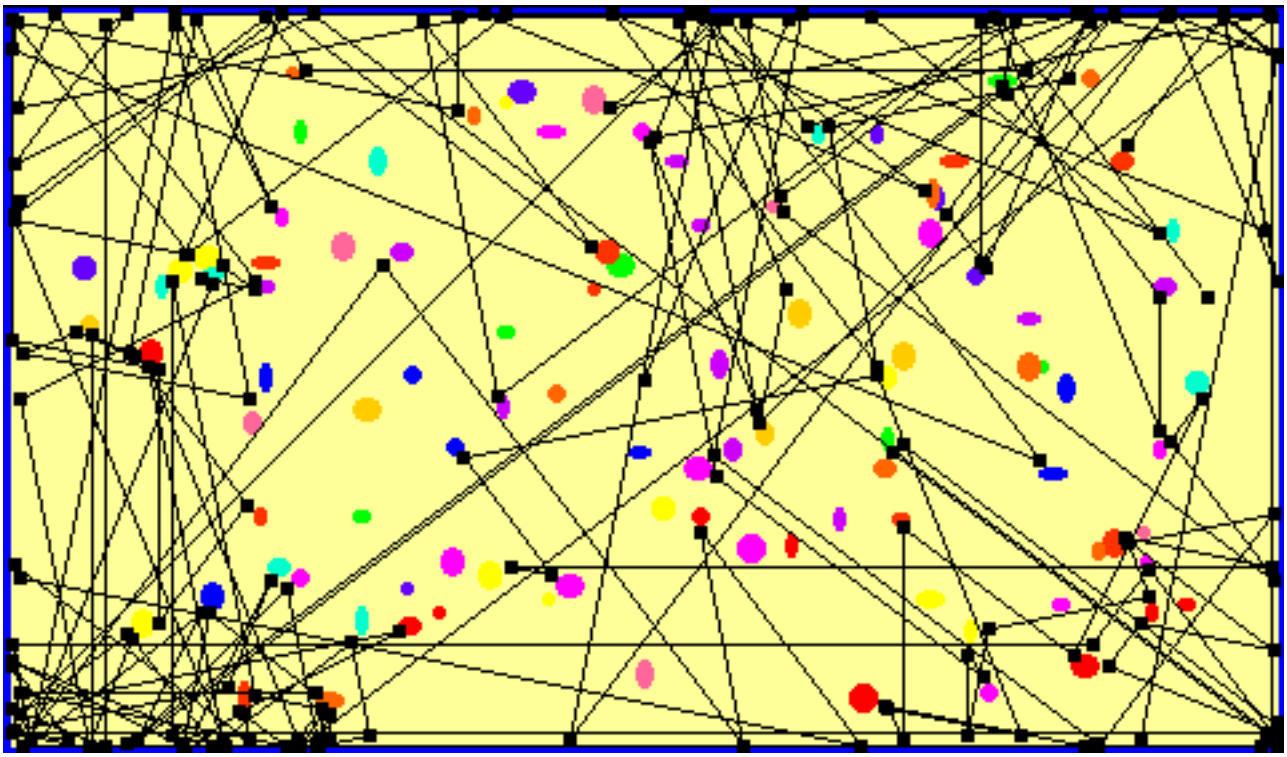
```
let r1 = [reg ANY Z ANY 1. (-1.); reg ANY P ANY 0. 1.; reg ANY ANY Z 1. (-1.)];;
let r2 = [reg P P P (-0.04) 0.26; reg P ANY Z 0.96 (-0.6);reg ANY Z ANY 0.7 (-0.94) ];;
let r3 = [{c1 = 0.980406474734; c2 = -0.19526600857; h1 = P; h2 = Z; h3 = Z};
        {c1 = 0.454257716303; c2 = -0.850182677228; h1 = P; h2 = P; h3 = Z};
        {c1 = 0.312271308714; c2 = 0.640419212152; h1 = P; h2 = ANY; h3 = P}];;
```

15. Flipper fou

Lancé au centre, une bille rebondit en variant sa direction au hasard lorsqu'elle rencontre les bords ou un obstacle (les fonctions nth, hasard, has, round, efface, cadre, piste sont les mêmes que précédamment).

```
#open "graphics";;
let fond =16777100 and verif =150 and pi = 3.14159 and ds, dm, no, ouv = 5., 40., 100, pi /. 12.;;

let pnt x y = fill_rect (x-2)(y-2) 5 5;;
let angle (x, y) (x', y') = aux (x'-x) (y'-y)
        where rec aux x y = if x = 0 then (if y < 0 then -. pi/.2. else pi/. 2.)
                        else if x < 0 then (if y > 0 then pi +. aux (-x) (-y) else aux (-x) (-y) -. pi)
                        else atan ((float_of_int y) /. (float_of_int x));;
let sqr x = x*.x ;;
let dist (x, y) (x', y') = sqrt (sqr (float_of_int (x - x')) +. sqr (float_of_int (y - y')));;

let rec detprinc a = if a < -.pi then detprinc (a +. 2. *. pi)
        else if pi < a then detprinc (a -. 2.*.pi) else a;;
        (* détermination principale entre -π et π d'un angle a *)
let obs = make_vect no (0, 0);; (* tableau formé par no obstacles *)
```

```
let initobs n = for i = 0 to n - 1 do (* au hasard dans une sélection de couleurs *)
        set_color (hasard [blue; yellow; magenta; red; green; 16737945; 16724736; 1
                                6737792; 16763904; 13369599; 6684927; 65484]) ;
        let x = has 30 450 and y = has 20 255 in
                obs.(i) <- (x, y);
                fill_ellipse x y (has 2 5) (has 2 5)
        done;;

let bord x0 y0 a = let d = ref 0. and (x, y) = (ref x0, ref y0) in
        (* distance du point à un obstacle dans la direction a  *)
while (!d <. dm) & piste !x !y do x := !x + (round (ds *. cos a)); y := !y  + (round (ds *. sin a));
        d := !d +. ds done; !d;;

let obstacle (x0, y0) a0 = let d = ref dm and a = ref 0.
                (* renvoie les réels angle et distance au plus proche obstacle *)
        in for i = 0 to no - 1 do let d' = dist (x0, y0) obs.(i) in
        if d' < !d then let a' = detprinc ((angle (x0, y0) obs.(i)) -. a0) in
                if abs_float a' < ouv then (d := d'; a := a') done;
        let dd = bord x0 y0 (a0 -. ouv) in if dd < !d then (d := dd; a := a0 -. ouv);
        let dm = bord x0 y0 a0 in if dm < !d then (d := dm; a := a0);       (* distance frontale *)
        let dg = bord x0 y0 (a0 +. ouv) in if dg < !d then (d := dg; a := a0 +. ouv); (!a, !d);;

let flipper vie = open_graph""; efface(); cadre(); initobs no; set_color black;
let dir = ref (random__float (2.*.pi) -. pi) and x = ref (size_x ()) / 2) and y = ref((size_y ()) / 2)
        in pnt !x !y; moveto !x !y;
        for s = 1 to vie do let (ao,dob) = obstacle(!x, !y) !dir in
                let x', y' = !x + (round (ds *. (cos !dir))), !y + (round (ds *. (sin !dir))) in
                if piste x' y' then (lineto !x !y; if dob < ds or not piste !x !y
                                                        then pnt !x !y; x := x'; y := y')
                else (pnt !x !y; dir := random__float (2. *. pi) -. pi ) done;
        pnt !x !y; read_key (); close_graph () ;;

        flipper 5000;;
```



16. Simulation d'une tortue

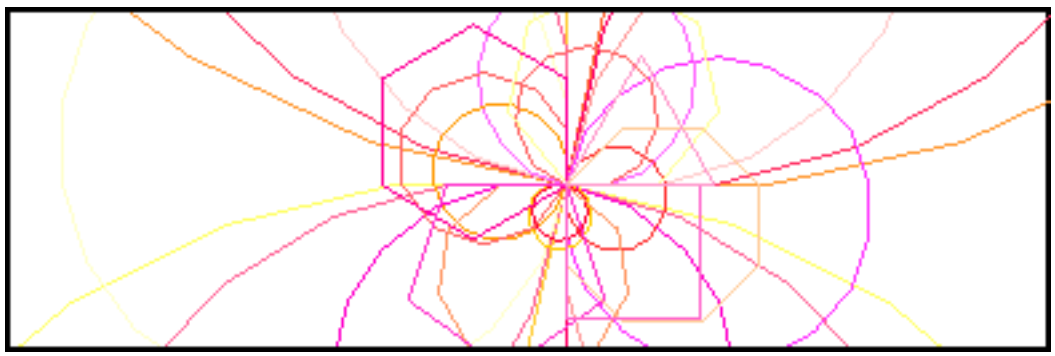
Nous construisons un type nommé "état" composé d'un point courant (abscisse et ordonnée) et d'une direction courante avec un booléen indiquant si on dessine ou non.

Un seul objet de ce type (il pourrait y en avoir plusieurs) va nous servir, grâce à des ordres d'avancer "av" ou de tourner "vire", on peut lui indiquer tout un déplacement, et lui faire faire notamment suivre des courbes définies en équation intrinsèque.

```
#open "graphics";;
let pi = 3.14159;;

let fond = 16777150;;

let round x = if x >=. 0. then int_of_float(x +. 0.5) else -(int_of_float (-. x +. 0.5));;

let efface () = set_color fond; fill_rect 0 0 (size_x ()) (size_y ());;

type etat = {mutable x : float; mutable y : float; mutable dir : float; mutable levé : bool};;

let robo = {x = 240.; y = 140.; dir = (pi /. 2.); levé = false};;

let centre () = robo.x <-  240.; robo.y <- 140.; robo.dir <- (pi /. 2.); robo.levé <- true;
                           moveto (round robo.x) (round robo.y); robo.levé <- false;;

let rec vire a = robo.dir <- (robo.dir +. a) (* en radian *)
and droite () = vire (-. pi  /. 2.)
and gauche () = vire (pi  /. 2.);;

let av d = let dx = d *. cos(robo.dir) and dy = d *. sin(robo.dir)
        in robo.x <-  (robo.x  +. dx); robo.y <- (robo.y  +. dy);
        if robo.levé  then moveto (round robo.x) (round robo.y)
        else lineto (round robo.x) (round robo.y);;
```

17. Application au dessin de polygones

```
let polygone n c = for i = 1 to n do av c;
        vire (2. *. pi  /. (float_of_int n))
        done ;;

let cadre () = set_color white; fill_rect 0 0 (size_x ()) (size_y ()); set_color black; set_line_width 2;
        moveto 30 5; lineto 420 5; lineto 420 275; lineto  30 275; lineto 30 5; set_line_width 1;;

let essai () = open_graph""; efface (); cadre (); centre (); set_color  magenta;
        for i = 30 downto 3 do   set_color (16776960-10000 * i);
                                 polygone i (70. -. 5. *.(float_of_int i)); gauche() done;
        read_key (); close_graph ();;
```

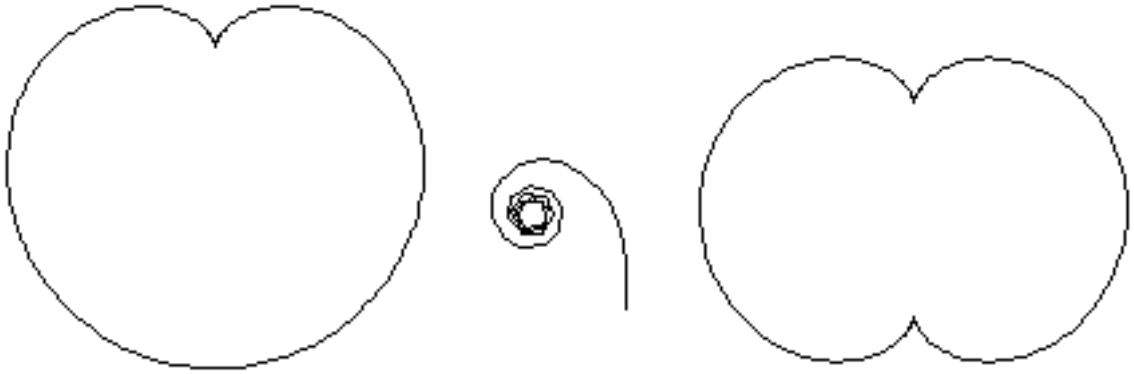


18. Retour sur l'étoile de Von Koch avec la tortue

```
let rec motif n c = if n = 0
                    then av  c
                    else     (motif (n - 1) (c /. 3.);vire (pi /. 3.);
                              motif (n - 1) (c  /. 3.); vire (-.2. *. pi /. 3.);
                              motif (n - 1) (c /. 3.); vire (pi /. 3.); motif (n - 1) (c /. 3.));;

let koch () = open_graph""; centre (); robo.levé <- true; vire(pi /. 2.); av 100.; vire (-.pi);
          robo.levé <- false; motif 5  300.; read_key(); close_graph();;
```

19. Courbes cycloïdales et Spirale de Cornu

L'équation intrinsèque de la cardioïde est ds = k.sin (a / 3), celle de la néphroïde ds = k.sin (a / 2). Pour la spirale de Cornu, chaque avancée d'un pas (ici ds = 5) provoque un virage de da qui doit être proportionnel à la distance déjà parcourue.



```
let i = 0.05;; (* c'est l'incrément choisi en radian *)

let rec cardioide a = if a <. 3. *. pi then (av (4.*.sin(a /. 3.)); vire i; cardioide (a  +. i))
and nephroide a = if a <. 4. *. pi then (av (3.*.sin(a /. 2.)); vire i; nephroide (a  +. i));;

let m = 0.005;; (* facteur de proportionnalité *)

let rec cornu d a = if d <. 300. then
                          (vire da; av 8.; cornu (d  +. 8.) (a  +. da) where da = m *. d);;
```

Pour le dessin du milieu, on lancera donc :

```
let essai () = open_graph""; centre (); cornu 0. 0.; read_key (); close_graph ();;
```