M1 ENJMIN - MJV102

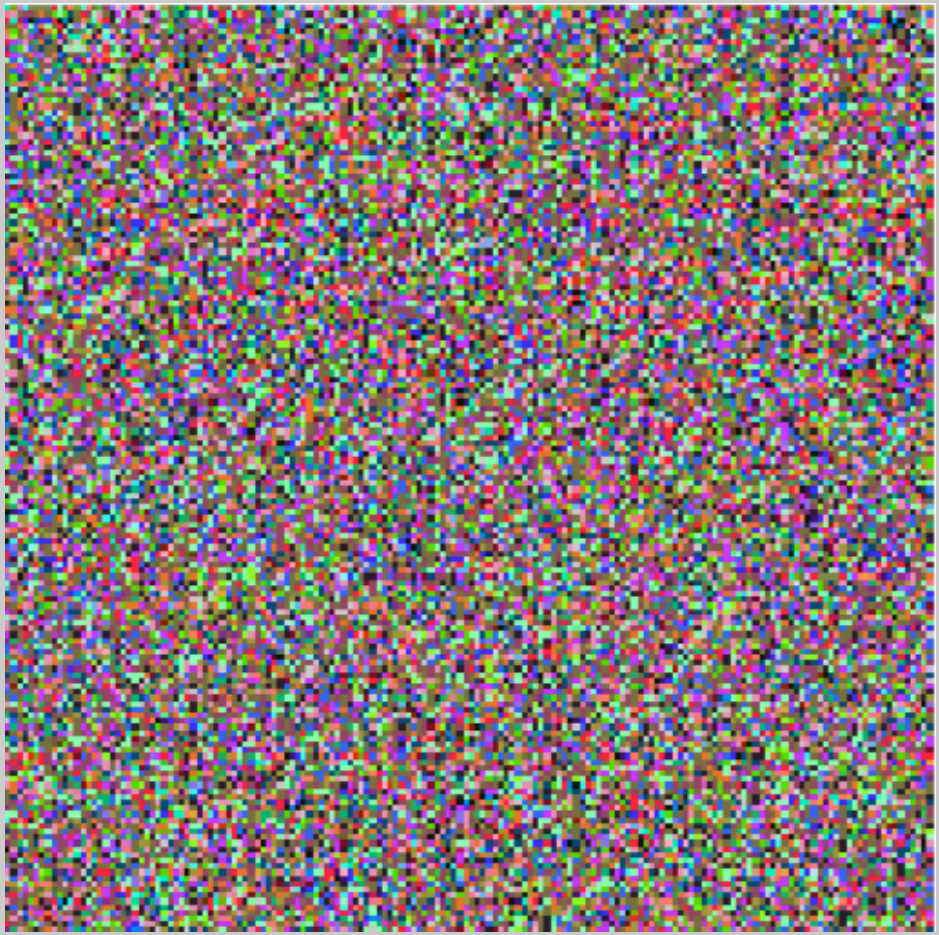
Conception et développement informatique

2 à 6 oct. 2017

Pierre Cubaud cubaud @ cnam.fr
Viviane Gal gal @ cnam.fr

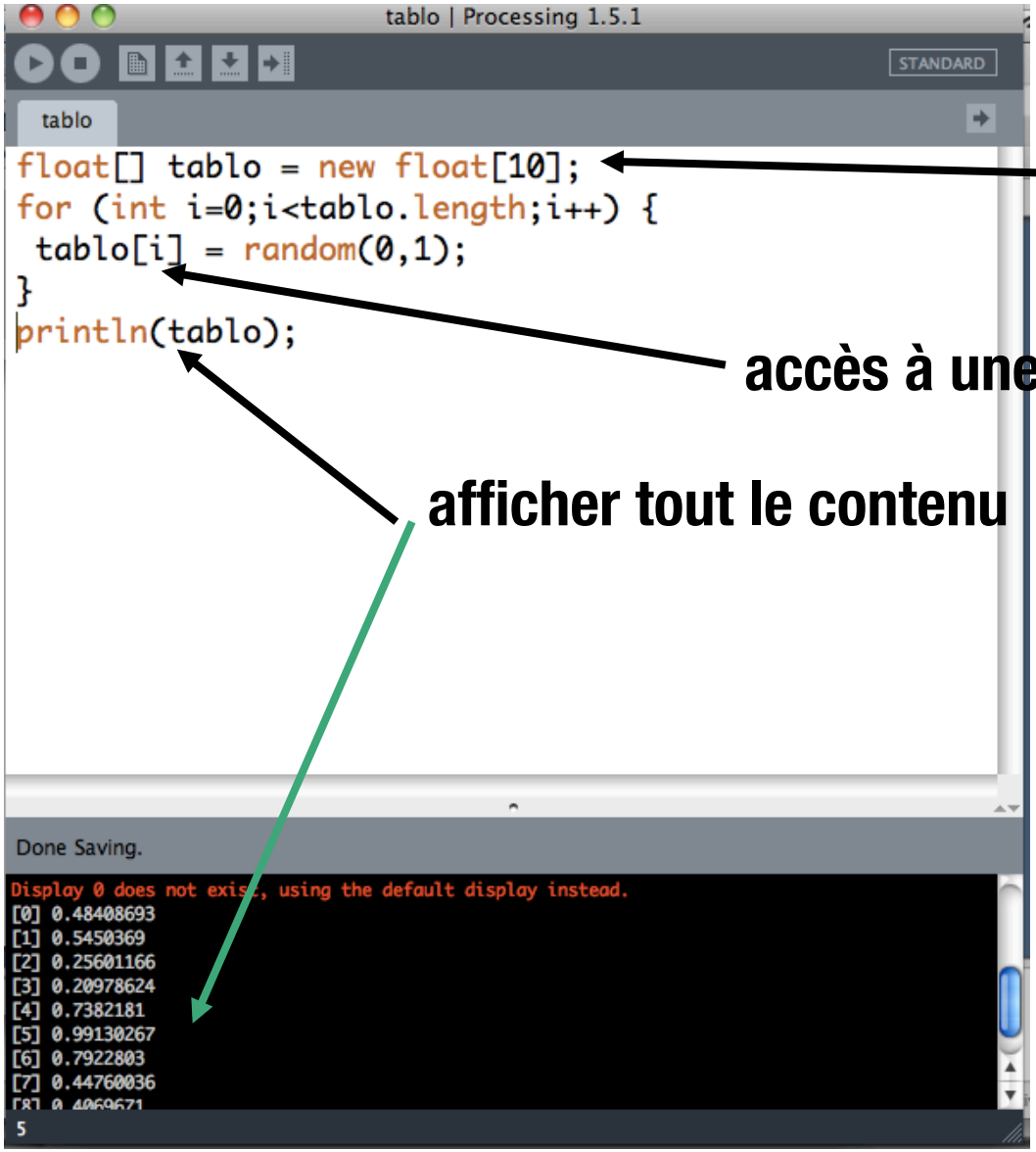
le cnam
École nationale du jeu et des médias interactifs numériques enjmin

JOUR 4. Data/objets



Le tartuffe de Molière encodé en ppm

4.1 Les tableaux



declaration ≠ allocation

```
float[] tablo;
tablo = new float[10];
```

la taille peut dépendre d'une expression entière

```
tablo = new float[x+n%25];
```

on commence à la case 0 jusqu'à length-1

```
x = tablo[0] - tablo[i] + tablo[tablo.length-1];
```

Exemple graphique

tablo

```
float[] tablo = new float[100];
for (int i=0;i<tablo.length;i++) {
 tablo[i] = random(0,1);
}
size(100,6*tablo.length);
fill(0);
int y=0;
for (int i=0;i<tablo.length;i++) {
 rect(0,y,100*tablo[i],3);
 y += 6;
}

save("tablo.png");
```

Manipuler les pixels d'une image : un tableau

image

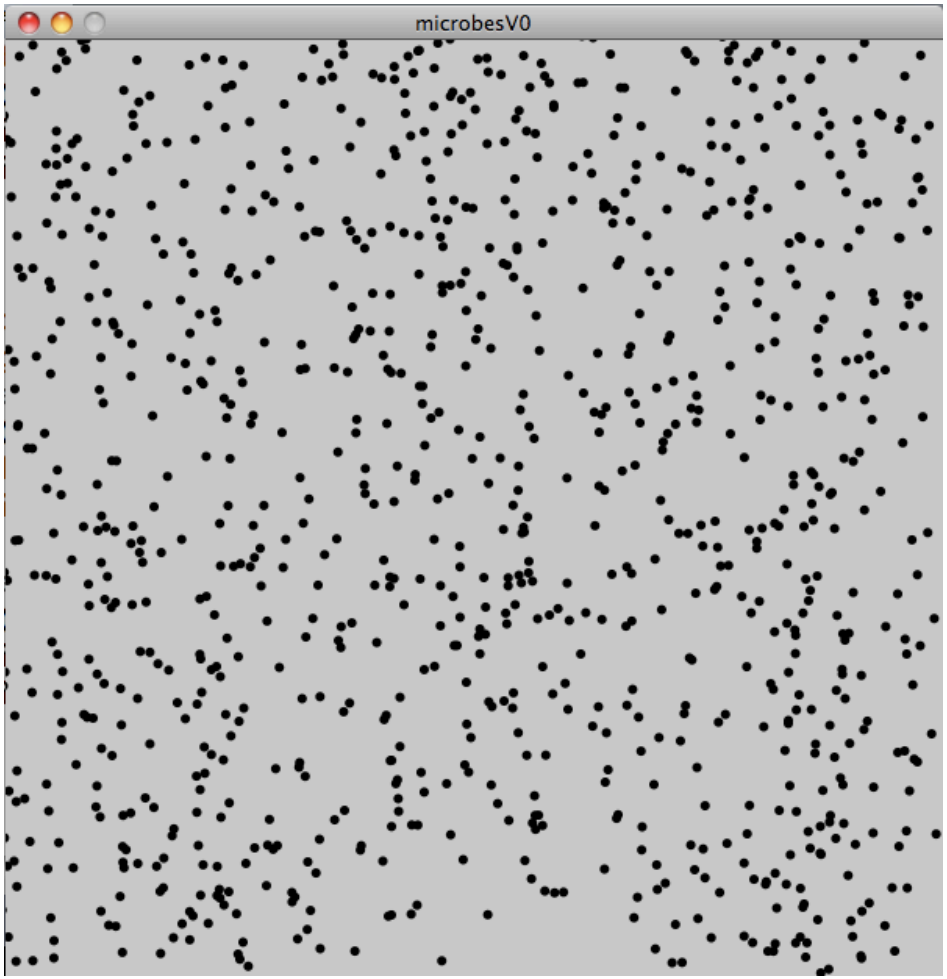
```
// chargement du fichier
PImage ima = loadImage("lenna.gif");
// traitement sur l'image
ima.loadPixels();
for (int i=0;i<ima.width*ima.height;i++){
  ima.pixels[i] = 255 - ima.pixels[i];
}
ima.updatePixels();
// affichage du resultat
size(ima.width,ima.height);
image(ima,0,0);
// sauvegarde
ima.save("resultat.gif");
```

Pointillisme ("Design génératif" P_4_3_1)

```
PImage img;
int Dx, Dy;

void setup() {
  img = loadImage("truc.jpg");
  println(img.width+" x "+img.height);
  size(img.width, img.height);  ////////////// PAS CORRECT
  smooth();
  Dx = int(img.width / 20);
  Dy = int(img.height / 20);
}

void draw() {
  background(255);
  for (int x = 0; x < img.width; x+=Dx) {
   for (int y = 0; y < img.height; y+=Dy) {
    // get current color
    color c = img.pixels[y*img.width+x];
    // greyscale conversion
    int greyscale =round(red(c)*0.222+green(c)*0.707+blue(c)*0.071);
    float w6 = map(greyscale, 0,255, 25,0);
    noStroke();
    fill(c);
    ellipse(x, y, w6, w6);
    }
  }
}
```

(pointillisme.pde)

le mouvement brownien – version tableaux



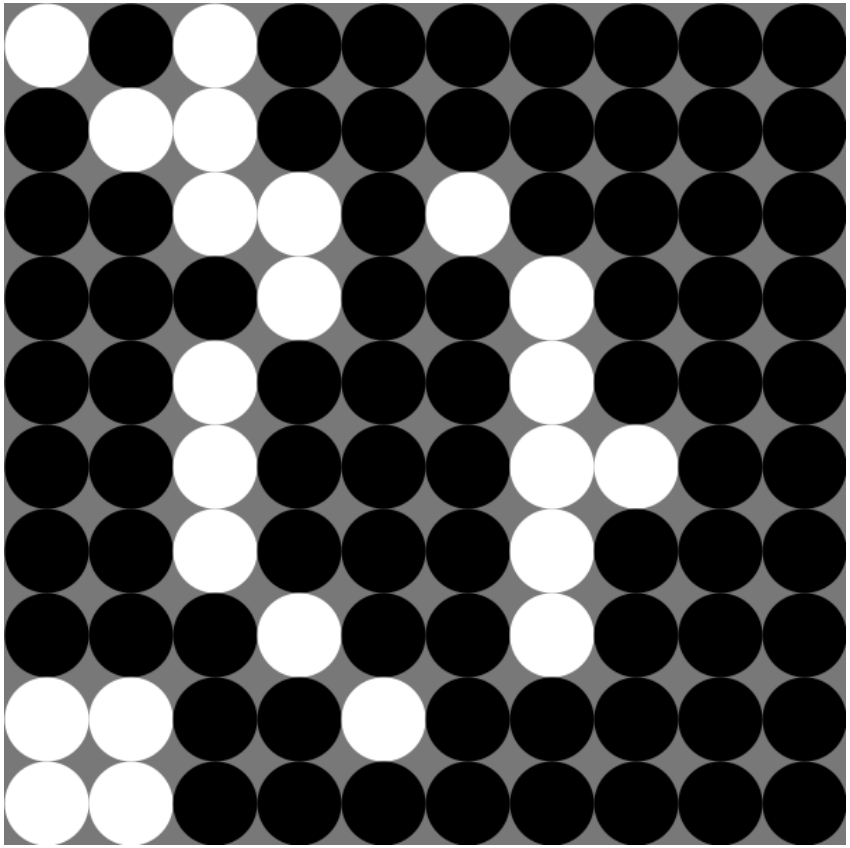
microbesV0

```
final int NBMICROBES = 1000;
float[] mX = new float[NBMICROBES];
float[] mY = new float[NBMICROBES];

void setup() {
 size(600,600);
 smooth();
 for (int i=0;i<NBMICROBES;i++) {
  mX[i] = random(0,width);
  mY[i] = random(0,height);
 }
}
```

on contraint
les objets à rester
dans la zone de dessin

```
void draw() {
 background(200);
 // dessins
 for (int i=0;i<NBMICROBES;i++) {
  fill(0);noStroke();
  ellipse(mX[i],mY[i],6,6);
 }
 // deplacements
 for (int i=0;i<NBMICROBES;i++) {
  mX[i] = (mX[i]+random(-1,+1))%width;
  mY[i] = (mY[i]+random(-1,+1))%height;
 }
}
```

Les matrices : ex. des automates cellulaires



Règle du jeu de la vie : cf wikipedia

```
int DIM = 10;
float R;

int[][]C = new int[DIM][DIM];
int[][]D = new int[DIM][DIM];
int i,j;
float x,y;

void setup(){
  size(600,600);
  smooth();
  noStroke();
  R = width/DIM;
  frameRate(1);
  for (i=0;i<DIM;i++)
   for (j=0;j<DIM;j++)
    {if (random(1)<0.5) C[i][j]=0; else C[i][j]=1;}

}

void draw(){
 background(120);
 // calcul nvelle etape
 for (i=0;i<DIM;i++)
  for (j=0;j<DIM;j++){
   int m=SommeMoore(i,j);
   if (C[i][j]==0)
     if (m==3) D[i][j]=1; else D[i][j]=0;
   else
     if ((m==2)||(m==3)) D[i][j]=1; else D[i][j]=0;
  }

// affichage resultat
 x = R/2; y = R/2;
 for (i=0;i<DIM;i++){
  for (j=0;j<DIM;j++){
    if (D[i][j]==0) fill(0); else fill(255);
    ellipse(x,y,R,R);
    x += R;
   }
   y += R;x = R/2;
 }

 // recopie etat courant
 for (i=0;i<DIM;i++) for (j=0;j<DIM;j++) C[i][j]=D[i][j];

 println(frameCount);
}

void keyPressed(){
  noLoop();
  save("jeu"+frameCount+".png");
}

int SommeMoore(int i, int j) {
 int im1 = (i == 0)?DIM-1:i-1;
 int ip1 = (i == DIM-1)?0:i+1;
 int jm1 = (j == 0)?DIM-1:j-1;
 int jp1 = (j == DIM-1)?0:j+1;
 //int s= C[i][j];
 int s = 0;
 s+= C[im1][jm1];
 s+= C[im1][j];
 s+= C[im1][jp1];
 s+= C[i][jm1];
 s+= C[i][jp1];
 s+= C[ip1][jm1];
 s+= C[ip1][j];
 s+= C[ip1][jp1];
 return s;
}
```

(jeudelavide.pde)

1) Le bouton, avec classe

Comment faire si on veut 2 (ou 100) boutons ???

- 4 paramètres par boutons : x, y, rollover, selected
- lourdeur de la boucle draw()
- test de detection de zone ??

⇒avantage décisif de la programmation "objets"

on va créer une **classe** Bouton qui regroupe les traitements de dessin et la gestion des événements et les paramètres propres à chaque bouton

Squelette du code de la classe :

```
class Button {
  /// ici declaration des attributs x,y, selected, rollover

  Button(float Px, float Py, boolean Pselected) {
    x = Px;
    y = Py;
    selected = Pselected;
  }

  void display() {
   /// ici code d'affichage
  }

  void rollover(int mx, int my) {
   /// ici code detection de rollover
  }

  void clic(int mx, int my) {
   //// ici code gestion du booleen selected
  }
}
```

constructeur pour les objets Button

les autres méthodes

Code complet de la classe

```
class Button {
  boolean selected = false;
  boolean rollover = false;
  float x,y;

  Button(float Px, float Py, boolean Pselected) {
    x = Px;
    y = Py;
    selected = Pselected;
  }

  void display() {
    stroke(0);noFill();
    if (rollover) strokeWeight(4);
    else strokeWeight(1);
    rect(x,y,30,30);
    if (selected) {
      noStroke();fill(0);
      rect(x+10,y+10,10,10);
    }
 }
```

```
  void rollover(int mx, int my) {
    if (mx > x && mx < x + 30 && my > y && my < y + 30)
      rollover = true;
    else
      rollover = false;
  }

  void clic(int mx, int my) {
    if (rollover) selected = ! selected;
  }

}
```

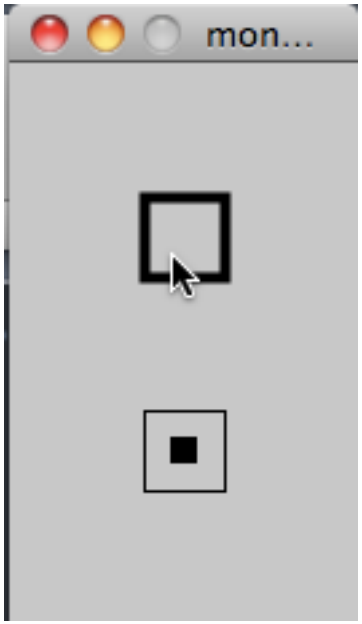
Utilisation dans le programme Processing

monboutonCLASSE | Processing 1.5.1

STANDARD

monboutonCLASSE | Bouton

```
Button b1,b2;

void setup() {
  size(130,210);
  smooth();
  b1 = new Button(50,50,false);
  b2 = new Button(50,130,true);
}

void draw() {
  background(250);
  b1.display();
  b2.display();
}

void mouseMoved() {
  b1.rollover(mouseX,mouseY);
  b2.rollover(mouseX,mouseY);
}

void mousePressed() {
  b1.clic(mouseX,mouseY);
  b2.clic(mouseX,mouseY);
}
```



Rque : mettre le code de la classe dans un fichier séparé est plus commode

Le mouvement brownien, avec classe !

microbesV0

```
final int NBMICROBES = 1000;
float[] mX = new float[NBMICROBES];
float[] mY = new float[NBMICROBES];

void setup() {
 size(600,600);
 smooth();
 for (int i=0;i<NBMICROBES;i++
  mX[i] = random(0,width);
  mY[i] = random(0,height);
 }
}
```

```
void draw() {
 background(200);
 // dessins
 for (int i=0;i<NBMICROBES;i++) {
  fill(0);noStroke();
  ellipse(mX[i],mY[i],6,6);
 }
 // deplacements
 for (int i=0;i<NBMICROBES;i++) {
  mX[i] = (mX[i]+random(-1,+1))%width;
  mY[i] = (mY[i]+random(-1,+1))%height;
 }
}
```

classe ?
attributs ?
méthodes ?

La classe particule

brownienclasse | particule

```
class particule {
  float x,y;

  particule(){
    x=random(0,width);
    y=random(0,height);
  }

  void dessin(){
    noStroke();fill(0);
    ellipse(x,y,6,6);
  }

  void evolution(){
    x = (x+random(-1,+1))%width;
    y = (y+random(-1,+1))%height;
  }
}
```

Le programme principal

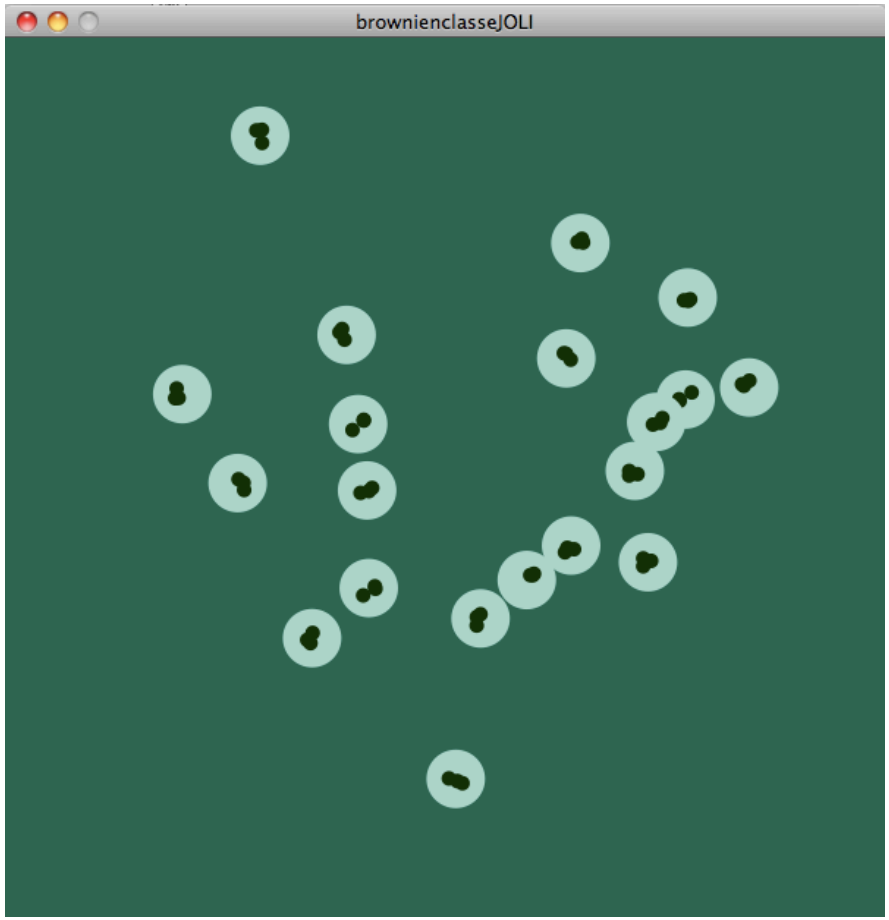
brownienclasse | particule

```
particule[] p = new particule[1000];

void setup() {
 size(600,600);
 smooth();
 for (int i=0;i<p.length;i++) p[i] = new particule();
}

void draw() {
 background(200);
 for (int i=0;i<p.length;i++){
  p[i].dessin();
  p[i].evolution();
 }
}
```

déclaration d'un tableau d'objets

allocation pour chaque objet

Une version plus jolie



Seul le code de la classe Particule est impacté :-)

Nouvelle méthode dessin() :

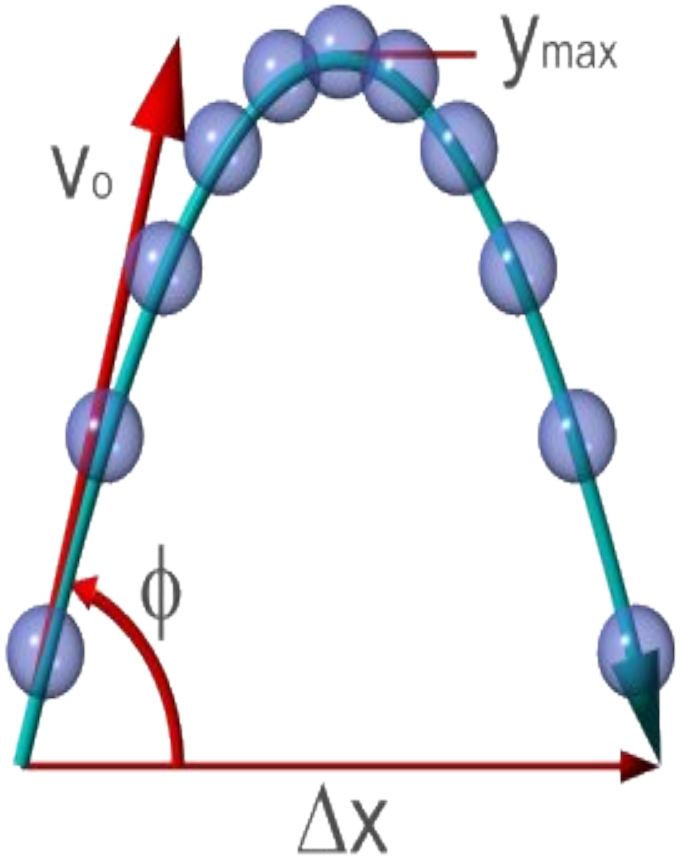
```
void dessin(){
  noStroke();fill(#A2D8CB);
  ellipse(x,y,40,40);
  fill(0,50,0);
  ellipse(random(x-5,x+5),random(y-5,y+5),10,10);
  ellipse(random(x-5,x+5),random(y-5,y+5),10,10);
  ellipse(random(x-5,x+5),random(y-5,y+5),10,10);
}
```

avec quand même un nouveau background(#0A6A52) dans la fonction draw()

Rque : pour choisir des couleurs, utiliser le sélecteur de couleurs dans le menu Tools de Processing

Un peu de physique

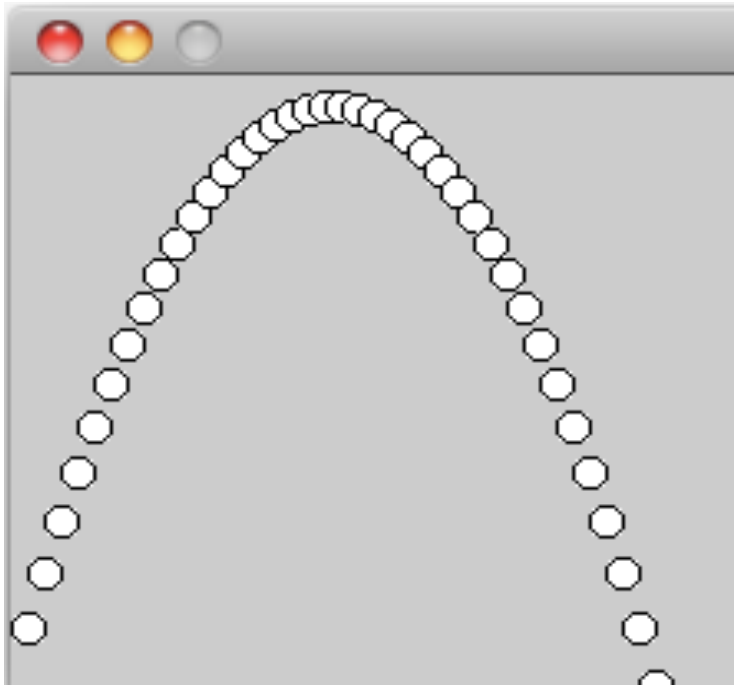
$$\sum_i \vec{F}_i = m\vec{a}$$



$$\sum \vec{F} = \begin{pmatrix} 0 \\ -mg \\ 0 \end{pmatrix} = m\vec{a} = \begin{pmatrix} m\ddot{x} \\ m\ddot{y} \\ m\ddot{z} \end{pmatrix} \Rightarrow \begin{cases} \dot{x} = v_0 \cos\Phi \\ \dot{y} = v_0 \sin\Phi - gt \\ \dot{z} = 0 \end{cases} \Rightarrow \begin{cases} x = v_0 t \cos\Phi \\ y = v_0 t \sin\Phi - \frac{1}{2} g t^2 \\ z = 0 \end{cases}$$

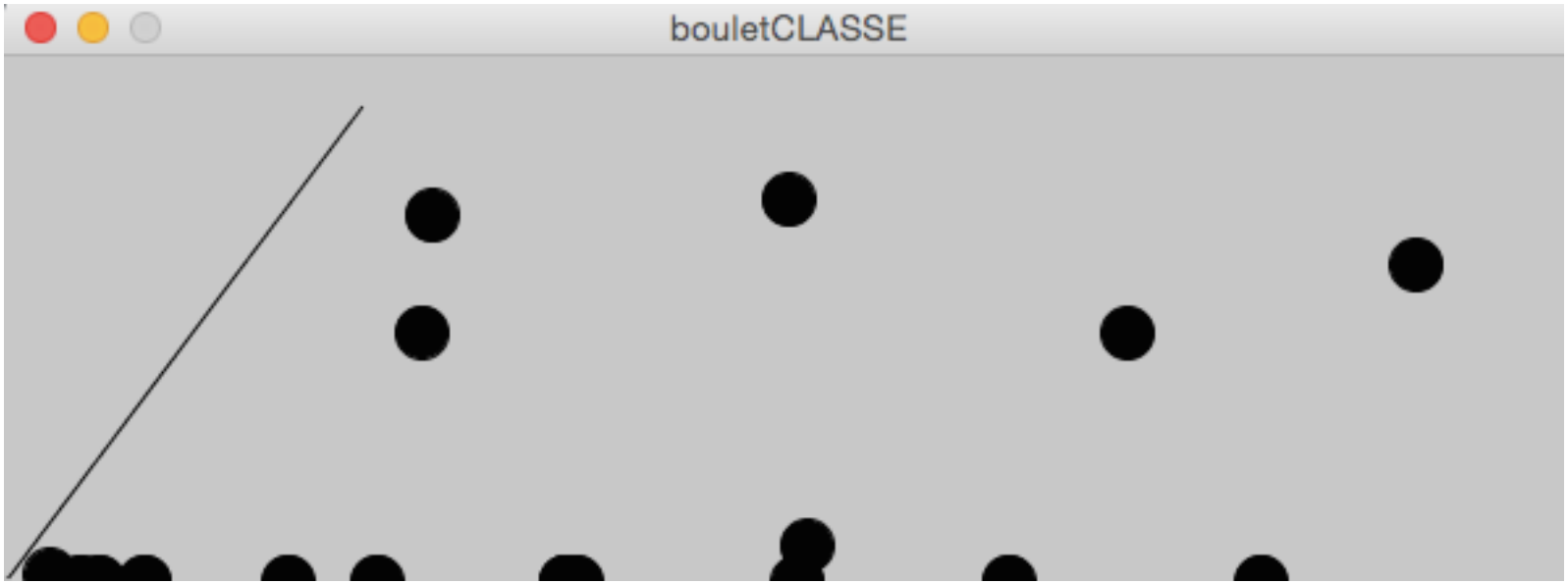
boulet

```
float x,y,vx,vy,ax,ay;

void setup(){
  size(600,200);
  x = 0;
  y = 0;
  vx = 5;
  vy = 20;
}

void draw()  {
  ax = 0;
  ay = -1                    = 1 m/s2
  vx = vx + ax;
  vy = vy + ay;
  x = x + vx;
  y = y + vy;
  ellipse(x,height-y,10,10);
}
```



ajouter les rebonds

Passage à une version en classe



```
class balle {
  float x,y;
  float vx,vy;
  float ax,ay;
  balle(){
    vx=0.1*mouseX;
    vy=0.1*(height-mouseY);
    x=0;
    y=0;
  }
  void evolution(){
   ax = 0;
   ay = -1.0; // ici G = 1 m/s2
   vx = vx + ax;
   vy = vy + ay;
   x = x + vx;
   y = y + vy;
   if (y<0) { vy *= -0.9;y = 0;} // collision sol
   if (y>height){ vy *= -0.9; y = height;} // collision plafond
   if (x<0) { vx *= -0.9; x = 0; } // collision mur gauche
   if (x>width){ vx *= -0.9; x = width;} // collision mur droit
   }

  void dessin(){
   ellipse(x,height-y,20,20);
  }
}
```

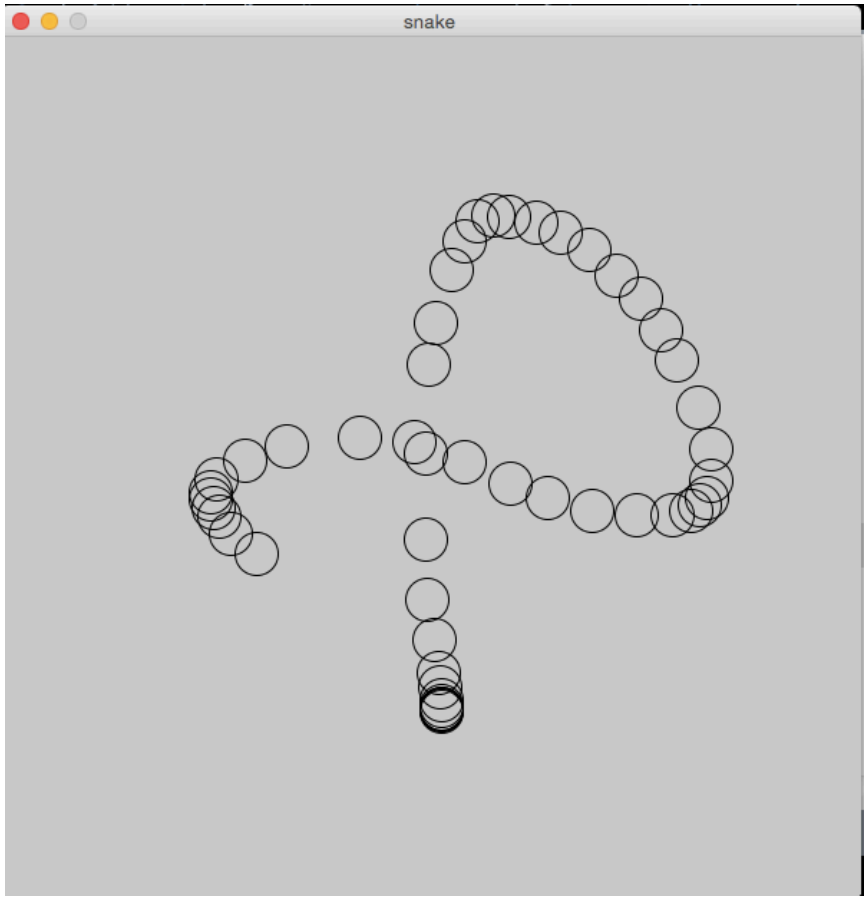
```
balle[] b = new balle[100];
int N=0;

void setup(){
  size(600,200);
  smooth();fill(0);stroke(0);
}

void draw()  {
  background(200);
  line(0,height,mouseX,mouseY);
  for (int i=0;i<N;i++)b[i].dessin();
  for (int i=0;i<N;i++)b[i].evolution();
 }

void keyPressed(){
  println(N);
  b[N] = new balle();
  N++;
  N = constrain(N,1,100);
}
```

(bouletCLASSE.pde)

Des tableaux aux listes : arraylist

```
ArrayList<maclasse> maliste;   //declaration
maliste= new ArrayList<maclasse>();  //creation

int N = maliste.size();

maclasse ob = maliste.get(i); // acces

maclasse ob = new maclasse(); // insertion
maliste.add(ob);

maliste.remove(i);   // destruction
```

snake : historique des positions

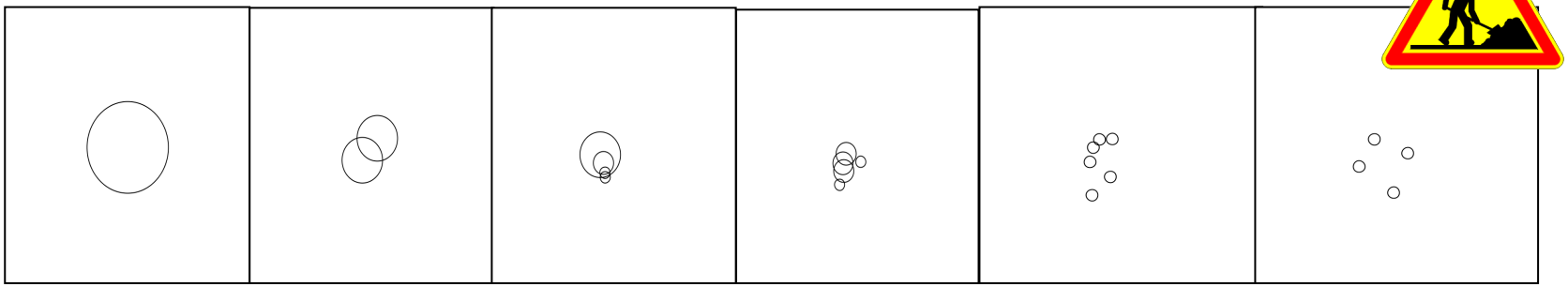


```
ArrayList<PVector> hist = new ArrayList<PVector>();

void setup() {
  size(600,600);
  noFill();
  smooth();
}
```

(snake.pde)

```
void draw() {
  background(200);
  for (int i=0;i<hist.size();i++) {
   ellipse(hist.get(i).x, hist.get(i).y, 30, 30);
  }
}

void mouseMoved() {
    PVector newpos= new PVector(mouseX,mouseY);
    hist.add(newpos);
    if (hist.size()>50) hist.remove(0);
}
```

Un exemple



Au début, il y a sur l'écran une seule particule en mouvement, de grande taille (diamètre 100 pixels).

Si l'utilisateur clique dans l'intérieur de la particule, celle-ci se divise en 2 particules, de diamètre moitié (donc 50 pixels à ce stade).

A chaque fois que l'utilisateur clique dans une particule, le phénomène se reproduit.

Quand le diamètre d'une particule descend au dessous d'un seuil fixé à l'avance (20 pixels, par ex.), la particule est détruite.

```
class Brown {
  float x; float y; float dia;
  Brown(float px, float py, float pdia){
    this.x = px; this.y = py; this.dia = pdia;
  }
  void display(){
    ellipse(x,y,dia,dia);
  }
  void move(){
    x += random(-3,+3);
    y += random(-3,+3);
  }
  boolean finished(){
    return (dia <= 20);
  }
  boolean rollOver(){
    boolean test = (mouseX > x-dia/2)&&(mouseX < x+dia/2 );
    test = test && (mouseY > y-dia/2)&&(mouseY < y+dia/2 );
    return test;
  }
}
```

```
ArrayList<Brown> lesbrowns;

void setup() {
  size(600, 600);
  ellipseMode(CENTER);
  noFill();stroke(0);strokeWeight(2);
  lesbrowns = new ArrayList<Brown>();
  lesbrowns.add(new Brown(width/2, height/2, 200));
}

void draw() {
  background(255);
  for (int i = 0; i<lesbrowns.size(); i++) {
    Brown b = lesbrowns.get(i);
    b.move();
    b.display();
    if (b.finished()) {
      lesbrowns.remove(i);
    }
  }
  if (lesbrowns.size() == 0) {
    fill(0);text("game over",width/2,height/2);}
}
```

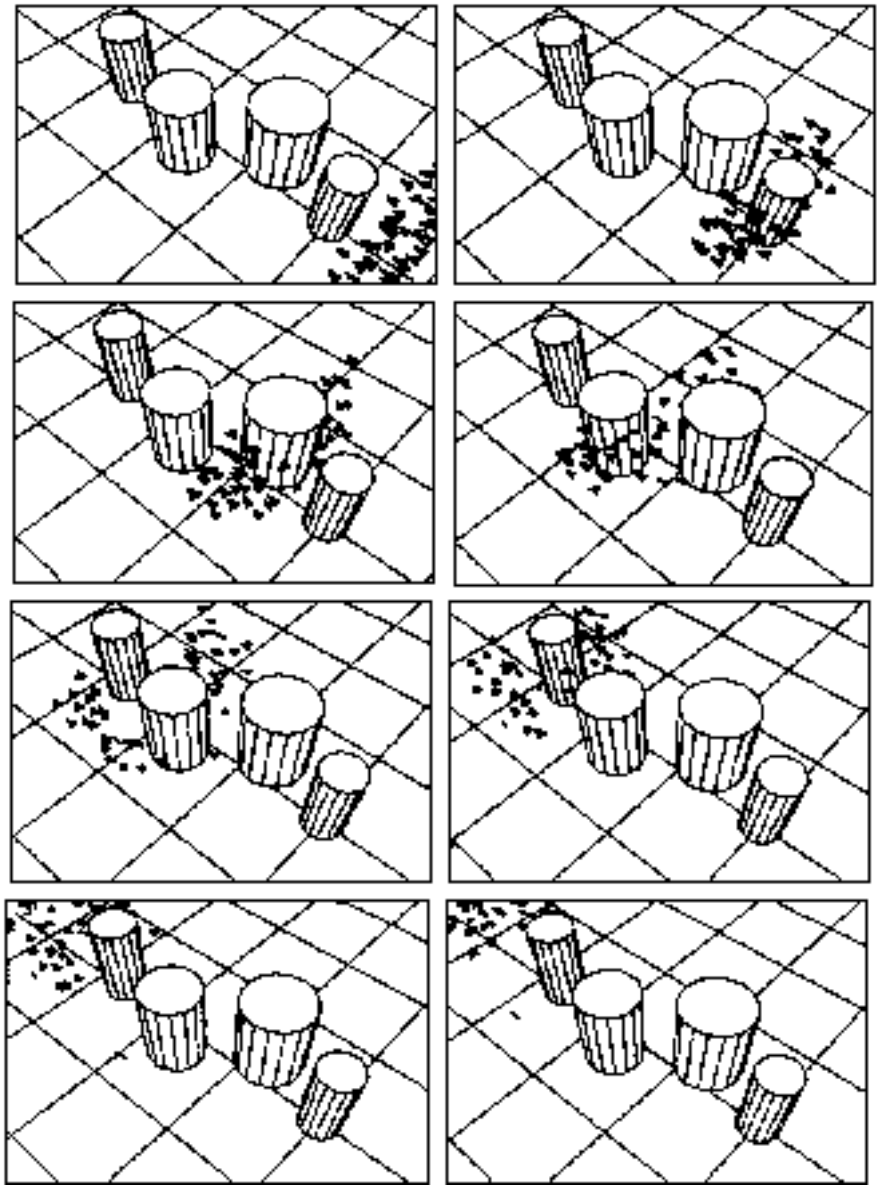
```
void mouseReleased() {
 for (int i = 0; i<lesbrowns.size(); i++) {
    Brown b = lesbrowns.get(i);
    if (b.rollOver()) {
     // on divise la particule en deux plus petites
     b.dia /= 2;
     lesbrowns.add(new Brown(b.x, b.y, b.dia));
     break; // on traite une seule particule dans le for
    }
  }
}

void keyPressed(){
  saveFrame("minijeu#####.png");
}
```

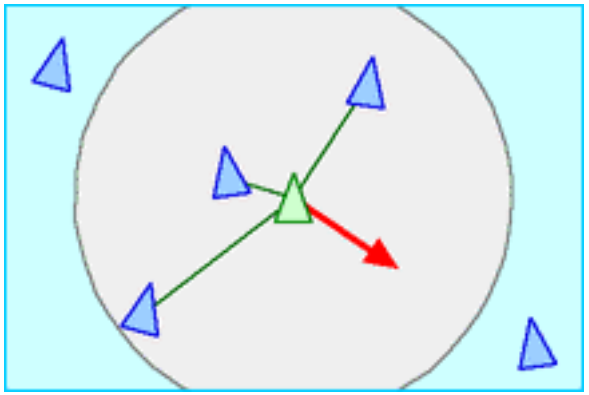
(minijeu.pde)

Boids

C. W. Reynolds,
"Flocks, Herds, and Schools:
A Distributed Behavioral Model",
Computer Graphics,
vol. 21, no. 4, pp 25-34, 1987.

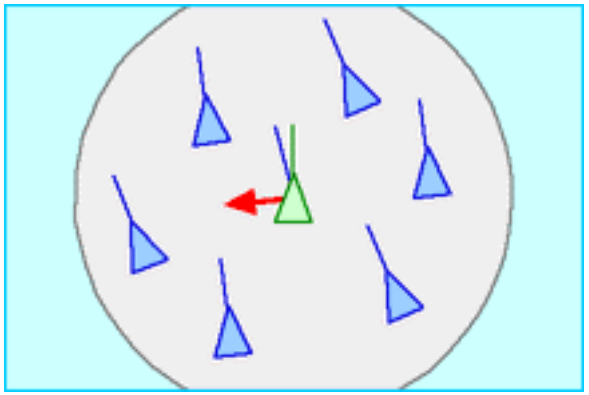
VOIR SON SITE !!
http://www.red3d.com
/cwr/boids



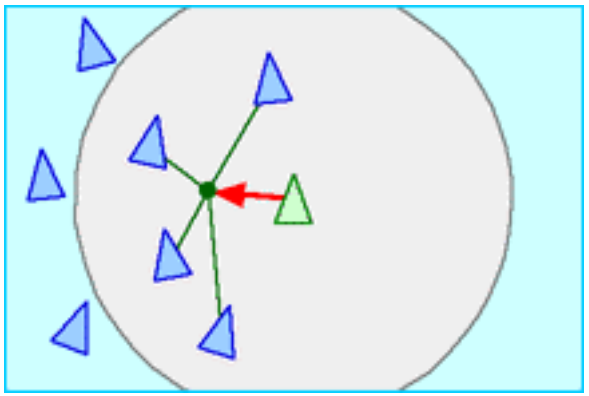


Separation: steer to avoid crowding local flockmates

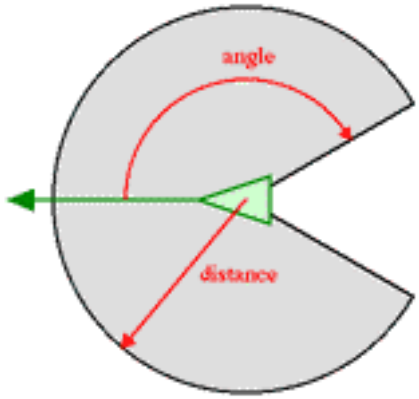
deux forces concurrentes :
-la cohésion au sein du groupe
-l'évitement de collision (prioritaire)



Alignment: steer towards the average heading of local flockmates



Cohesion: steer to move toward the average position of local flockmates



a boid's neighborhood

http://www.red3d.com/cwr/boids

Exemple de programmation avec Processing :