Les joies confuses

Guillaume Pelletier-Auger

20 août 2016

Résumé

Un film d'animation programmé en JavaScript avec l'aide de la bibliothèque p5.js.

1 Introduction

Ce film d'animation combine divers sketchs ensemble.

2 L'objet *Scene*

La pierre d'assise de ce projet est l'objet prototype *Scene*, qui ressemble en certains aspects à l'objet prototype *Oscillator*. L'objet *Scene* est différent d'*Oscillator* parce qu'il contrôle une bien plus grande quantité de variables. La meilleure façon de renvoyer ces variables à leurs variables globales homologues reste à déterminer.

Je vois deux options. La première option : L'objet *Scene* pourrait contenir une copie des valeurs requises sous formes de propriétés (variables locales accessibles publiquement). La x-sheet se chargerait donc de faire rouler chaque instance de *Scene* au bon moment, et ensuite appliquerait les propriétés de cette *Scene*.

Deuxième option : Les méthodes *update* et *mix* de l'objet *Scene* se chargeraient elles-mêmes d'appliquer les propriétés aux variables globales. Je pense qu'il s'agit là de la meilleure solution.

```
Scene = function(func) {
    this.func = func;
};

Scene.prototype.update = function(t){
    var that = this;
    this.func(t, that);
    this.makeGlobal();
};

Scene.prototype.mix = function(otherScene, l) {
    this.makeGlobal();
```

```
};

Scene.prototype.makeGlobal = function() {
    globalValues = this.localValues;
};

Scene.prototype.runSpiral = function(t) {

};

Scene.prototype.runPhysicsEngine = function(t) {

};

var explosion312 = new Scene(function(t, that) {
    that.zoom = 2;
    that.rotationSpeed = 12;
    that.runSpiral(t);
});

var explosion320 = new Scene(function(t, that) {
    that.localValues.sketch = {
        zoom : 3,
        rotationSpeed : 2
    };
    that.localValues.superEllipse = {
        n2 : 3,
        sc : 20,
        scPow : 7,
        m : 8
    };
    that.runPhysicsEngine(t);
});
```

2.1 Ce qu'une instance de *Scene* contrôle

Chaque *Scene* doit pouvoir contrôler une énorme quantité de variables dans le sketch.

1. Les éléments généraux du sketch : le niveau de zoom, l'angle de rotation, la taille des cercles.
2. Les valeurs du dégradé : les 3 valeurs R, G, et B, la valeur *step*.
3. Les paramètres de l'engin de physique :
4. Les paramètres de la spirale :

Mon idée pour l'instant c'est que chaque scène retournerait un objet littéral qui contiendrait chacune de ces valeurs, un peu comme ça :

```
return {
    zoom : 3,
```

```
    rotationAngle : 240,
    gradient : {
            r : 255,
            g : 120,
            b : 0
    },
    spiral : {
        speed  : 120,
        angle  : 12
    },
    physicsEngine : {
        sc : 20,
        scPow : 10
    }
}
```

3 L'objet *globalValues*

Une idée : Je pourrais avoir un objet littéral appelé *globalValues* qui contiendrait toutes les données requises pour animer le film. De cette façon, tout serait bien classé et structuré au lieu d'être une énorme pile de variables distinctes. Je me servirais ensuite de cet objet pour accéder à toutes les valeurs dont j'ai besoin.

```
//Déclaration de l'objet globalValues, structuré en arbre.
var globalValues = {
    graph : [],
    gradient : {
        color1 : {
            offset : 0,
            r : 255,
            g : 120,
            b : 0
        },
        color2 : {
            offset : 0.2,
            r : 255,
            g : 120,
            b : 0
        },
        color3 : {
            offset  : 0.8,
            r : 255,
            g : 120,
            b : 0
        }
    },
    spiral : {
        speed  : 120,
        angle  : 12
    },
    physicsEngine : {
        sc : 20,
```

```
        scPow : 10
    }
}
```

L'objet *globalValues* n'a pas besoin de contenir la valeur *gradientSpeed* puisque cette valeur sert uniquement à l'objet *Scene*. Même chose pour spiralSpeed, spiralAngle.

L'objet *globalValues* n'a besoin que des valeurs qui sont nécessaires pour *imprimer* le sketch. Par exemple, je pourrais avoir une fonction *printBackgroundGradient* :

```
function printBackgroundGradient() {
    var gradient = ctx.createRadialGradient(0, 0, 0, 0, 0, width);
    var col1 = globalValues.gradient.color1;
    var col2 = globalValues.gradient.color2;
    var col3 = globalValues.gradient.color3;
    gradient.addColorStop(col1.offset, "rgba(" + col1.r + ", " + col1.g + ", " + col1.b + ",1)");
    gradient.addColorStop(col2.offset, "rgba(" + col2.r + ", " + col2.g + ", " + col2.b + ",1)");
    gradient.addColorStop(col3.offset, "rgba(" + col3.r + ", " + col3.g + ", " + col3.b + ",1)");

    ctx.fillStyle = gradient;
    rect(-width * 0.5, -height * 0.5, width, height);
}
```

4 La fonction *printDots*

Tout pourrait être imprimé avec seulement deux fonctions, donc. Une fonction pour imprimer le background et une fonction pour imprimer les points. Non, il me faut aussi contrôler le zoom et la rotation du sketch en général. Est-ce tout ?

```
function printDots() {
    for (var i = 0; i < globalValues.graph.lenght; i++) {
        var dot = globalValues.graph[i];
        fill(dot.col.r, dot.col.g, dot.col.b);
        ellipse(dot.pos.x, dot.pos.y, dot.size, dot.size);
    }
}
```

5 Pensées et questions ouvertes

Je crois que la variable publique qui contrôle l'angle de rotation du canvas devrait être *rotationSpeed*, et non pas l'angle de rotation lui-même.

Les deux fonctions principales qui génèrent mes images, physicsEngine et runSpiral, devraient-elles être activées par les instances de *Scene*, ou devraient-elle rouler continuellement, indépendamment, et lire elles-mêmes l'objet globalValues ?

Si je veux pouvoir faire une interpolation linéaire entre 2 spirales différentes (2 spirales rendues à des stades différents de leur course), et c'est une chose qu'il me faut absolument pouvoir faire, ça veut dire que la spirale doit être générée par l'instance de *Scene* elle-même.

La fonction *physicsEngine* est bien différente parce qu'elle crée elle-même des transitions intéressantes lorsque ses valeurs sont modifiées. Cependant, puisqu'il faut que mes instances de *Scene* génèrent elles-mêmes tous les points du graphe (pour que les interpolations linéaires en les scènes soient possibles), peut-être serait-ce mieux que les instances de *Scene* roulent également leur propre *physicsEngine*.

La valeur localValues ne pourrait-elle pas contenir uniquement les données de positions, de couleurs et de taille de chaque point, les données nécessaires à la génération du fond d'écran dégradé, et les données *rotationSpeed* et *zoom* ?

Ainsi, toutes les autres données, tel que les paramètres multiples de mes superformules et de mes courbes de Lamé, seraient préservées à l'intérieur de chaque instance de *Scene*, inutile et donc cachée au monde extérieur ? Ça me semble bien être la meilleure façon de faire. Dans cette optique, je pourrais avoir un autre paramètre au prototype *Scene* : *privateValues*.

privateValues est un objet qui contient toutes les données dont une scène a besoin pour rouler et qui sont inutiles de partager à l'extérieur.

```
var explosion320 = new Scene(function(t, that) {
    that.localValues.sketch = {
        zoom : 3,
        rotationSpeed : 2
    };
    that.privateValues.superEllipse = {
        n2 : 3,
        sc : 20,
        scPow : 7,
        m : 8
    };
    that.runPhysicsEngine(t);
});
```

Ça me semble très positif de séparer les valeurs entre deux groupes : les valeurs qui sont communiquées à l'extérieur de l'objet et les valeurs qui y restent. Finalement, les données dans *privateValues* sont les différents paramètres qui servent à générer les données dans *localValues*.

6 L'array *localValues.graph*

L'array *localValues.graph* doit contenir 1000 objets différents déclarés ainsi :

```
var dot = {
    pos : createVector(10, 200),
    col : {
        r : 130,
        g : 255,
        b : 10
    },
    size : 10
};
this.localValues.graph.push(dot);
```

7 Nouvelles idées pour 0.03

L'objet *Scene* doit fonctionner d'une meilleure manière. Une scène doit pouvoir hériter d'une autre scène et avoir ses propres variantes. Des instances de l'objet *Scene* doivent pouvoir se déclarer ainsi :

```
//First, I declare a new Scene and add properties and methods to its prototype.

var greenSpiral = new Scene();
greenSpiral.setBackground(function(t){
    //Stuff that can dynamically generate the background gradient.
});
greenSpiral.setPalette(2430);
greenSpiral.setGraph(function(t){
    //Stuff that generates the graph of x and y positions dynamically.
});

//Now, I declare a new variable that will inherit from greenSpiral,
//but have a different background color.

var differentSpiral = new greenSpiral();
differentSpiral.setBackground(function(){
    //Stuff that defines a different background gradient than the prototype greenSpiral.
});
```

À l'intérieur de mon prototype original de l'objet *Scene*, je peux avoir de nombreuses commandes qui ouvrent la voie à la création de variantes.

```
if (!this.privateValues.paletteIndex){
    this.privateValues.paletteIndex = 514;
}
```

Ceci permet, par exemple, à mon objet *differentSpiral* d'hériter de tous les attributs et de toutes les méthodes de l'objet *greenSpiral*, mais d'avoir une palette de couleur différente.

Mon objet *Scene* pourrait donc ressembler à ça :

```
Scene = function(mainFunc) {
    this.mainFunc = func;
};

Scene.prototype.setBackground = function(backgroundFunc) {
    this.runBackground = backgroundFunc;
};

Scene.prototype.setColors = function(colorsFunc) {
    this.runColors = colorsFunc;
};

Scene.prototype.run = function(offset) {
    offset = (offset) ? offset  : 0;
    this.runBackground(drawCount - offset);
    this.runLayout(drawCount - offset);
    this.runGraph(drawCount - offset);
    this.runColors(drawCount - offset);
    this.makeGlobal();
};

Scene.prototype.runColors = function() {
    if (!this.privateValues.paletteIndex) {
            this.privateValues.paletteIndex = 1000;
    }

    if (allPalettes) {
        this.privateValues.palette = allPalettes[this.privateValues.paletteIndex];
    } else {
        this.privateValues.palette = palette;
    }

    this.privateValues.colorGraph = [];
    var currentColor = 0;
    for (var i = 0; i < 1000; i++) {
        var colorValues = hexToRgb(this.privateValues.palette[currentColor]);
        currentColor++;
        if (currentColor > 4) {
            currentColor = 0;
        }
        this.privateValues.colorGraph.push(colorValues);
    }
};
```

8 Nouvelles notes

Je dois me créer un système pour ajuster les niveaux de mes couleurs. Ça ressemblerait un peu à ça :

```
function adjustLevels(colInput, dark, mid, light) {
    var r = colInput.r;
    var g = colInput.g;
    var b = colInput.b;
    r = constrain(map(r, 0, 255, dark, 255), 0, 255);
    g = constrain(map(g, 0, 255, dark, 255), 0, 255);
    b = constrain(map(b, 0, 255, dark, 255), 0, 255);
    return {
        r : r,
        g : g,
        b : b
    };
}
```