E-ARTSUP.
DÉPARTEMENT DESIGN INTERACTIF & COMMUNICATION VISUELLE.

TEMPS
(FRAMES, HORAIRES ET DÉCOMPTES)

Digital Lab S.07 // Alexandre Rivaux arivaux@gmail.com

Département Design interactif & communication visuelle:

Enseignants:
Nicolas Baumgartner
Félicie d'Estienne D'Orves
Rémi Jamin
Wolf Ka
Jonathan Munn
Gustave Bernier
Alexandre Rivaux

1 - Temps

Le temps est une notion essentielle dans un programme. Cette notion permet de connaitre le temps d'exécution d'un programme, à quelle vitesse ce dernier s'exécute, de découper des animations selon un certains timing mais aussi de prévoir des animations ou actions à des instants précis.

Nous verrons ici trois méthodes permettant d'appréhender le temps dans un programme : les frames, les horloges et le décompte.

2 - Les frames

Les frames (ou images) sont directement inspiré du cinéma et de l'animation, ce sont les images affichées par secondes. En programmation une frame est une exécution complète de la boucle draw(), ainsi le nombre total de frames correspond au nombre total de fois que boucle draw() s'est effectuée. Les frames par seconde (ou FPS) permettent de connaitre la vitesse d'exécution du programme.

Processing permet d'obtenir rapidement le nombre total de frames effectuées ainsi que le nombre de frames par seconde à l'aide des méthodes suivantes :

```
float nbTotalDeFrame = frameCount;
float nbDeFrameParSeconde = frameRate;
```

Sachant que notre programme s'exécute à une vitesse moyenne de 60 frames par seconde nous pouvons donc obtenir le nombre de secondes durant lequel le programme s'est exécuté par un calcul simple :

Temps = frameCount/frameRate

Cependant cette méthode n'est pas conseillée compte tenu du fait qu'elle ne nous renverra jamais un temps réel.
En effet, au cours de son exécution le nombre de frames par seconde d'un programme peut grandement varier en fonction du nombre de calculs à effectuer. Nous pouvons d'ailleurs remarquer que ce dernier oscille entre des valeurs décimales comprises entre 58 et 60. Cette méthode n'est donc pas la plus juste lorsque nous souhaitons calculer un temps précis.

Dans l'exemple ci-dessous nous dessinons une ligne d'un bout à l'autre de la scène en 600 frames. Lorsque le programme atteint la frame 600, nous arrêtons la double draw() à l'aide de la méthode noLoop().

```
int maxFrame = 600; //Limit de frame
float maxWidth; //Taille Max de la ligne
void setup()
{
  size(800, 600, P2D);
}
void draw()
{
  background(0);
  if (frameCount <= maxFrame)
  {
    maxWidth = map(frameCount, 0, maxFrame, 0, width);
  }
  else
  {
    noLoop();
  }
  stroke(255);
  line(0, height/2, maxWidth, height/2);

  println(«nombre total de frame : «+frameCount);
  println(«FrameRate : «+frameRate);
}
```

3 - Horaires

Une seconde méthode permettant de gérer du temps durant un programme consiste à se référer à l'horloge de la machine sur laquelle se dernier s'exécute. Cela permet, quelques soit la vitesse d'exécution du programme, de toujours obtenir une valeur de temps universelle et juste.

Processing nous permet de questionner l'horloge de manière très simple à l'aide des méthodes suivantes :

```
hour() // renvoie des valeurs entre 0 - 23.
minute() // renvoie des valeurs entre 0 - 59.
second() // renvoie des valeurs entre 0 - 59.
millis()
day() // renvoie des valeurs entre 1 - 31.
month() // renvoie des valeurs entre 1 - 12.
year()
```

Nous noterons que les méthodes hour(), minute() et second() renvoi les valeur actuelles de l'horloge alors que la méthode millis() renvoi le temps d'exécution du programme en millisecondes.

Il est donc très facile, à partir de ces méthodes, de réaliser une horloge digitale.

```
int sWidth = 800;
int sHeight = 400;

void setup()
{
  size(sWidth, sHeight, P2D);
}

void draw()
{
  background(0);
  textSize(30);

  textAlign(RIGHT);
  text(hour()+» : «+minute()+» : «+second()+» : «+millis(),
width/2-20, height/2);

  textAlign(LEFT);
  text(day()+» / «+month()+» / «+year(), width/2+20, height/2);
}
```

4 - Décompte

Le décompte ou timer est une des méthodes les plus rependues permettant de gérer du temps lors de l'exécution d'un programme. Cette méthode permet de lancer un décompte en millisecondes. Pour cela nous utiliserons une class Timer. Cette classe permet de calculer une temps passé ainsi qu'un temps restant. Elle se construit de la manière suivante :

```
class Timer {

  int savedTime;
  int totalTime;
  int passedTime;
  int remainingTime;
  boolean timerStarted, timerStopped;

  Timer(int tempTotalTime) {
    totalTime = tempTotalTime;
    timerStarted = false;
    timerStopped = true;
  }

  void start() {
    if (timerStopped) {
      savedTime = millis();
      timerStarted = true;
      timerStopped = false;
    }
  }

  void stop() {
    timerStopped = true;
    timerStarted = false;
  }

  void reset() {
    savedTime = millis();
  }

  boolean isFinished() {
    if (timerStarted) {
      passedTime = millis()- savedTime;
      if (passedTime > totalTime) {
        timerStarted = false;
        timerStopped = true;
        return true;
      }
      else {
        return false;
      }
    }
    else {
      return true;
    }
  }
```

4 - Décompte

```
int getRemainingTime() {
  if (timerStarted) {
    remainingTime = totalTime-passedTime;
    return remainingTime;
    }
    else {
      return -1;
    }
  }
}
```

**Les méthodes start(), stop() et reset() permettent respectivement de lancer arrêter ou réinitialiser le timer.
Les méthodes isFinished() et getRemainingTime() permettent quant à elles de savoir si le timer est fini et quelle est son temps passé en millisecondes.**

Cette classe utilise un algorythme simpme permettant de décompter le temps.

**Lorsque ce dernier est lancé, le timer sauvegarde le temps d'exécution actuel du programme à l'aide de la méthode millis() dans une variable correspondant au point de départ de notre Timer.
Enfin nous effectuons le calcul suivant :**

TempsPassé = TempsactuelleEnMillisecondes – TempsDeDepart

Enfin si cette valeur est supérieur au temps total que nous souhaitons décompter alors notre timer est terminé.

Ainsi, pour créer un timer à l'aide de cet objet il nous suffira alors de créer un objet de type Timer, de le déclarer avec le temps que nous souhaitons décompter puis de le lancer à l'aide de la méthode start(). Nous pourrons ensuite, lors de l'exécution de la boucle draw() interroger l'état de notre timer (en cours ou fini) ainsi que son temps restant.

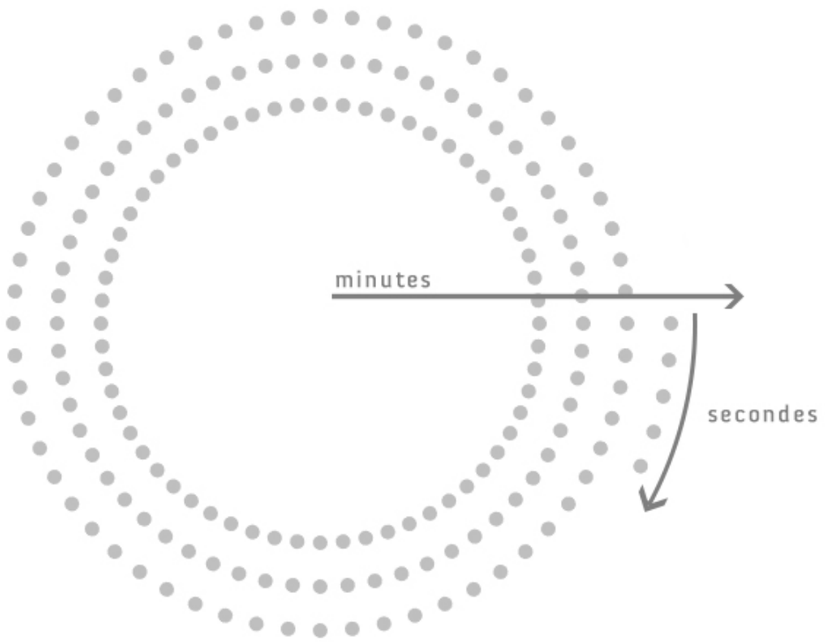
```
int sWidth = 800;
int sHeight = 400;

Timer monTimer;
float taille;

void setup()
{
  size(sWidth, sHeight, P2D);
  monTimer = new Timer(5000);
  monTimer.start();
}

void draw()
{
  background(0);
```

4 - Décompte

```
if (monTimer.isFinished())
{
  monTimer.start();
  }

  taille = map(monTimer.getRemainingTime(), 0, monTimer.totalTime,
100, 300);

  noFill();
  stroke(255);
  ellipse(width/2, height/2, taille, taille);

  fill(255);
  textAlign(CENTER);
  text(monTimer.getRemainingTime(), width/2, height/2);

  println(monTimer.getRemainingTime());
}
```

5 - Exemple

Dans cet exemple nous allons réaliser une horloge décomptant le temps d'exécution du programme en minutes et secondes. Pour cela nous allons utiliser le concept simple du cadran de montre en secondes. À chaque minute passée nous dessinerons un nouveau cadran plus grand que le cadran précédent, chaque cadran se composant de cercles représentant les 60 secondes passées.



Pour cela nous aurons besoins de créer différentes classes : une classe cadran, que nous appellerons secondsDial, et une classe point qui permettra de dessiner un cercle par seconde sur notre cadran.

La classe point sera relativement simple et possédera peu de méthodes. Nous aurons besoins de connaitre une position x et y qui seront définies dans une méthode de la classe secondsDial. Nous définirons un taille ainsi qu'une couleur, enfin nous définirons une méthode display() qui dessinera nos points.

class Point

```
{
  //variables
  float x;
  float y;
  float taille;
  float rgb;

  Point(float x_, float y_)
  {
    x = x_;
    y = y_;
    taille = 10;
    rgb = 0;
  }
```

5 - Exemple

```
void run()
 {
   display();
 }

  void display()
  {
    fill(127);
    noStroke();
    ellipse(x, y, taille, taille);
  }
}
```

Notre classe cadran sera quant à elle un peu plus difficile à mettre en place. Nous aurons besoins des variables suivantes :

- **La position x,y du cadran**
- **Le rayon du cadran**
- **Un ArrayList d'objet Point (nos cercles)**
- **Un compteur de cycle afin de savoir si notre cadran à atteint 60 secondes**
- **Un décompte des secondes**
- **Une limite de secondes**

Afin de réaliser une horloge décomptant le temps d'exécution du programme nous utiliserons la méthode millis(). Celle-ci nous permettra de connaitre en millisecondes le temps d'exécution du programme alors que la méthode seconde() nous renvoi la valeur seconde de notre horloge machine (cf partie 03). Les milliscondes s'incrémentant nous aurons donc besoins de fixer une limite afin de savoir quand nous aurons atteint les 60 secondes pour chaque cadran. C'est la raison pour laquelle nous avons besoins ici d'une variable secondesLimites qui sera équale au temps actuel d'exécution du programme en millisecondes + 60000 millisecondes (soit 1 minute).

NB : Nous pourrions utiliser un timer afin de réaliser ces calculs mais nous verrons ici comment les réaliser "from scratch" afin de se familliariser avec la méthode millis() et le décompte d'un temps

Notre constructeur sera alors :

```
class secondsDial
{
  //variables
  float posX, posY;
  float radius;
  ArrayList<Point> points;
  int countCycle;
  int secondes;
  int secondesLimite;
```

5 - Exemple

```
//constructeur
secondsDial(float radius_, float posX_, float posY_)
{
  posX = posX_;
  posY = posY_;
  radius = radius_;
  countCycle = 0;
  secondes = 0;
  secondesLimite = millis()+60000;

  points = new ArrayList<Point>();

 }
}
```

Afin de décompter les secondes de notre cadran nous réaliserons une méthodes updateSecondes nous permettant de mapper la valeur millis() en secondes pour notre cadran.

```
void updateSeconde()
 {
  if(millis()<=secondesLimite)
  {
    secondes = int(map(millis(), secondesLimite-60000,
secondesLimite, 0, 60));
  }
  else
  {
    secondes = 60;
  }
 }
```

Nous réaliserons ensuite une méthodes booléenne nous permettant de savoir si notre cadran a atteint sa limite de décompte (60 secondes)

```
boolean cycle()
 {
  if (countCycle == 0)
  {
    if (secondes >= 60)
    {
      countCycle = 1;
      return true;
    }
    else
    {
      return false;
    }
  }
  else
  {
    return true;
  }
 }
```

5 - Exemple

Une utiliserons ensuite une méthode update() afin d'ajouter un objet point à notre tableau dynamique à chaque seconde passée. Afin de définir la position de nos point sur notre cadran nous utiliserons les base de trigonométrie nous permettant de définir la position d'un point sur un cercle.

```
void update()
{
  if (!cycle())
  {

    float theta = map(secondes, 0, 60, 0, 360);
    float x = radius*cos(radians(theta))+posX;
    float y = radius*sin(radians(theta))+posY;

    if (points.size()-1 != secondes)
    {
      points.add(new Point(x, y));
    }
  }
}
```

Enfin nous réaliserons une méthode run regroupant nos mises à jour et notre dessin.

```
void run()
 {
   updateSeconde();
   update();
   display();
 }
```

Une fois notre objet cadran réalisé il ne nous restera plus qu'à l'appeler dans notre boucle draw(). Sachant que nous souhaitons ajouter un cadran de plus en plus grand à chaque minutes nous aurons besoins de créer une tableau dynamique d'objet secondsDials. Enfin à l'aide d'une condition nous interrogerons le dernier cadran afin de savoir si son cycle est terminé, et ce à l'aide de sa méthode booléenne cycle(). Si ce dernier est fini nous ajouterons alors un nouveau cadran dont le rayon sera défini par le rayon du cadran inférieur + 10 pixels. Enfin il ne faudra pas oublier d'afficher nos cadrans à l'aide de leur méthodes run()

```
int sWidth = 700;
int sHeight = sWidth/2;

ArrayList<secondsDial> mySD;
int lastElement;
```

5 - Exemple

```
void setup()
{
  size(sWidth, sHeight, P2D);
  smooth(8);
  mySD = new ArrayList<secondsDial>();
  mySD.add(new secondsDial(50, width/2, height/2));
}

void draw()
{
  background(255);


  for (int i =0; i < mySD.size(); i++)
  {
    secondsDial sd = mySD.get(i);
    sd.run();

  }
```

```
  if(mySD.get(lastElement).cycle())
  {
    //println(«newElements»);
    float newRadius = mySD.get(lastElement).radius+10;
    //println(mySD.get(lastElement).points.size());
    mySD.add(new secondsDial(newRadius, width/2, height/2));
    lastElement = mySD.size()-1;
  }
 }
```

6 - Horloge

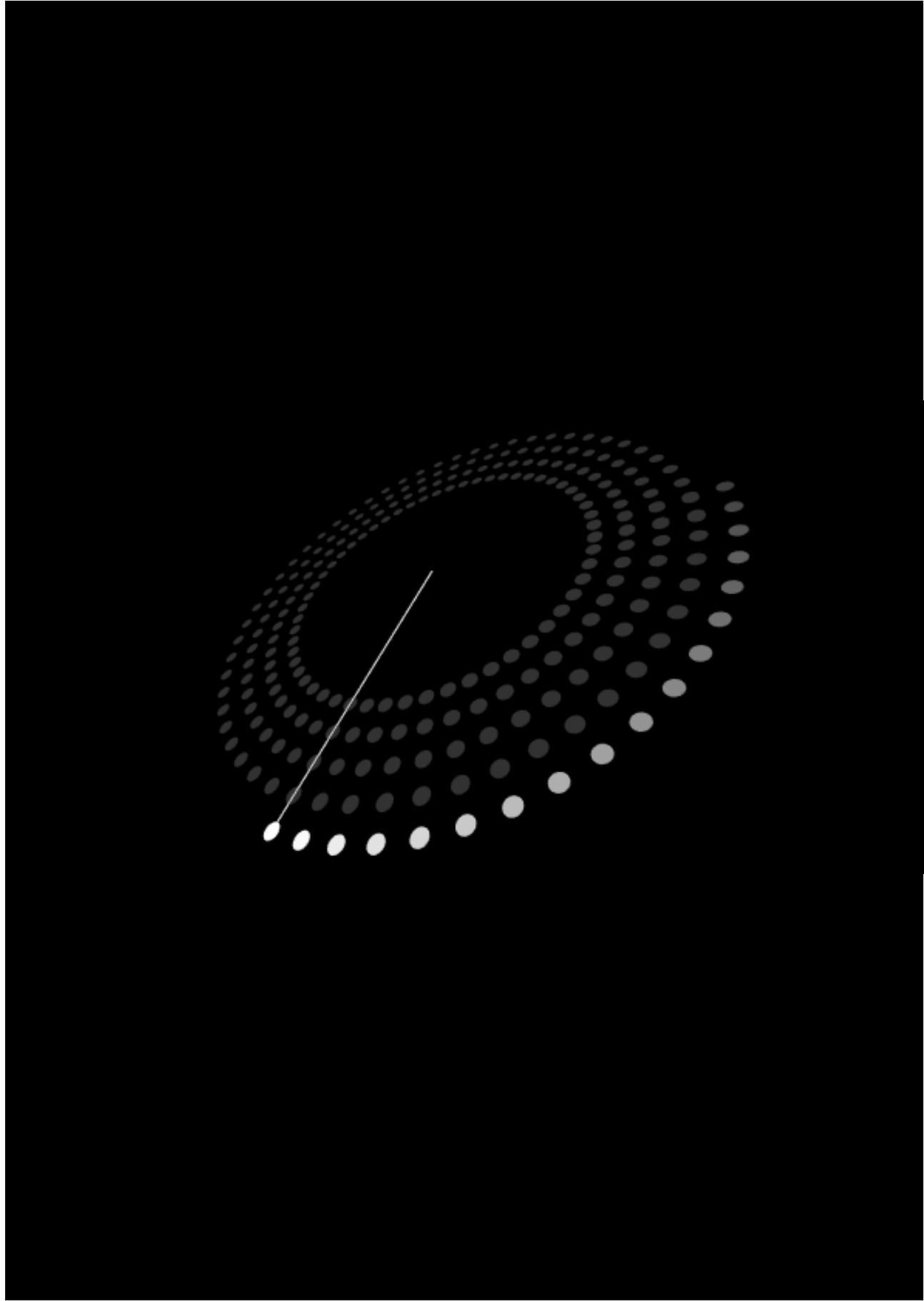
Réalisez 3 programmes basé sur le principe de l'horloge et utilisant les trois méthodes de gestion du temps vu durant ce cours.

<u>Contraintes technique :</u>

- **Format du sketch : 800*400**

<u>Rendu :</u>

- **Images des programmes au format Jpg**
- **Sketchs de chaque programme**



Contact

Alexandre Rivaux
Visual Designer & Partner Bonjour, interactive Lab

www.bonjour-lab.com
arivaux@gmail.com