E-ARTSUP.
DÉPARTEMENT DESIGN INTERACTIF & COMMUNICATION VISUELLE.

DEVELOPPEMENT ORIENTÉ OBJET [PARTIE 1/2]
(TABLEAUX ET CLASSES)

Digital Lab S.05 // 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 - Programmation Orientée Objet (OOP)

La programmation orientée objet (POO) est un paradigme de programmation conçu par Ole-Johan Dahl et Kristen Nygaard au début des années 1960 et poursuivi par les travaux d'Alan Kay dans les années 1970. Un objet représente un concept, une idée ou toute entité du monde physique. Un objet permet d'encapsuler un nombre des propriétés et méthodes pour faciliter le traitement de groupe.

Un objet est directement inspiré du monde réel. En effet nous sommes entouré d'objets et chacun de ces objets à des propriétés propres. Par exemple nous avons les objets "téléphones". Chacun de ses objets "téléphones" sont définis par le fait qu'ils ont tous des variables communes, à savoir le fait de pouvoir passer des communications vocales par le biais d'un réseau téléphonique. Mais ils possèdent aussi des variables qui leur sont propre comme le fait de pouvoir envoyer des sms ou non, d'être tactile ou pas, d'avoir des poids et des tailles qui diffèrent… Bref nos téléphones sont des objets appartenant à une même classe, la classe "Téléphone.

2 - Les objets

Un objet est directement inspiré du monde réel. En effet nous sommes entouré d'objets et chacun de ces objets à des propriétés propres. Par exemple nous avons les objets «téléphones». Chacun de ses objets «téléphones» sont définis par le fait qu'ils ont tous des variables communes, à savoir le fait de pouvoir passer des communications vocales par le biais d'un réseau téléphonique. Mais ils possèdent aussi des variables qui leur sont propre comme le fait de pouvoir envoyer des sms ou non, d'être tactile ou pas, d'avoir des poids et des tailles qui diffèrent… Bref nos téléphones sont des objets appartenant à une même classe, la classe «Téléphone».

¨Prenons un sketch de 200*200 dans lequel des balles de tailles différentes se déplacent à des vitesses différentes. Nous avons donc des «objets» qui ont un comportement commun, à savoir le fait d'être des balles et de se déplacer, mais aussi différents puisque leur vitesses diffèrent. Nous avons l'exemple parfait pour créer une «classe» d'objets.

Une classe est une matrice, un patron, un plan de construction… d'un objet. La classe d'un objet permet de créer et d'instancier les variables communes à tous nos objet, de développer leur comportements et de créer la méthode de construction de notre objet. D'un point de vu développement, elle est assez proche de la façon dont nous construisons un sketch.

Lorsque nous créons nos sketchs nous avons l'habitude de créer des variables, puis une méthode setup() qui nous permettra de créer notre sketch (taille, moteur de rendu…) et enfin une méthode draw() (notre boucle). Lorsque nous créons une classe nous obtenons à peu de choses près la même syntaxe :

```
class maClasse
{
	//Ici mes variables

	//Mon constructeur, l'équivalent de mon setup() pour mon
sketch
	Balle()
	{
	}

	//Mes méthode
	void display()
	{
	}
}
```

2 - Les objets

Entrons maintenant dans le vif du sujet en créant notre classe “Balle”. Pour cela nous allons reprendre l’ensemble de nos variables, nous les initialiserons dans notre constructeur et enfin nous dessinerons notre balle dans une méthode display(). Cela nous donne donc :

```
class Balle
{
  //variables globales
  float x; //position x de la balle
  float y; //position y de la balle
  float vx; //vitesse x de la balle
  float vy; //vitesse y de la balle
  float t; // taille de la balle

  //Mon constructeur, celui doit toujours avoir le même nom que la
  class afin d’être reconnu comme un constructeur.
   Balle()
   {
     //initialisation des variables.
     t = random(5, 10);
     x = random(t, width-t);
     y = random(t, height-t);
     vx = random(-3, 3);
     vy = random(-3, 3);
   }

   void display()
   {
     //mise à jour des variables
     x += vx;
     y += vy;

     //detection des murs
     if (x <= t || x >= width-t)
     {
       vx = -vx;
     }
     if (y <= t || y >= height-t)
     {
       vy = -vy;
     }
     fill(255);
     stroke(0);
     ellipse(x, y, t*2, t*2);
   }
}
```

2 - Les objets

Nous venons d'écrire notre classe mais comme nous avons vu, il s'agit ici de la matrice qui nous permettra de créer nos objets.

Il nous faut maintenant créer nos objets et les instancier pour ensuite les afficher sur la scène. Dans notre sketch, nous allons créer un nouvel objet "maBalle" faisant partie de la classe "Balle".

```
Balle maBalle;
```

Dans notre setup() nous instancierons notre objet.

```
maBalle = new Balle();
```

Enfin dans notre draw, nous appellerons la méthode display() de notre balle

```
maBalle.display();
```

```
void setup()
{
  size(400, 200, P2D);
  smooth();
  maBalle = new Balle();
}

void draw()
{
  background(255);
  maBalle.display();
}

class Balle
{
  //variables globales
  float x; //position x de la balle
  float y; //position y de la balle
  float vx; //vitesse x de la balle
  float vy; //vitesse y de la balle
  float t; // taille de la balle

  //Mon constructeur, celui doit toujours avoir le même nom que la class afin d'être reconnu comme un constructeur.
  Balle()
  {
    //initialisation des variables.
    t = random(5, 10);
    x = random(t, width-t);
    y = random(t, height-t);
    vx = random(-3, 3);
```

2 - Les objets

```
  vy = random(-3, 3);
}

void display()
{
  //mise à jour des variables
  x += vx;
  y += vy;

  //detection des murs
  if (x <= t || x >= width-t)
  {
    vx = -vx;
  }
  if (y <= t || y >= height-t)
  {
    vy = -vy;       fill(255);
  stroke(0);
  ellipse(x, y, t*2, t*2);
 }
}
```

On remarque que pour appeler une méthode de notre classe nous procédons de la manière suivante objet.methode().
Il en va de même pour accéder aux variables. Ainsi println(maBalle.x) me renverra la valeur x de ma balle. On peut par la suite aisément modifier ces valeurs.

Nous venons de créer notre première classe mais nous obtenons 1 balle et non 100 comme promis précédemment. Pour cela rien de plus simple, et c'est là le grand pouvoir des classes, il nous suffit de créer un tableau d'objets et non plus un seul objet.

```
Balle[] maBalle;
int nbBalle;

void setup()
{
  size(200, 200, P2D );
  smooth();
  nbBalle = 100;
  maBalle = new Balle[nbBalle];
  for(int i = 0; i < maBalle.length; i++)
  {
    maBalle[i] = new Balle();
  }
}

void draw()
{
  background(255);
  for(int i = 0; i < maBalle.length; i++)
  {
    maBalle[i].display();
  }
}
```

3 - ArrayList

Nous savons maintenant comment créer une classe et instancier des objets mais allons plus loin dans les possibilités que nous offre la programmation orientée objet. Nous venons de créer 100 objets mais que ce passe-t-il dans le cas où nous voulions en créer à chaque clic?
Il faudrait que nous rajoutions des éléments à notre tableau.

Imaginons un sketch simple, de 400*200 dans lequel nous pourrions rajouter une balle à chaque clic dont la position de départ sera définie par la position de la souris.

Pour cela nous allons toujours utiliser un tableau mais d'un autre type puisque nous utiliserons un ArrayList(). À l'inverse d'un tableau classique maBalle[] qui possède une taille fixe, les ArrayList sont des tableaux dont la taille n'est pas définie et peut être en constante expansion. Cependant sa déclaration diffère d'un tableau classique. Ainsi pour passer notre classe balle en ArrayList pour devrons changer notre code comme il suit :

```
int nbBalle;

void setup()
{
  size(200, 200, JAVA2D);
  smooth();
  nbBalle = 100;
  maBalle = new ArrayList<Balle>();
}
```

De même la methode nous permettant d'ajouter des balles sur notre scène diffère d'un tableau classique

```
for (int i=0; i<nbBalle; i++)
  {
    maBalle.add(new Balle());
  }
```

Enfin la dernière différence avec le tableau classique est la façon nous nous parcourons notre ArrayList afin d'appeler la méthode display() de notre classe.

```
for(int i = 0; i < maBalle.size(); i++)
  {
    Balle b = maBalle.get(i);
    b.display();
  }
```

3 - ArrayList

Maintenant que nous avons vu comment créer et parcourir un ArrayList, il ne nous reste plus qu'à modifier notre code précédent afin de créer nos balles à chaque clic de la souris

```
ArrayList<Balle> maBalle;
int nbBalle;

void setup()
{
  size(400, 200, P2D);
  smooth();
  nbBalle = 10;
  maBalle = new ArrayList<Balle>();
  for (int i=0; i<nbBalle; i++)
  {
    maBalle.add(new Balle()));
  }
}

void draw()
{
  background(255);
  if(mousePressed)
  {
    maBalle.add(new Balle());
  }
  for(int i = 0; i < maBalle.size(); i++)
  {
    Balle b = maBalle.get(i);
    b.display();
  }
}
```

Le résultat est là mais cependant il nous manque une règle à respecter. Nous voulions que nos balles apparaissent à la position de la souris, hors pour le moment elle apparaissent à des positions aléatoire. En effet lorsque l'on retourne dans le constructeur de la classe nous remarquons que notre x et y sont définies de manière aléatoire.

```
Balle()
 {
   //initialisation des variables.
   t = random(5, 10);
   x = random(t, width-t);
   y = random(t, height-t);
   vx = random(-3, 3);
   vy = random(-3, 3);
 }
```

Nous pourrions changer cela par un mouseX mouseY mais en faisant cela nos 10 première balles présentes sur scène auront le même point d'origine. Nous allons donc légèrement changer notre constructeur en lui indiquant qu'à son appel il devra recevoir des variables. Nous pourrons alors par la suite définir aisément la position de chacune de nos balles à la création.

3 - ArrayList

```
Balle(float x_, float y_)
 {
   //initialisation des variables.
   t = random(5, 10);
   x = x_;
   y = y_;
   vx = random(-3, 3);
   vy = random(-3, 3);
 }
```

Il nous faudra alors changer la façon dont nous instancions nos balles comme il suit

```
maBalle.add(new Balle(position x, position y));
```

Nous avons maintenant un tableau dynamique (ArrayList) et nous créons autant de balles que nous voulons.
Ajoutons maintenant une dernière possibilité, à savoir fixer une limite à notre tableau dynamique. En effet cela nous permettra d'économiser notre ordinateur qui risquerait, au bout d'un millions de particules, de ralentir.

Pour cela nous rajoutons une condition permettant de supprimer la première balle (la plus vieille) créée lorsque nous atteignons un maximum de 30 balles

```
if(maBalle.size() >= 30)
   {
     maBalle.remove(0);
   }
```

Nous voila donc avec notre première classe de particules.

4 - Clip Génératif

Un artiste vous demande de réaliser un clip génératif afin de faire la promotion de son titre.

Ensemble vous définirez le concept, l'univers storyboard... Chaque groupe se verra attribuer une partie du clip qu'ils devront developper. L'ensemble sera ensuite réuni pour ne former qu'un seul et même clip.

<u>Contraintes technique :</u>

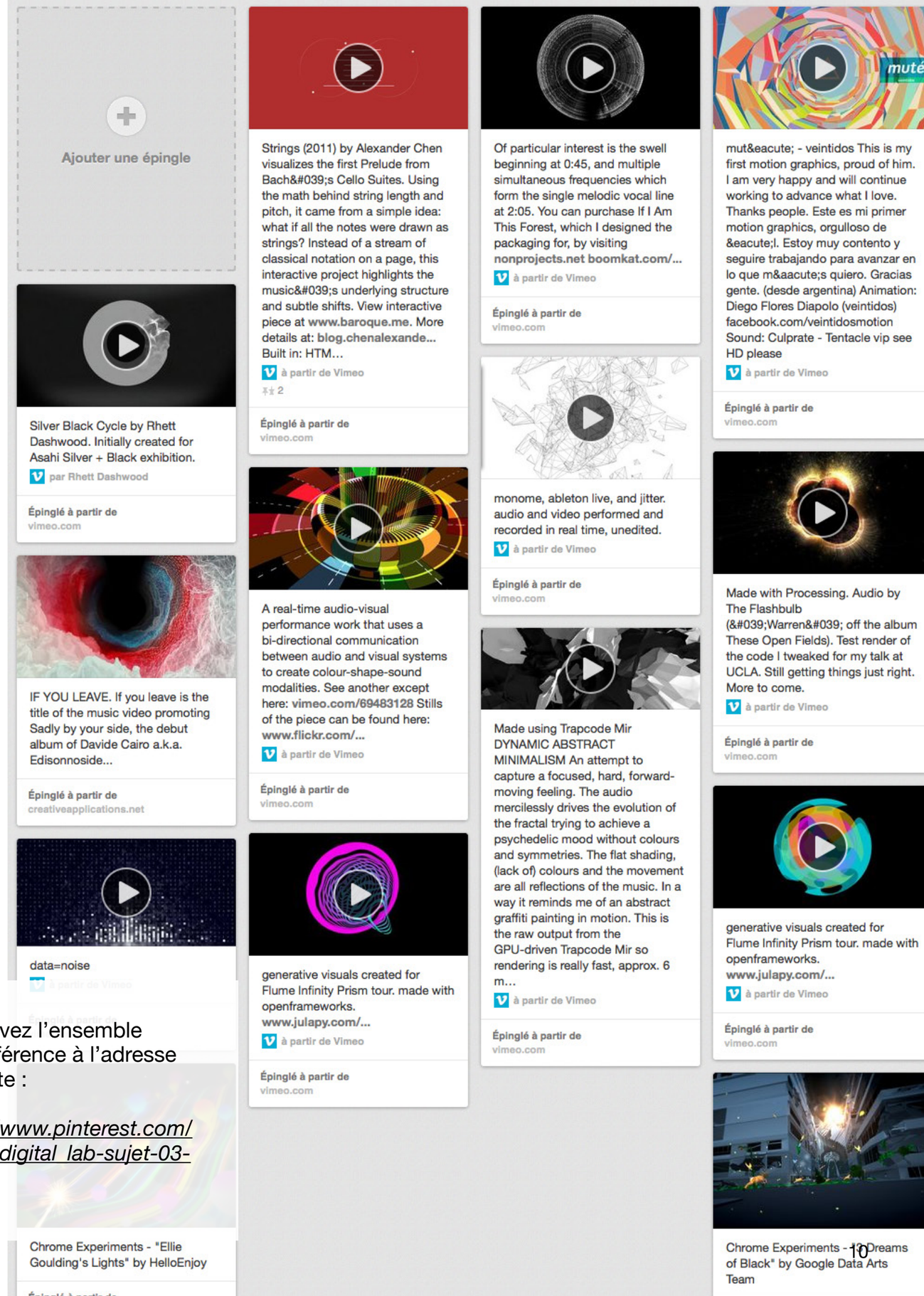
- **Format du Clip : 1280*720**

<u>Rendu :</u>

- **Images au format Jpg**
- **Sketchs de chacune de parties**
- **Document PDF présentant :**
  - **Concept global / Storyboard**
  - **Concept de chaque partie/groupe**
  - **Moodboard**
  - **Recherche graphique et technique**
  - **Images du rendu final**

Retrouvez l'ensemble des référence à l'adresse suivante :

<u>http://www.pinterest.com/alexr4/digital_lab-sujet-03-clip/</u>



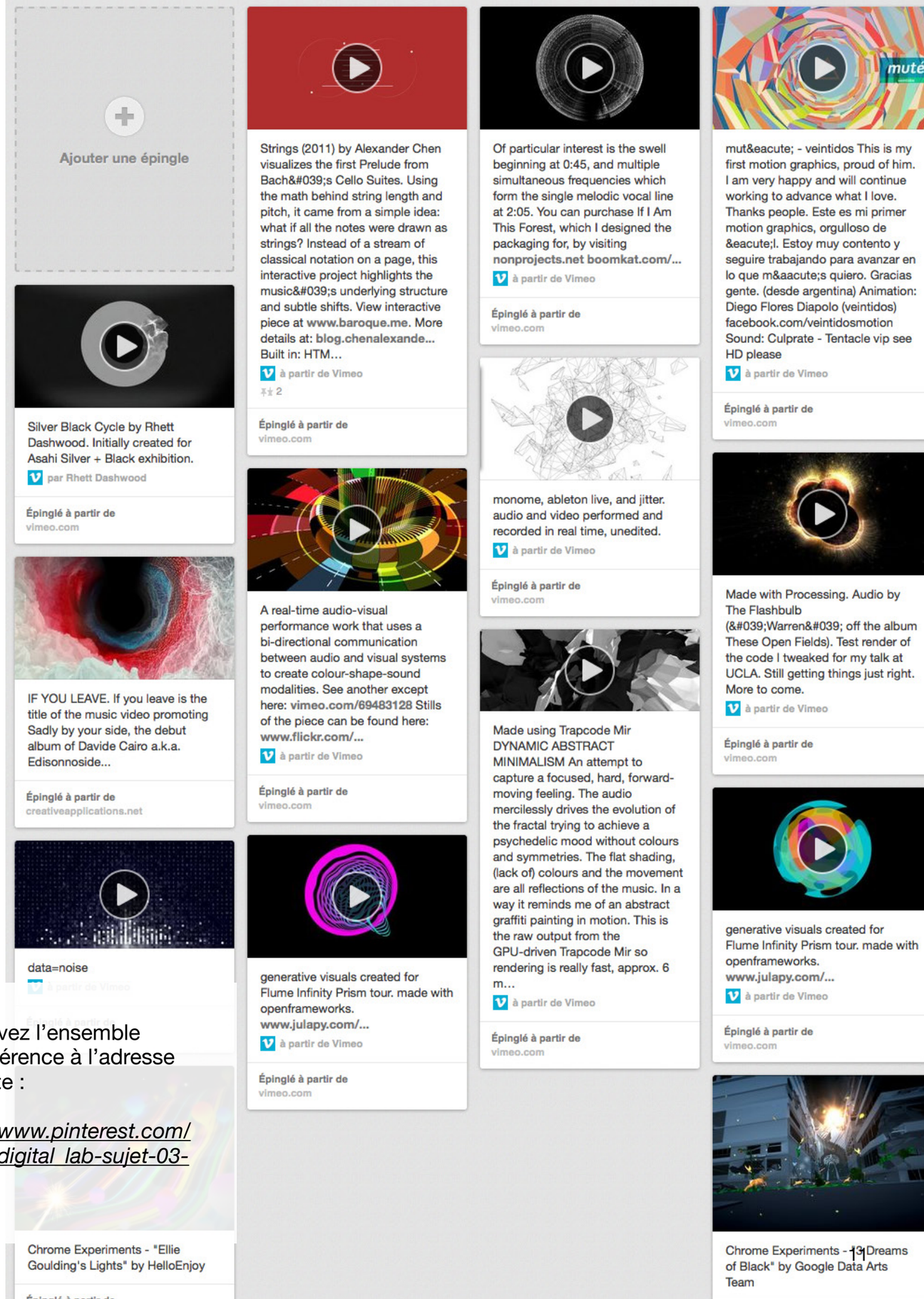
4 - Clip Génératif

Date de rendu :

26/11	Présentation des concepts et Moodboard
3/12	Storyboard Global et définition du CDC
3/12	Developpement
7/01	Developpement
7/01	Rendu
14/01	Projection

Retrouvez l'ensemble des référence à l'adresse suivante :

http://www.pinterest.com/alexr4/digital_lab-sujet-03-clip/



Contact

Alexandre Rivaux
Visual Designer & Partner Bonjour, interactive Lab

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