Meetup Code-Algorithm-Processing-Dream-Paris

https://www.meetup.com/fr-FR/Code-Algorithm-Processing-Dream-Paris/

Atelier « migration »
de Processing vers P5

le 16 janvier 2017

au Friends Vaugirard ☺

Table des matières

1.1 Préambule 3
1.2 Quelques ressources 3
1.3 Et pourquoi pas Processing.js ? 4
2.1 P5.js, première conversion de sketch Processing 7
2.2 P5.js, second exemple de conversion de sketch Processing 8
2.2 P5.js, troisième exemple de conversion de sketch Processing 9
3.1 Ressources, références 14

1.1 Préambule

L'idée de cet atelier est partie d'une constatation personnelle, à savoir que la migration de sketch de Processing vers P5 peut être tantôt assez facile, tantôt très difficile, selon la complexité du sketch Processing à convertir.

Pour rappel, Processing s'appuie sur le langage Java, tandis que P5.js s'appuie sur le langage Javascript. Ces 2 langages présentent quelques similitudes en termes de syntaxe, mais aussi de grosses différences, notamment au niveau du typage des données et du modèle objet. Ces différences peuvent être à l'origine d'incompréhensions, et de découragement.

Au travers de divers scénarios de migration, ce dossier propose une assistance aux développeurs qui en éprouvent le besoins.

1.2 Ressources sur le portage de Processing vers P5.js

Quelques documentations en ligne intéressantes pour faciliter le processus de migration :

http://gerard.paresys.free.fr/Methodes/Methode-Processing-p5.html

https://github.com/processing/p5.js/wiki/Processing-transition

Un convertisseur en ligne :

http://faculty.purchase.edu/joseph.mckay/p5jsconverter.html

Ce convertisseur dégrossit un peu le travail de migration, mais la conversion est loin d'être parfaite et il y a souvent un gros travail de retouche à faire derrière.

1.3 Et pourquoi pas Processing.js ?

Processing.js est un projet parallèle à P5.js, on peut même considérer qu'il est un concurrent direct de P5.js.

Processing.js a été initié par John Resig, un développeur chevronné qui est aussi l'initiateur d'un autre projet bien connu : jQuery. John Resig travaille actuellement pour la Khan Academy, et Processing.js est utilisé par ce site comme support de certains cours d'initiation à la programmation.

Processing.js propose deux modes de fonctionnement :

- Conversion à la volée d'un sketch Processing, réalisée au sein même de la page HTML où le sketch s'exécute (pratique mais moins performant)
- Préconversion d'un sketch Processing en sketch Javascript (compatible uniquement avec Processing.js et pas avec P5.js) : plus performant à l'exécution

L'insertion d'un sketch Processing (non converti en JS) dans une page web se fait de la façon suivante :

```
<script src="
https://cdnjs.cloudflare.com/ajax/libs/processing.js/1.6.3/processing.min.js"></script>
<canvas data-processing-sources="mon_sketch.pde"></canvas>
```

L'outil de préconversion en Javascript est accessible ici :

http://processingjs.org/tools/processing-helper.html

Voici un sketch Processing fourni avec l'IDE de Processing, comme exemple de sketch :

```
// Brightness, by Rusty Robison.
int barWidth = 20;
int lastBar = -1;
void setup() {
  size(640, 360);
  colorMode(HSB, width, 100, width);
  noStroke();
  background(0);
}
void draw() {
  int whichBar = mouseX / barWidth;
  if (whichBar != lastBar) {
    int barX = whichBar * barWidth;
    fill(barX, 100, mouseY);
    rect(barX, 0, barWidth, height);
    lastBar = whichBar;
  }
}
```

Voici le sketch de la page précédente converti en Javascript par Processing.js :

```
// this code was autogenerated from PJS
(function($p) {

    var barWidth = 20;
    var lastBar = -1;

    function setup() {
        $p.size(640, 360);
        $p.colorMode($p.HSB, $p.width, 100, $p.width);
        $p.noStroke();
        $p.background(0);
    }
    $p.setup = setup;
    setup = setup.bind($p);

    function draw() {
        var whichBar = $p.mouseX / barWidth;
        if (whichBar != lastBar) {
            var barX = whichBar * barWidth;
            $p.fill(barX, 100, $p.mouseY);
            $p.rect(barX, 0, barWidth, $p.height);
            lastBar = whichBar;
        }
    }
    $p.draw = draw;
    draw = draw.bind($p);
})
```

On constate que Processing.js effectue un excellent travail de conversion. L'ensemble des fonctions de Processing sont préfixées par le symbole $p (le dollar est peut être un clin d'œil de John Resig par rapport au projet jQuery).

Sur la page suivante, on retrouvera le même sketch, mais adapté cette fois à P5.js.

2.1 P5.js, première conversion de sketch Processing

Voici le même sketch « Brigthness » adapté manuellement en Javascript, pour fonctionner directement avec P5.js :

Version P5	Version Processing (d'origine)

`// Brightness, by Rusty Robison.`
`var barWidth = 20;`
`var lastBar = -1;`
 	
`function setup() {`
`  createCanvas(640, 360);`
`  colorMode(HSB, width, 100, width);`
`  noStroke();`
`  background(0);`
`}`
 	
`function draw() {`
`  var whichBar = mouseX / barWidth;`
`  if (whichBar != lastBar) {`
`  var barX = whichBar * barWidth;`
`  fill(barX, 100, mouseY);`
`  rect(barX, 0, barWidth, height);`
`  lastBar = whichBar;`
`  }`
`}`
	`// Brightness, by Rusty Robison.`
	`int barWidth = 20;`
	`int lastBar = -1;`
	
	`void setup() {`
	`  size(640, 360);`
	`  colorMode(HSB, width, 100, width);`
	`  noStroke();`
	`  background(0);`
	`}`
	
	`void draw() {`
	`  int whichBar = mouseX / barWidth;`
	`  if (whichBar != lastBar) {`
	`  int barX = whichBar * barWidth;`
	`  fill(barX, 100, mouseY);`
	`  rect(barX, 0, barWidth, height);`
	`  lastBar = whichBar;`
	`  }`
	`}`

Dans un cas comme celui-ci, on voit que les adaptations (indiquées en rouge) ont été minimes, pour obtenir un code immédiatement opérationnel avec P5.js.

Les appels de fonction Java (void …) doivent être remplacé par « function » pour Javascript, mais surtout les variables, qui sont fortement typées en Java, doivent être remplacées par des variables (via le mot clé « var ») dont le typage est dynamique en Javascript (on parle de « typage faible », ou encore de langage « faiblement typé », caractéristique que Javascript partage avec PHP notamment).

Nous allons poursuivre avec d'autres exemples de conversion vers P5.js.

2.2 P5.js, second exemple de conversion de sketch Processing

Nous allons effectuer ce second exemple de conversion en utilisant un exemple de code proposé par Paul Orlov, dans son livre « Programming for Artists ».

NB : Ce livre en russe est librement téléchargeable sur le blog de Paul Orlov.

Code P5	Code Processing d'origine

`var a = new Array(500) ;`
 	
`function setup() {`
`  createCanvas(700, 500);`
`  var i, j ;`
`  for (i = 0; i < 500; i++){`
`  a[i] = new Array(2);`
`  for (j = 0; j < 2; j++){`
`  a[i][j] = random(10,490);`
`  }`
`  }`
`}`
 	
`function draw() {`
`  smooth();`
`  noStroke();`
`  background(0);`
`  var i, eDist, eSize, eColor, cx, cy;`
`  for (i = 0; i < a.length; i++){`
`  eDist = dist(mouseX , mouseY , a[i][0],`
`  a[i][1]);`
`  eSize = map(eDist , 0, 200, 5, 100);`
`  eColor = map(eDist , 0, 200, 50, 255);`
`  fill(eColor , 200);`
 	
`  cx = noise(mouseX)*10 + a[i][0];`
`  cy = noise(mouseY)*10 + a[i][1];`
 	
`  ellipse(cx, cy , eSize , eSize);`
`  }`
`}`
	`float [][] a = new float[500][2];`
	
	`void setup() {`
	`  size(500, 500);`
	`  for(int i = 0; i < a.length; i++){`
	`  for(int j = 0; j < a[i].length; j++){`
	`  a[i][j] = random(10,490); 9`
	`  }`
	`  }`
	`}`
	
	`void draw() {`
	`  smooth();`
	`  noStroke();`
	`  background(0);`
	`  for(int i = 0; i < a.length; i++){`
	`  float eDist = dist(mouseX, mouseY,`
	`  a[i][0], a[i][1]);`
	`  float eSize = map(eDist, 0, 200, 5, 100);`
	`  float eColor = map(eDist, 0, 200, 50,`
	`  255);`
	`  fill(eColor, 200);`
	
	`  float cx = noise(mouseX)*10 + a[i][0];`
	`  float cy = noise(mouseY)*10 + a[i][1];`
	
	`  ellipse(cx, cy, eSize, eSize);`
	`  }`
	`}`

Dans cet exemple, la difficulté se situe essentiellement autour du tableau « a » qui est un tableau à 2 dimensions. La syntaxe de création d'un tableau en Java est différente de celle d'un tableau en Javascript. Hormis, cette difficulté, il y a beaucoup de similitudes dans la manipulation des tableaux, dans les 2 langages.

On constate que la déclaration des variables de type « integer » et « float » est effectuée en Javascript par le même mot clé « var ». Une variable pourrait donc changer de type « en cours de route » si l'on n'y prend pas garde.

Si en Java, une variable de type « integer » reçoit le résultat d'un calcul et que ce calcul est de type « nombre à virgule flottante », alors le résultat du calcul sera

automatiquement adapté au type de la variable réceptrice (integer). Javascript n'aura pas ce type de comportement et la variable réceptrice deviendra un « nombre à virgule flottante ». Cela peut avoir des conséquences sur la précision des calculs et donc des résultats ou effets obtenus. On pourra contourner ce problème en Javascript en utilisant la fonction Javascript parseInt() qui aura pour effet de renvoyer un « integer » quoi qu'il arrive.

2.2 P5.js, troisième exemple de conversion de sketch Processing

A partir d'un sketch Processing emprunté à un site japonais, sketch dont le principe consiste à afficher 100 balles rebondissant sur les 4 côtés de l'écran, nous allons aborder le problème de la conversion de classes Java en classes Javascript.

https://github.com/p5aholic/p5codeschool/blob/master/samples/Chapter15_3/sketch07/sketch07.pde

Code source initial en Processing :

```
Ball[] balls = new Ball[100];

void setup() {
  size(750, 350);

  // Générer 100 objets de classe Ball
  for (int i = 0; i < balls.length; i++) {
    balls[i] = new Ball();
  }
}

void draw() {
  background(255);

  // Exécuter les méthodes de mise à jour et d'affichage pour tous les objets Ball
  for (int i = 0; i < balls.length; i++) {
    balls[i].update();
    balls[i].display();
  }
}
```

```
class Ball {
  float x, y;
  float vx, vy;
  int radius;
  color c;
  // constructeur
  Ball() {
    this.radius = (int)random(10, 20);
    this.x = random(this.radius, width-this.radius);
    this.y = random(this.radius, height-this.radius);
    this.vx = random(-5, 5);
    this.vy = random(-5, 5);
    this.c = color(random(255), random(255), random(255), random(255));
  }
  // méthode de mise à jour
  void update() {
    this.x += this.vx;
    this.y += this.vy;
    if (this.x-this.radius <= 0 || this.x+this.radius >= width) {
      this.vx *= -1;
    }
    if (this.y-radius <= 0 || this.y+this.radius >= height) {
      this.vy *= -1;
    }
  }
  // méthode d'affichage
  void display() {
    noStroke();
    fill(c);
    ellipse(x, y, 2*radius, 2*radius);
  }
}
```

La conversion des fonctions setup() et draw() ne présente pas de difficulté, mais la conversion de la classe Ball de Java vers Javascript peut se faire de différentes manières :

- Soit par une réécriture en Javascript selon la norme ES5
- Soit par une réécriture en Javascript selon la norme ES6 (syntaxe Javascript apparue en 2015, plus proche de la syntaxe Java)

Nous allons étudier dans un premier temps la seconde option, car elle est plus facile à aborder pour un développeur ne maîtrisant pas toutes les subtilités des objets Javascript.

Pour ce faire, il nous faut retravailler un peu le code de la classe Ball, pour qu'elle soit conforme aux principes de la norme ES6 :

```
class Ball {
  constructor() {
    this.radius = random(10, 20);
    this.x = random(this.radius, width-this.radius);
    this.y = random(this.radius, height-this.radius);
    this.vx = random(-5, 5);
    this.vy = random(-5, 5);
    this.c = color(random(255), random(255), random(255), random(255));
  }
  update() {
    this.x += this.vx;
    this.y += this.vy;
    if (this.x-this.radius <= 0 || this.x+this.radius >= width) {
      this.vx *= -1;
    }
    if (this.y-radius <= 0 || this.y+this.radius >= height) {
      this.vy *= -1;
    }
  }
  display() {
    noStroke();
    fill(c);
    ellipse(x, y, 2*radius, 2*radius);
  }
}
```

On peut intégrer le code de la page précédente à notre sketch, cela fonctionnera avec les navigateurs les plus récents, sous réserve qu'ils soient à jour (c'est le cas de Chrome, Firefox, Safari, etc..). Mais les navigateurs plus anciens ne supporte pas ES6, aussi si l'on souhaite bénéficier d'une portabilité maximale, on aura intérêt à utiliser un outil comme BabelJS pour convertir notre code ES6 en ES5 :

https://babeljs.io/repl/

Lors que vous effectuerez la conversion de votre code avec BabelJS, vous constaterez que BabelJS ajoute quelques fonctions spécifiques, qui lui permettent d'effectuer plus simplement le portage du code vers ES5. Ces fonctions doivent être insérées avec la classe dans votre sketch pour que ce dernier fonctionne. Voici en particulier le code qui sera ajouté pour notre sketch en cours d'étude (le code de la classe Ball se trouve page suivante) :

```
"use strict";

var _createClass = function () { function defineProperties(target, props) { for (var
i = 0; i < props.length; i++) { var descriptor = props[i]; descriptor.enumerable =
descriptor.enumerable || false; descriptor.configurable = true; if ("value" in
descriptor) descriptor.writable = true; Object.defineProperty(target, descriptor.key,
descriptor); } } return function (Constructor, protoProps, staticProps) { if
(protoProps) defineProperties(Constructor.prototype, protoProps); if (staticProps)
defineProperties(Constructor, staticProps); return Constructor; }; }();

function _classCallCheck(instance, Constructor) { if (!(instance instanceof
Constructor)) { throw new TypeError("Cannot call a class as a function"); } }
```

Voici le code de la classe Ball converti en ES5 par BabelJS :

```
var Ball = function () {
  // constructeur
  function Ball() {
    _classCallCheck(this, Ball);
    this.radius = random(10, 20);
    this.x = random(this.radius, width - this.radius);
    this.y = random(this.radius, height - this.radius);
    this.vx = random(-5, 5);
    this.vy = random(-5, 5);
    this.c = color(random(255), random(255), random(255), random(255));
  }
  _createClass(Ball, [{
    key: "update",
    value: function update() {
      this.x += this.vx;
      this.y += this.vy;
      if (this.x - this.radius <= 0 || this.x + this.radius >= width) {
        this.vx *= -1;
      }
      if (this.y - radius <= 0 || this.y + this.radius >= height) {
        this.vy *= -1;
      }
    }
  }, {
    key: "display",
    value: function display() {
      noStroke();
      fill(c);
      ellipse(x, y, 2 * radius, 2 * radius);
    }
  }]);
  return Ball;
}();
```

3.1 Références

BabelJS, convertisseur JS de ES5 vers ES6 :

https://babeljs.io/repl/

Tutoriel pour la création de classes ES6

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Classes

Site officiel P5.js :

http://p5js.org/

Exemples de sketchs Processing intéressants à convertir (pour se faire la main) :

https://www.openprocessing.org/sketch/169537

https://github.com/PaulOrlov/processing-1

http://ptahi.ru/2016/03/25/processing-sketch-based-on-histograms-of-iterated-chaotic-functions/