Chapitre 1 : Introduction

Image et Interaction 2D

Fabrice Aubert
fabrice.aubert@univ-lille.fr

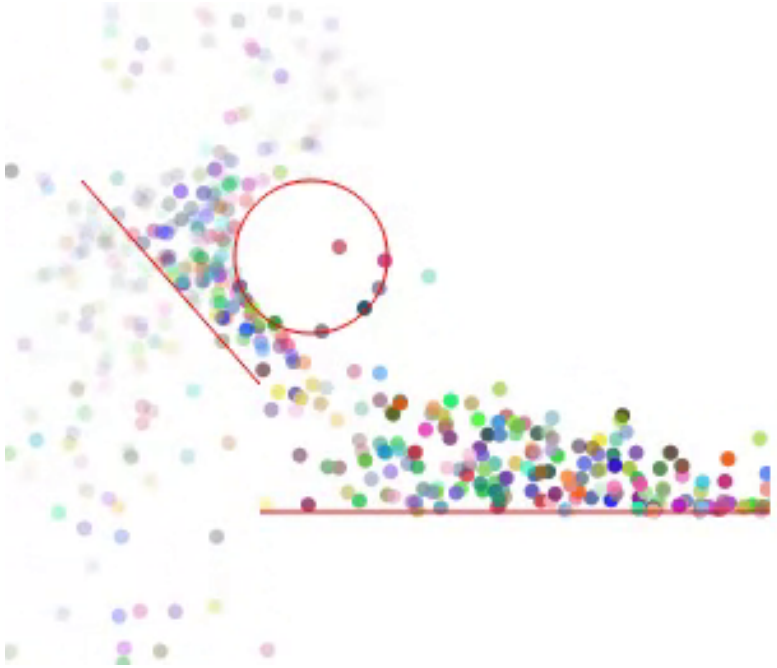
Licence Informatique 2018-2019

1 Objectifs de II2D

Introduction sur 3 thèmes

- Informatique Graphique (animation de particules 2D)
- Traitement d'Image (détermination de curseurs sur vidéo)
- Interaction Graphique (manipulation des données de l'animation)

Mini-projet support



- Permet d'aborder quelques ingrédients de l'informatique graphique :
  - **Application de** "Géométrie/Algèbre" (2D) : vecteur, position, localisation, intersection.
  - **Application de** "Mécanique" (du point) : vitesse, accélération, force.
  - "Algorithmique" : représentation et traitement de listes de points 2D.

2 Technique : HTML/Javascript

Affichage

- Utilisation de la balise HTML `canvas` comme "fenêtre graphique" :

```
<body>

<canvas id="un_canvas" width="500px" height="500px"></canvas>

</body>
```

- Affichage dans le canvas en javascript par la création d'un contexte 2D.

```
var canvas = document.getElementById('un_canvas');
var ctx = canvas.getContext('2d');

var draw=function() {
  ctx.fillStyle = 'rgb(200,0,0)'; // sets the color to fill in the rectangle with
  ctx.fillRect(10, 10, 55, 50);   // draws the rectangle at position 10, 10 with a width of 55 and a height of 50
}
```

$\Rightarrow$ voir https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D

Boucle d'affichage

- Pour réaliser une animation, on réitère les étapes suivantes :
  - Traiter les données (changer les positions, les couleurs, ...).
  - Tracer la "scène" :
    - Effacer le canvas avec une couleur de fond.
    - Tracer **tous les** éléments graphiques.
  - On recommence : boucle infinie **mais "non bloquante" et "avec synchronisation"**
- "non bloquante" : la "boucle infinie" ne doit pas bloquer le système (traitement des messages de la souris par exemple).
- "avec synchronisation" : laisser le système choisir quand rendre visible le tracé (double tampon, synchronisation verticale)
- $\Rightarrow$ utilisation de `window.requestAnimationFrame`

requestAnimationFrame

```
var draw = function() {
  ctx.clearRect(0,0,500,500); // effacer le canvas
  drawScene(); // tracer tous les éléments graphiques
}

var loop = function() {
   updateData(); // traiter les données
   draw(); // tracer
   window.requestAnimationFrame(loop); // recommencer
}
```

- `requestAnimationFrame(loop)` demande au système d'appeler notre fonction `loop` quand il sera prêt pour un nouvel affichage.
- la synchronisation est faite pour assurer 60 appels par seconde (donc 60 images par seconde).
- il faut complexifier la gestion de la boucle si le traitement des données est trop long pour assurer une fréquence stable.

Javascript et class

- Pour décomposer le code et le rendre (un peu) plus modulaire, on utilisera les classes javascript.

```
class Vector {
  constructor(x,y) {
    this.x=x;
    this.y=y;
  }

  // return the new vector p1+p2
  static add(p1,p2) {
    return new Vector(p1.x+p2.x,p1.y+p2.y);
  }

  // modify this = this + p
  add(p) {
    this.x+=p.x;
    this.y+=p.y;
    return this;
  }
}

var a=new Vector(2,2);
var b=new Vector(1,1);
var c=Vector.add(a,b);
a.add(b);
a.add(b).add(c);
a.add(Vector.add(b,c));
```

A noter :

- 1 seul constructeur possible.
- `static` pour les méthodes de classes

Affectation par référence vs par recopie (1/2)

Comparez :

```
var a=new Vector(0,0); // a est une référence au nouvel objet
var b=a; // b est affecté avec la référence de a
b.x=3; // modifie donc a.x
console.log(a.x); // affiche 3
```

et

```
var a=new Vector(0,0); // a est une référence au nouvel objet
var b=new Vector(a.x,a.y); // b est un clone de a (affectation par recopie)
b.x=3; // ne modifie pas a
console.log(a.x); // affiche 0
```

Affectation par référence vs par recopie (2/2)

Exemple de la classe d'un cercle.

```
class Circle {
  constructor(c,r) {
    this.center=new Vector(c.x,c.y); // clone de c
    this.radius=r;
  }
}
```

et non pas

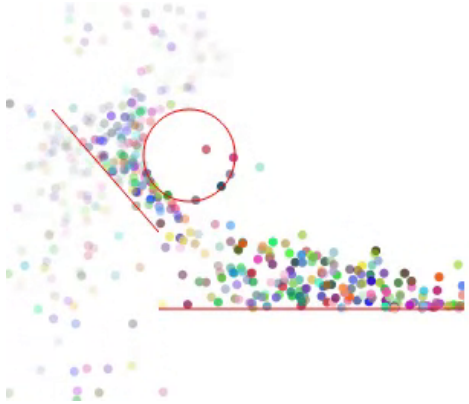
```
class Circle {
  constructor(c,r) {
    this.center=c; // référence sur c (à proscrire dans ce contexte)
    this.radius=r;
  }
}

var a=new Vector(0,0);
var c=new Circle(a,10);
a.x+=1;
// => effet de bord : le centre du cercle c est donc modifié...
```

Cependant l'affectation par référence reste judicieuse/nécessaire à de nombreuses occasions...

3 Mise en place de l'animation de particules

Cahier des charges



- Les particules sont des points qui sont créés au cours du temps à une certaine fréquence (nombre de naissances par seconde).
- Les particules disparaissent au cours du temps (selon une longévité donnée).
- Les données initiales des particules (position, couleur, etc) seront données par un générateur autorisant un facteur aléatoire.
- Les particules sont en mouvement au cours du temps : elles auront une vitesse (nombre de mètres par seconde)
- Le mouvement sera issu des forces appliquées (la gravité notamment) : les forces feront évoluer la vitesse.
- Les particules subissent des obstacles (collision avec rebonds).

Les unités de temps et de distances

- Pour simplifier, on ne fera pas du temps réel : c'est-à-dire que le temps dans l'animation ne correspondra pas au vrai temps.
- On choisit de faire évoluer notre propre temps entre chaque image générée en fixant un pas de temps arbitraire (noté `deltaTime` dans l'exemple qui suit).

```
var deltaTime=0.2; // il se passe 0.2s pour la simulation entre chaque image
var time=0; // temps initial de la simulation

var loop = function() {
    time+=deltaTime;
    updateData(); // traiter les données
    draw(); // tracer
    window.requestAnimationFrame(loop); // recommencer
}
```

- Remarque : la variable `time` n'est pas importante pour les traitements ; c'est surtout `deltaTime` qui interviendra.
- Pour les distances, pour alléger le discours, on confondra pixel et mètre (en fait peu importe l'unité choisie).

Position, Vitesse, Direction

- Une position de particule est représentée par un point 2D (données = coordonnées x,y).
- Une vitesse de particule est représentée par un vecteur 2D (données = mètres par seconde en x, mètres par seconde en y).
- Les points et vecteurs apparaitront également à d'autres occasions (par exemple, déterminer la direction du rebond d'une particule sur un obstacle).
- Remarque : un vecteur caractérise la notion de direction (représentation par une flêche) accompagnée d'une longueur (i.e. la norme du vecteur).
- Remarque : on ne note pas un vecteur par $\overrightarrow{u}$ mais par u (i.e. sans flêche) ; le vecteur $\overrightarrow{AB}$ sera noté $B-A$ (ce qui correspond à son calcul).
- Exercice : $A(2,3)$, $u(1,2)$, qu'est ce que $A+u$? Que donne et comment noter $\overrightarrow{AB}+\overrightarrow{BC}$?

Classe Vector

- En informatique graphique on représente point et vecteur par **une même classe** qu'on appellera ici `Vector` (distinguer par 2 classes aboutit à des lourdeurs inutiles).
- Pour un code clair, il est impératif d'abstraire tout ce qui concerne des données (x, y) par l'utilisation de cette classe `Vector`. Notamment on s'interdira de faire apparaitre explicitement des x et y inutiles dans le code (en dehors de la classe `Vector`). Par exemple :

```
Faire :
b=Vector.add(a,u)

Et non pas :
b.x=a.x+u.x
b.y=a.y+u.y
```

- La classe `Vector` constitue la brique rudimentaire de toute librairie graphique. Méthodes incontournables :
  - Permettre de faire $c = a + b$, $c = a - b$, $a+ = b$, $a- = b$, $b = k * a$ (k réel), $a* = k$
  - La longueur d'un vecteur $length(u) = \sqrt{u.x^2 + u.y^2}$
  - La distance entre 2 points A et B : $distance(A, B) = length(B - A)$
  - Et bien d'autres... (à enrichir selon les besoins de votre projet)

4 Gestion des particules

Proposition de décomposition (cf tp)

- Une classe `Particle` : représente une particule (attributs position, couleur, longévité, vitesse, etc ; méthodes pour afficher, ...)
- Une classe `ParticleManager` : gère la liste de l'ensemble des particules (principalement naissance, mort, et parcours de toutes les particules)
- Des classes `GeneratorPoint`, `GeneratorBox`, ... : se contente d'initialiser une particule naissante (position initiale dans une zone donnée notamment). Représente une "source" de particules.

Ajout/Suppression de particules

- Un des objectifs est de pouvoir avoir un nombre conséquent de particules avec éventuellement une fréquence de naissances et morts élevées.
- Faire alors un `new` d'une particule à chaque naissance et une suppression à chaque mort peut s'avérer sensible (notamment par le déclenchement arbitraire du garbage collector).
- Pour éviter cela, on affecte dès le départ un tableau de particules de **taille prédéfinie et fixe** : toutes les particules sont allouées au lancement de l'application.
- Cela nécessite un attribut supplémentaire pour la classe `Particle` (un booléen `isAlive` par exemple pour savoir si elle est active ou non) :
  - Pour une naissance `isAlive` passe à `true`
  - Pour une mort `isAlive` passe à `false`.

Générateur

- Représente une source de particules : fréquence de naissance, particules qui restent à naitre, initialisation d'une particule.
- Ce n'est pas le générateur qui gère la liste des particules (c'est le `ParticleManager`).

```
class GeneratorPoint {
  constructor() {
    this.nbBirth=0;     // nombre de particules à naitre
    this.birthRate=20; // fréquence de naissance
    this.position=new Vector(0,0); // position de la source
    this.vMin=new Vector(0,0); // vMin, vMax : bornes des vitesses aléatoires
    this.vMax=new Vector(0,0);
    this.minTimeLeft=30; // longévité minimale
    this.maxTimeLeft=31; // longévité maximale
  }

  initParticle(p) {
    p.position.setVector(this.position);

    p.velocity.x=randInt(this.vMin.x,this.vMax.x);
    p.velocity.y=-randInt(this.vMin.x,this.vMax.y);

    p.color={r:randInt(10,255),g:randInt(10,255),b:randInt(10,255)};

    p.lifeLeft=randInt(this.minTimeLeft,this.maxTimeLeft);
  }
};
```

Démarche pour la gestion des particules (mort/naissance)

- A chaque `updateData` (cf boucle d'affichage) :
  - On met à jour le nombre de particules à naitre pour chaque générateur.
  - Pour chaque particule (parcours du tableau de particules) :
    - Si la particule est en vie : on décroit son reste à vivre (meurt éventuellement)
    - Si la particule est morte : s'il reste des particules à naitre pour un générateur, on l'initialise (naissance)