```
/*

 .-------. .-./`)     ,-----.    .--.   .--.      ____
 \  _(`)_ \\ .-.')  .'  .-,  '.  |  | _/  /    .'  __ `.
 | (_ o._)|/ `-' \ / ,-.|  \ _ \ | (`' ) /    /   '  \  \
 |  (_,_) / `-'`"`;  \  '_ /  | :|(_ ()_)     |___|  /  |
 |   '-.-'  .---. |  _`,/ \ _/  || (_,_)   __     _.-`   |
 |   |      |   | : (  '\_/ \   ;|  |\ \  |  |.'   _     |
 |   |      |   |  \ `"/  \  ) / |  | \ `'   /|  _( )_   |
 /   )      |   |   '. \_/``".'  |  |  \    / \ (_ o _) /
 `---'      '---'     '-----'    `--'   `'-'   '.(_,_).'


     .......     * *       °°°  PIOKA



 Dessins génératifs pour Flux (flux.bzh)
 Quimper, Dour Ru, 20231003 / emoc / pierre@lesporteslogiques.net
 Processing 4.0b2 @ kirin / Debian Stretch 9.5


 Interactions clavier
   's' pour enregistrer (en haute définition selon la valeur de SAVE_HIRES ci-dessous)
   'r' pour relancer la génération d'une nouvelle image

 */

boolean SAVE_HIRES = true; // Enregistrer en haute définition, pour impression, ou pas

// fonctions de dates pour l'export d'images de log
import java.util.Date;
import java.text.SimpleDateFormat;
String SKETCH_NAME = getClass().getSimpleName();

// Paramètres principaux
int mode_courbe = 6;             //  /!\ PRESETS : changer cete valeur modifie le graphisme
PImage logo;
ArrayList courbes = new ArrayList();

color[] colors = {
  #5D12D2, #B931FC, #362FD9, #6499E9, #9EDDFF, #A6F6FF, #5B0888, #713ABE, #9D76C1,
  #27005D, #9400FF, #0E21A0, #4D2DB7, #9D44C0, #793FDF, #7091F5, #614BC3, #6F61C0,
  #A084E8, #6528F7, #A076F9, #6527BE, #9681EB  };

color[] colors2 = {   // couleurs secondaires, de contraste
  #FF6AC2, #FFE5E5, #1AACAC, #2E97A7, #BEFFF7, #E5CFF7, #AED2FF, #97FFF4, #33BBC5,
  #85E6C5, #C8FFE0, #8BE8E5, #45CFDD  };

color[] colors3 = {
  #5D12D2, #B931FC, #362FD9, #AED2FF, #97FFF4, #FF6AC2, #FFE5E5, #6499E9, #9EDDFF, #A6F6FF, #5B0888, #713ABE, #9D76C1,
  #27005D, #1AACAC, #2E97A7, #9400FF, #0E21A0, #4D2DB7, #9D44C0, #33BBC5, #85E6C5, #793FDF, #7091F5, #614BC3, #6F61C0,
  #A084E8, #C8FFE0, #6528F7, #A076F9, #BEFFF7, #E5CFF7, #6527BE, #9681EB, #8BE8E5, #45CFDD  };

PFont pixeltiny;
// Paramètres de dimensionnement et de buffer
// 40 x 60 @ 150 dpi : 2362 x 3543, arrondi à 2364 x 3544 (/4 : 591 x 886)
float mul = 4; // rapport entre la fenêtre d'affichage et la taille de buffer
PGraphics pg;

// Debug
//boolean SHOW_CONTROL = true; // passé en variable dans la classe
boolean SHOW_LEGEND = true;



void setup() {
  size(591, 886);
  pg = createGraphics(2364,3544); // buffer 40x60 @ 150dpi
  smooth();
  logo = loadImage("logo_flux.png");
  pixeltiny = loadFont("PixelTiny-48.vlw");
  initCourbes(mode_courbe);
}

void draw () {
  background(255);
  pg.beginDraw();
  pg.background(255);
  pg.endDraw();
  for (int i = 0; i < courbes.size(); i++) {
    Courbe s = (Courbe) courbes.get(i);

    pg.beginDraw();
    println("courbe " + i + " en cours");
    s.draw(pg);
    pg.endDraw();

  }

  // Signature
  pg.beginDraw();
  pg.fill(255);
  pg.rect(pg.width - (150 * mul) - 10, pg.height - (10 * mul) - 16, 216, 21);
  pg.fill(#5B0888);
```

```
  String emoc = signature();
  pg.textFont(pixeltiny, 36);
  pg.text(emoc, pg.width - (150 * mul), pg.height - (10 * mul));
  pg.endDraw();

  /* image(logo, width-125 - 10, 10, 125, 42); // Typo flux */
  image(pg, 0, 0, width, height);
  noLoop();
}

void initCourbes(int mode_courbe) {
  // Chacune de ces fonctions déclenchent des presets différents
  if (mode_courbe == 1) initCourbes1(mul);
  if (mode_courbe == 2) initCourbes2(mul);
  if (mode_courbe == 3) initCourbes3(mul);
  if (mode_courbe == 4) initCourbes4(mul);
  if (mode_courbe == 5) initCourbes5(mul);
  if (mode_courbe == 6) initCourbes6(mul);
  if (mode_courbe == 7) initCourbes7(mul);
  if (mode_courbe == 8) println("pas de presets pour le moment");
  if (mode_courbe == 9) println("pas de presets pour le moment");
}

String signature() {
  Date now = new Date();
  SimpleDateFormat formater = new SimpleDateFormat("yyyyMMdd-HHmmss");
  return "emoc-" + formater.format(now) + "-ic" + mode_courbe;
}

void keyPressed() {
  if (key == 's') {
    Date now = new Date();
    SimpleDateFormat formater = new SimpleDateFormat("yyyyMMdd_HHmmss");
    System.out.println(formater.format(now));
    saveFrame(SKETCH_NAME + "_" + formater.format(now) + "_ic" + mode_courbe + ".png");
    if (SAVE_HIRES) pg.save(SKETCH_NAME + "_" + formater.format(now) + "_HR_ic" + mode_courbe + ".png");
  }
  if (key == 'r') {
    // Relancer une génération en supprimant toutes les courbes
    for (int i = courbes.size() - 1; i >= 0; i--) {
      courbes.remove(i);
    }
    initCourbes(mode_courbe);
    redraw();
  }
}

class Courbe {
  float[][] sg;          // tableau contenant les coordonnées de points pour chaque segment
  float mul;             // multiplicateur d'échelle
  float x;               // coordonnées x du point de départ
  float y;               // coordonnées y du point de départ
  int seg;               // nombre de segments
  float ln;              // longueur de chaque segment (en pixels)
  float wd;              // largeur de chaque segment (en pixels)
  float angstart;        // angle de départ
  float angd1;           // variation de l'angle entre chaque segment, borne basse
  float angd2;           // variation de l'angle entre chaque segment, borne haute
  float anginvert;       // % en dessous duquel l'angle s'inverse à chaque segment (0 : pas d'inversion, 100 : inversion à chaque
segment)
  //float lnc;           // longueur des controles de courbe
  float lnmod;           // modulation de la longueur des vecteurs de controle
  float cbase = 155;     // couleur de base
  color col;             // couleur de la ligne
  float branches;        // nombre branches max
  float branche_dim;     // diminution des dimensions (largeur, longueur) des nouveaux embranchements (entre 0 et 1)
  float branches_gen;    // nombre de génération d'embranchements, diminue à chaque nouvelle génération
  boolean viewcontrol;   // afficher ou pas les vecteurs de contrôle


  /*
    Courbe (float _mul, float _x, float _y, int _seg, float _length, float _width, float _angstart, float _ang1, float _ang2, float
_branches, color _col) {
   _mul : multiplicateur d'échelle
   _x : coordonnée x du point départ
   _y : coordonnée y du point de départ
   _seg : nombre de segments
   _length : longueur d'un segment (en pixels)
   _width : largeur d'un segment (en pixels)
   _angstart : angle de départ (en degrés)
   _ang1 : variation de l'angle entre chaque segment, borne basse (en degrés)
   _ang2 : variation de l'angle entre chaque segment, borne haute (en degrés)
   _anginvert : pourcentage en dessous duquel l'angle s'inverse à chaque calcul (entre 0 : pas d'inversion et 100 : inversion à
chaque segment)
   _branches : nombre d'embranchements
   _branche_dim : diminution des dimensions des nouveaux embranchements (entre 0 et 1)
   _branches_gen : nombre de génération d'embranchements
   _col : couleur de la courbe
   _viewcontrol : affichage ou pas des traites de contrôle
   */


  Courbe (float _mul, float _x, float _y, int _seg, float _ln, float _wd,
          float _angstart, float _ang1, float _ang2, float _anginvert,
          float _br, float _brdim, float _brgen, color _col, boolean _viewcontrol) {
```

```
  mul = _mul;
  x = _x;
  y = _y;
  seg = _seg;
  ln = _ln;
  wd = _wd;
  lnmod = 1;
  angstart = radians(_angstart);
  angd1 = radians(_ang1);
  angd2 = radians(_ang2);
  anginvert = _anginvert;
  sg = new float[seg][15];
  col = _col;
  branches = _br;
  branche_dim = _brdim;
  branches_gen = _brgen;
  viewcontrol = _viewcontrol;

  initSegments();
}

void nouvelleBranche( float sx, float sy, float bang) {

  if (random(100) > 80 && branches > 0 && branches_gen > 0) {

    branches --;
    branches_gen --;

    if (branches < 0) branches = 0;

    courbes.add(  new Courbe(
      mul,
      sx, // x
      sy, // y
      int(seg * branche_dim), // nbre de segments
      ln * branche_dim, // longueur des segments
      wd * branche_dim, // largeur des segments
      degrees(bang) + random(-20, 20), // angle de départ
      degrees(angd1), degrees(angd2), // angle mini, angle maxi
      anginvert,    // inversion d'angle si random(100) < à cette valeur
      branches,   // nombres de branches
      branche_dim,
      branches_gen,
      col,
      viewcontrol)
      );
  }
}

void initSegments() {
  float s_x = 0, s_y = 0, s_a = 0, s_ax = 0, s_ay = 0, s_bx = 0, s_by = 0;
  float s_c1ax, s_c1ay, s_c2ax, s_c2ay, s_c1bx, s_c1by, s_c2bx, s_c2by;
  float s_x0 = 0, s_y0 = 0, s_a0 = 0, s_ax0 = 0, s_ay0 = 0, s_bx0 = 0, s_by0 = 0;

  /* que contient le tableau sg des segments ?

   c2 (c2ax, c2ay)
   /
   /
   *  a (ax,ay) ------------------ * a {n+1} ------- ...
   / |                             |
   /  |                             |
   c1  |                             |
   |                               |
   * point de départ (x, y)        * (point suivant
   |                               |
   |  c2 (c2bx, c2by)              |
   | /                             |
   |/                              |
   *  b (bx, by) ------------------* b {n+1} --------- ...
   /
   /
   c1 (c1bx, c1by)

   sg[n][0] ;  // coordonnée X du point de départ            (s_x)
   sg[n][1] ;  // coordonnée Y du point de départ            (s_y)
   sg[n][2] ;  // angle de direction du segment              (s_a)
   sg[n][3] ;  // coordonnée X du point externe a            (s_ax)
   sg[n][4] ;  // coordonnée Y du point externe a            (s_ay)
   sg[n][5] ;  // coordonnée X du point externe b            (s_bx)
   sg[n][6] ;  // coordonnée Y du point externe b            (s_by)
   sg[n][7] ;  // coordonnée X du point de controle 1 de a   (s_c1ax)
   sg[n][8] ;  // coordonnée Y du point de controle 1 de a   (s_c1ay)
   sg[n][9] ;  // coordonnée X du point de controle 2 de a   (s_c2ax)
   sg[n][10] ; // coordonnée Y du point de controle 2 de a   (s_c2ay)
   sg[n][11] ; // coordonnée X du point de controle 1 de b   (s_c1bx)
   sg[n][12] ; // coordonnée Y du point de controle 1 de b   (s_c1by)
   sg[n][13] ; // coordonnée X du point de controle 2 de b   (s_c2bx)
   sg[n][14] ; // coordonnée Y du point de controle 2 de b   (s_c2by)

   */

  // on commence par créer tous les points et les angles entre les segments...
  for (int i = 0; i < seg; i++) {
    if (i == 0) { // cas particulier du premier segment
```

```
      sg[i][2] = angstart; //random(-PI, PI);
      sg[i][0] = x;
      sg[i][1] = y;
    } else {
      if (random(100) < anginvert) {
        angd1 = -angd1;
        angd2 = -angd2;
      }
      sg[i][2] = sg[i-1][2] + random(angd1, angd2);
      sg[i][0] = sg[i-1][0] + ln * cos(sg[i-1][2]);
      sg[i][1] = sg[i-1][1] + ln * sin(sg[i-1][2]);
    }
  }
  for (int i = 0; i < seg; i++) {

    float lnc;

    if (i == 0) { // cas particulier du premier segment

      s_x = sg[0][0];
      s_y = sg[0][1];
      s_a = sg[0][2];

      s_ax = s_x + wd * cos(s_a - HALF_PI);
      s_ay = s_y + wd * sin(s_a - HALF_PI);
      s_bx = s_x + wd * cos(s_a + HALF_PI);
      s_by = s_y + wd * sin(s_a + HALF_PI);

      lnc = ln / 2 * lnmod;

      s_c1ax = s_ax + lnc * cos(s_a - PI);
      s_c1ay = s_ay + lnc * sin(s_a - PI);
      s_c2ax = s_ax + lnc * cos(s_a);
      s_c2ay = s_ay + lnc * sin(s_a);

      s_c1bx = s_bx + lnc * cos(s_a - PI);
      s_c1by = s_by + lnc * sin(s_a - PI);
      s_c2bx = s_bx + lnc * cos(s_a);
      s_c2by = s_by + lnc * sin(s_a);

      // ajouter les valeurs au tableau
      sg[i][0]  = s_x;
      sg[i][1]  = s_y;
      sg[i][2]  = s_a;
      sg[i][3]  = s_ax;
      sg[i][4]  = s_ay;
      sg[i][5]  = s_bx;
      sg[i][6]  = s_by;
      sg[i][7]  = s_c1ax;
      sg[i][8]  = s_c1ay;
      sg[i][9]  = s_c2ax;
      sg[i][10] = s_c2ay;
      sg[i][11] = s_c1bx;
      sg[i][12] = s_c1by;
      sg[i][13] = s_c2bx;
      sg[i][14] = s_c2by;

    } else {

      s_x = sg[i][0];
      s_y = sg[i][1];
      s_a = sg[i][2];

      float ang1, ang2;

      ang1 = atan2(sg[i][1] - sg[i-1][1], sg[i][0] - sg[i-1][0]);
      if (i == seg - 1) ang2 = ang1;
      else ang2 = atan2(sg[i+1][1] - sg[i][1], sg[i+1][0] - sg[i][0]);

      if (abs(ang2 - ang1) > (TWO_PI - abs(angd2) - abs(angd1))) {
        ang2 = ang2 * -1;
      }
      // probleme ici quand on est proche de -PI et + PI la moyenne est faussée...
      //float angmed = (ang2 + ang1) / 2;
      float angmed = (ang2 - ang1) / 2 + ang1;

      //println(i + " ang1 : " + ang1 + " ang2 : " + ang2 + " angmed : " + angmed );

      s_ax = s_x + wd * cos(angmed - HALF_PI);
      s_ay = s_y + wd * sin(angmed - HALF_PI);
      s_bx = s_x + wd * cos(angmed + HALF_PI);
      s_by = s_y + wd * sin(angmed + HALF_PI);


      // calculer les points de contrôle
      // TODO : définir la longeur des vecteurs de controle
      // en fonction de la distance entre les points de controle...
      lnc = ln / 2 * lnmod;

      s_c1ax = s_ax + lnc * cos(angmed - PI);
      s_c1ay = s_ay + lnc * sin(angmed - PI);
      s_c2ax = s_ax + lnc * cos(angmed);
      s_c2ay = s_ay + lnc * sin(angmed);

      s_c1bx = s_bx + lnc * cos(angmed - PI);
```

```
        s_c1by = s_by + lnc * sin(angmed - PI);
        s_c2bx = s_bx + lnc * cos(angmed);
        s_c2by = s_by + lnc * sin(angmed);

        // ajouter les valeurs au tableau
        sg[i][0]  = s_x;
        sg[i][1]  = s_y;
        sg[i][2]  = s_a;
        sg[i][3]  = s_ax;
        sg[i][4]  = s_ay;
        sg[i][5]  = s_bx;
        sg[i][6]  = s_by;
        sg[i][7]  = s_c1ax;
        sg[i][8]  = s_c1ay;
        sg[i][9]  = s_c2ax;
        sg[i][10] = s_c2ay;
        sg[i][11] = s_c1bx;
        sg[i][12] = s_c1by;
        sg[i][13] = s_c2bx;
        sg[i][14] = s_c2by;

      }
    }
  }

  void updateColBase(float cnew) {
    cbase = cnew;
  }

  void draw(PGraphics pg) {
    pg.stroke(255);
    pg.fill(255);
    pg.strokeWeight(mul);

    for (int i = 0; i < seg-1; i++) {
      pg.stroke(col);
      pg.fill(col);

      if (i == 0) pg.ellipse(sg[i][0], sg[i][1], wd*2, wd*2);
      if (i == seg - 2) pg.ellipse(sg[i+1][0], sg[i+1][1], wd*2, wd*2);

      nouvelleBranche(sg[i][0], sg[i][1], sg[i][2]);

      pg.beginShape();
      pg.vertex(sg[i][3], sg[i][4]);
      pg.bezierVertex(sg[i][9], sg[i][10], sg[i+1][7], sg[i+1][8], sg[i+1][3], sg[i+1][4]);
      pg.vertex(sg[i+1][5], sg[i+1][6]);
      pg.bezierVertex(sg[i+1][11], sg[i+1][12], sg[i][13], sg[i][14], sg[i][5], sg[i][6]);
      pg.endShape(CLOSE);

      if (viewcontrol) {
        /* ************ afficher les points de controle */
        pg.stroke(0);
        pg.strokeWeight(mul * 0.5);
        pg.line(sg[i][3], sg[i][4], sg[i][7], sg[i][8]);
        pg.line(sg[i][3], sg[i][4], sg[i][9], sg[i][10]);
        pg.line(sg[i][5], sg[i][6], sg[i][11], sg[i][12]);
        pg.line(sg[i][5], sg[i][6], sg[i][13], sg[i][14]);
      }
    }
  }
}
void initCourbes1(float mul) {

  int index_col  = int(random(colors.length));
  int index_col2 = int(random(colors2.length));
  //int nb_courbes = 8;
    /*
    Courbe (float _mul, float _x, float _y, int _seg, float _length, float _width, float _angstart, float _ang1, float _ang2, float
_branches, color _col) {
   _mul : multiplicateur d'échelle
   _x : coordonnée x du point départ
   _y : coordonnée y du point de départ
   _seg : nombre de segments
   _length : longueur d'un segment (en pixels)
   _width : largeur d'un segment (en pixels)
   _angstart : angle de départ (en degrés)
   _ang1 : variation de l'angle entre chaque segment, borne basse (en degrés)
   _ang2 : variation de l'angle entre chaque segment, borne haute (en degrés)
   _anginvert : pourcentage en dessous duquel l'angle s'inverse à chaque calcul (entre 0 : pas d'inversion et 100 : inversion à
chaque segment)
   _branches : nombre d'embranchements
   _branche_dim : diminution des dimensions des nouveaux embranchements (entre 0 et 1)
   _branches_gen : nombre de génération d'embranchements
   _col : couleur de la courbe
   _viewcontrol : affichage ou pas des traites de contrôle
   */

  float angle1, angle2, lmin, lmax, dice;
  int compteur = 0;

  // ******************************************************************************

  for (int i=0; i < 5; i++) {
```

```
  dice = random(100);

  // Utiliser compteur pour définir le point de départ et l'angle des courbes
  float xx = 0, yy = 0, aa = 0;
  if (compteur == 0) { // départ du bas
    xx = random(pg.width);
    yy = random(pg.height + 30, pg.height + 100);
    aa = random(210, 330);
  }
  if (compteur == 1) { // départ de la gauche
    xx = random(- 100, -30);
    yy = random(100, pg.height - 100);
    aa = random(280, 440);
  }
  if (compteur == 2) { // départ du haut
    xx = random(100, pg.width - 100);
    yy = random(- 100, -30);
    aa = random(40, 130);
  }
  if (compteur == 3) { // départ de la droite
    xx = random(pg.width + 30, pg.width + 100);
    yy = random(100, pg.height - 100);
    aa = random(100, 260);
  }
  compteur ++;
  if (compteur > 3) compteur = 0; // reset du compteur!


  angle2 = random(10);
  angle1 = -angle2;
  //angle2 = angle1 + random(2, 8);

  courbes.add(  new Courbe(
    mul,
    xx, // x
    yy, // y
    int(random(40, 80)),       // nbre de segments
    random(60, 100) * mul,     // longueur des segments
    random(4, 24) * mul,       // largeur des segments
    aa,                        // angle de départ
    angle1, angle2,            // angle mini, angle maxi
    40,                        // inversion d'angle si random(100) < à cette valeur
    2.0,                       // nombres de branches
    0.9,                       // diminution des dimensions de branche
    3,                         // génération de branches
    colors[index_col],         // couleur
    true));                    // visibilité des traits de contrôle

  index_col ++;
  index_col %= colors.length;
}

// ****************************************************************************

for (int i=0; i < 12; i++) {

  dice = random(100);

  // Utiliser compteur pour définir le point de départ et l'angle des courbes
  float xx = 0, yy = 0, aa = 0;
  if (compteur == 0) { // départ du bas
    xx = random(pg.width);
    yy = random(pg.height + 30, pg.height + 100);
    aa = random(210, 330);
  }
  if (compteur == 1) { // départ de la gauche
    xx = random(- 100, -30);
    yy = random(100, pg.height - 100);
    aa = random(280, 440);
  }
  if (compteur == 2) { // départ du haut
    xx = random(100, pg.width - 100);
    yy = random(- 100, -30);
    aa = random(40, 130);
  }
  if (compteur == 3) { // départ de la droite
    xx = random(pg.width + 30, pg.width + 100);
    yy = random(100, pg.height - 100);
    aa = random(100, 260);
  }
  compteur ++;
  if (compteur > 3) compteur = 0; // reset du compteur!


  angle2 = random(10, 30);
  angle1 = -angle2;
  //angle2 = angle1 + random(2, 8);

  courbes.add(  new Courbe(
    mul,
    xx, // x
    yy, // y
    int(random(8, 16)),          // nbre de segments
    random(12, 40) * mul,        // longueur des segments
    random(1, 3) * mul,          // largeur des segments
```

```
      aa,                          // angle de départ
      angle1, angle2,              // angle mini, angle maxi
      10,                          // inversion d'angle si random(100) < à cette valeur
      4.0,                         // nombres de branches
      0.7,                         // diminution des dimensions de branche
      1,                           // génération de branches
      colors2[index_col2],         // couleur
      true));                      // visibilité des traits de contrôle

    index_col2 ++;
    index_col2 %= colors2.length;
  }

  // ********************************************************************************

  float xx = 0, yy = 0, aa = 0;
  xx = random(pg.width);
  yy = random(pg.height);
  int ic = int(random(colors2.length));

  for (int i=0; i < 12; i++) {

    dice = random(100);
    aa = random(360);
    angle2 = random(5, 10);
    angle1 = -angle2;
    //angle2 = angle1 + random(2, 8);

    courbes.add(  new Courbe(
      mul,
      xx, // x
      yy, // y
      int(random(12, 24)), // nbre de segments
      random(6, 36) * mul, // longueur des segments
      random(1, 3) * mul,  // largeur des segments
      aa,                  // angle de départ
      angle1, angle2,      // angle mini, angle maxi
      0,                   // inversion d'angle si random(100) < à cette valeur
      6.0,                 // nombres de branches
      0.7,                 // diminution des dimensions de branche
      5,                   // génération de branches
      colors2[ic],         // couleur
      true));              // visibilité des traits de contrôle

    index_col2 ++;
    index_col2 %= colors2.length;
  }
}

void initCourbes2(float mul) {

  int index_col  = int(random(colors.length));
  int index_col2 = int(random(colors2.length));
  int index_col3 = int(random(colors3.length));

  float angle1, angle2, lmin, lmax, dice;
  int compteur = 0;
  int ic, ic2, ic3;

  // ********************************************************************************

  for ( int j=0; j < 12; j++) {
    float xx = 0, yy = 0, aa = 0;
    xx = random(pg.width);
    yy = random(pg.height);
    ic = int(random(colors2.length));

    for (int i=0; i < 12; i++) {

      dice = random(100);

      ic = index_col;
      ic2 = index_col2;
      ic3 = int(random(colors3.length));

      aa = random(360);

      angle2 = random(5, 40);  // 5, 10
      angle1 = -angle2;

      courbes.add(  new Courbe(
        mul,
        xx, // x
        yy, // y
        int(random(12, 24)),  // nbre de segments
        random(6, 36) * mul,  // longueur des segments
        random(0.2, 4) * mul, // largeur des segments
        aa,                   // angle de départ
        angle1, angle2,       // angle mini, angle maxi
        0,                    // inversion d'angle si random(100) < à cette valeur
        12.0,                 // nombres de branches // 12
        0.7,                  // diminution des dimensions de branche
        8,                    // génération de branches
        colors3[ic3],         // couleur
        true));               // visibilité des traits de contrôle
```

```
      index_col2 ++;
      index_col2 %= colors2.length;
      index_col ++;
      index_col %= colors.length;
    }
  }
}

void initCourbes3(float mul) {

  int index_col  = int(random(colors.length));
  int index_col2 = int(random(colors2.length));

  float angle1, angle2, lmin, lmax, dice;
  int compteur = 0;

  // ********************************************************************************

  for ( int j=0; j < 16; j++) {
    float xx = 0, yy = 0, aa = 0;
    xx = random(pg.width);
    yy = random(pg.height);
    int ic = int(random(colors2.length));

    for (int i=0; i < 1; i++) {

      dice = random(100);

      aa = random(360);

      angle2 = random(12, 30);
      angle1 = -angle2;

      courbes.add(  new Courbe(
        mul,
        xx, // x
        yy, // y
        int(random(64, 128)),     // nbre de segments
        random(16, 32) * mul,     // longueur des segments
        random(2, 40) * mul,      // largeur des segments
        aa,                        // angle de départ
        angle1, angle2,            // angle mini, angle maxi
        40,                        // inversion d'angle si random(100) < à cette valeur
        3,                         // nombres de branches
        0.7,                       // diminution des dimensions de branche
        2,                         // génération de branches
        colors2[ic],               // couleur
        false));                   // visibilité des traits de contrôle

      index_col2 ++;
      index_col2 %= colors2.length;
      index_col ++;
      index_col %= colors.length;
    }
  }
}

void initCourbes4(float mul) {

  int index_col  = int(random(colors.length));
  int index_col2 = int(random(colors2.length));
  int index_col3 = int(random(colors3.length));

  float angle1, angle2, lmin, lmax, dice;
  int compteur = 0;

  // ********************************************************************************

  for ( int j=0; j < 32; j++) {
    float xx = 0, yy = 0, aa = 0;
    xx = random(pg.width);
    yy = random(pg.height);
    int ic = int(random(colors3.length));

    for (int i=0; i < 1; i++) {

      dice = random(100);

      aa = random(360);

      angle2 = random(40, 90);
      angle1 = -angle2;

      courbes.add(  new Courbe(
        mul,
        xx, // x
        yy, // y
        int(random(16, 96)),      // nbre de segments
        random(18, 128) * mul,    // longueur des segments
        random(0.2, 3) * mul,     // largeur des segments
        aa,                        // angle de départ
        angle1, angle2,            // angle mini, angle maxi
        60,                        // inversion d'angle si random(100) < à cette valeur
        1,                         // nombres de branches
```

```
        0.7,                      // diminution des dimensions de branche
        2,                        // génération de branches
        colors3[ic],              // couleur
        false));                  // visibilité des traits de contrôle

      index_col2 ++;
      index_col2 %= colors2.length;
      index_col ++;
      index_col %= colors.length;
    }
  }
}

void initCourbes5(float mul) {

  int index_col  = int(random(colors.length));
  int index_col2 = int(random(colors2.length));

  float angle1, angle2, lmin, lmax, dice;
  int compteur = 0;

  // ****************************************************************************

  int ic = int(random(colors.length));
  int ic2 = int(random(colors2.length));

  for ( int j=0; j < 16; j++) {
    float xx = 0, yy = 0, aa = 0;
    xx = random(pg.width);
    yy = random(pg.height);


    for (int i=0; i < 1; i++) {

      dice = random(100);

      aa = random(360);

      angle2 = 130;
      //angle1 = -angle2;
      angle1 = 200;

      courbes.add(  new Courbe(
        mul,
        xx, // x
        yy, // y
        int(random(4, 24)),       // nbre de segments
        random(72, 512) * mul,    // longueur des segments
        random(0.2, 4) * mul,     // largeur des segments
        aa,                       // angle de départ
        angle1, angle2,           // angle mini, angle maxi
        10,                       // inversion d'angle si random(100) < à cette valeur
        3,                        // nombres de branches
        0.7,                      // diminution des dimensions de branche
        2,                        // génération de branches
        colors2[ic2],             // couleur
        false));                  // visibilité des traits de contrôle

      index_col2 ++;
      index_col2 %= colors2.length;
      index_col ++;
      index_col %= colors.length;

      ic = index_col;
      ic2 = index_col2;
    }
  }
}

void initCourbes6(float mul) {

  int index_col  = int(random(colors.length));
  int index_col2 = int(random(colors2.length));
  int index_col3 = int(random(colors3.length));

  float angle1, angle2, lmin, lmax, dice;
  int compteur = 0;
  int ic, ic2, ic3;

  // ****************************************************************************

  for ( int j=0; j < 4; j++) {
    float xx = 0, yy = 0, aa = 0;

    for (int i=0; i < 12; i++) {

      dice = random(100);

      ic = index_col;
      ic2 = index_col2;
      ic3 = int(random(colors3.length));

      // départ du bas
      xx = random(pg.width);
      yy = random(pg.height + 30, pg.height + 100);
```

```
      aa = random(230, 310);

      angle2 = random(5, 20);
      angle1 = -angle2;

      courbes.add(  new Courbe(
        mul,
        xx, // x
        yy, // y
        int(random(12, 24)),   // nbre de segments
        random(8, 48) * mul,   // longueur des segments
        random(0.1, 12) * mul, // largeur des segments
        aa,                    // angle de départ
        angle1, angle2,        // angle mini, angle maxi
        0,                     // inversion d'angle si random(100) < à cette valeur
        12.0,                  // nombres de branches
        0.7,                   // diminution des dimensions de branche
        8,                     // génération de branches
        colors3[ic3],          // couleur
        true));                // visibilité des traits de contrôle

      index_col2 ++;
      index_col2 %= colors2.length;
      index_col ++;
      index_col %= colors.length;

    }
  }
}
void initCourbes7(float mul) {

  int index_col  = int(random(colors.length));
  int index_col2 = int(random(colors2.length));
  int index_col3 = int(random(colors3.length));

  float angle1, angle2, lmin, lmax, dice;
  int compteur = 0;
  int ic, ic2, ic3;

  // ****************************************************************************

  for ( int j=0; j < 4; j++) {
    float xx = 0, yy = 0, aa = 0;

    for (int i=0; i < 12; i++) {

      dice = random(100);

      ic = index_col;
      ic2 = index_col2;
      ic3 = int(random(colors3.length));

      // départ du bas
      xx = random(pg.width);
      yy = random(pg.height + 30, pg.height + 100);
      aa = random(230, 310);

      angle2 = random(4, 24);
      angle1 = -angle2;

      courbes.add(  new Courbe(
        mul,
        xx, // x
        yy, // y
        int(random(18, 46)),  // nbre de segments
        random(18, 56) * mul, // longueur des segments
        random(1, 6) * mul,   // largeur des segments
        aa,                   // angle de départ
        angle1, angle2,       // angle mini, angle maxi
        20,                   // inversion d'angle si random(100) < à cette valeur
        12.0,                 // nombres de branches
        0.8,                  // diminution des dimensions de branche
        4,                    // génération de branches
        colors3[ic3],         // couleur
        true));               // visibilité des traits de contrôle

      index_col2 ++;
      index_col2 %= colors2.length;
      index_col ++;
      index_col %= colors.length;
    }
  }
}
```