Boids

Guillaume Wisniewski
guillaume.wisniewski@limsi.fr

octobre 2015

L'objectif du TP est de ré-implémenter le programme *Boids*. Boids est le nom d'un programme informatique de vie artificielle, développé par Craig W. Reynolds en 1986, simulant le comportement d'une nuée d'oiseaux en vol. Il a marqué l'histoire des sciences en montrant qu'un comportement complexe (le vol coordonné d'une nuée d'oiseaux) pouvait *émerger* de règles locales très simples (chaque oiseau adopte son comportement en fonction de celui de ses voisins immédiats). Aujourd'hui des techniques similaires sont utilisées pour animer automatiquement le comportement de foules dans des jeux vidéos ou des films [1] (*Le roi lion*, *Le seigneur des anneaux*, ...) ou simuler le comportement d'une foule (simulation multi-agent utilisé, par exemple, pour simuler l'évacuation d'un bâtiment).

1 Bibliothèque graphique

Tous les aspects graphiques du programme (affichage, interaction avec la souris, ...) seront gérés par la bibliothèque `Processing` [2]. `Processing` est une bibliothèque java permettant de créer facilement des images, des animations et de gérer les interactions. Le code concernant la partie graphique vous est fourni. Il faut cependant télécharger le code la bibliothèque sur le site du cours et indiquer au compilateur qu'il doit considérer celle-ci. Pour cela, glisser-déplacer le fichier `core.jar` à la racine du projet et indiquer au compilateur d'utiliser la bibliothèque (clique droit sur `core.jar` puis `BUILD PATH`

1. `http://www.massivesoftware.com/`, la compagnie qui a réalisé certains effets spéciaux du film *Le seigneur des anneaux* à l'aide de ce type de technique
2. `http://processing.org`

→ `ADD TO BUILD PATH`). Il faut également ajouter la ligne suivante aux débuts de vos fichiers java :

```
import processing.core.*;
```

Nous utiliserons tout au long du programme des vecteurs de $\mathbb{N}^2$ qui sont modélisés par la classe `PVector` de Processing. Il est *fortement* recommandé de lire la documentation [3] de cette classe avant de commencer le TP.

2 Architecture du programme

Le programme comportera trois classes :

— la classe `BoidGUI` constituera le programme principal chargé de la création de l'interface graphique et de l'animation ;

— la classe `Boid` modélisera *un* oiseau ;

— la classe `Flock` représentera un groupe (nuée) d'oiseau.

Seule les deux dernières classes sont à réaliser.

Le TP se déroulera en quatre étapes :

1. création d'une classe `RandomBoid` représentant un oiseau volant au hasard ;
2. création de la classe `Flock` ;
3. réalisation de l'interface graphique ;
4. création de la classe `Boid` représentant un oiseau ≪ intelligent ≫ qui vole en respectant certaines règles.

La classe `RandomBoid` a deux intérêts : avoir un point de comparaison permettant de juger la pertinence des règles proposées par Reynolds, mais surtout de pouvoir s'assurer rapidement que l'architecture du programme est correcte avant de commencer l'implémentation, plus difficile, des règles de Reynolds.

3 La classe `RandomBoid`

3.1 Attributs et constructeurs

Un oiseau peut être vu comme un point soumis à ensemble de forces et évoluant en fonction des lois classiques de la physique. Chaque oiseau est décrit par :

— sa position (un vecteur de $\mathbb{N}^2$) décrite par un attribut `pos` ;

— sa vitesse (un vecteur de $\mathbb{N}^2$) décrite par un attribut `vel` ;

— sa vitesse maximale (un `float`) ;

— la force maximale à laquelle il peut être soumis (un `float`) ;

3. accessible en utilisant n'importe quel moteur de recherche avec la requête ≪ processing PVector ≫

Les vecteurs de $\mathbb{N}^2$ seront représentés par des instances de la classe `PVector` de la bibliothèque `Processing`.

Deux constructeurs sont définis : un permettant de créer un oiseau en ne spécifiant que deux entiers correspondant à sa position et un autre permettant de spécifier, *en plus* sa vitesse maximale et sa force maximale. Par défaut la vitesse maximale est égale à 2 et la force maximale à $0,05$.[4] La vitesse est initialisée de manière aléatoire : chaque coordonnée du vecteur vitesse est choisie aléatoirement entre -1 et +1. Il en possible, en Java, de tirer un nombre aléatoire compris entre `borneSup` et `borneInf` à l'aide des instructions :

```
private Random randGen = new Random();
// ...
int v = this.randGen.nextInt(borneSup - borneInf) + borneInf;
```

① Écrivez les attributs et les constructeurs de la classe `RandomBoid`.

② Pourquoi les attributs de la classe doivent-ils être en `private` ?

3.2 Force et mise-à-jour de la position

À chaque instant, l'oiseau est soumis à une force, représentée par un vecteur de $\mathbb{N}^2$. Les composantes de cette force sont des entiers choisis au hasard entre -10 et 10.

Il est possible de déterminer la nouvelle position de l'oiseau en appliquant les lois classiques de la mécanique newtonienne :

— l'accélération a subie à un instant donné est égale à la somme des forces (la masse d'un oiseau est supposée égale à 1) ;

— après un intervalle de temps $\partial t = 1$, la nouvelle vitesse est :

$$v_{t+1} = v_t + a$$

— d'une manière similaire, la nouvelle position est donnée par :

$$x_{t+1} = x_t + v_{t+1}$$

Il est important de limiter la vitesse à chaque mise-à-jour de celle-ci.

Nous allons maintenant finir d'implémenter la classe `RandomBoid` :

③ Écrivez une méthode `PVector randomForce()` qui détermine la force à laquelle l'oiseau est soumis à un instant donné ;

④ Écrivez une méthode `void updatePosition()` qui détermine la nouvelle position d'un oiseau en fonction de la force à laquelle il est soumis ;

4. Pensez à regarder la documentation de la classe `PVector` et notamment la méthode `limit` pour appliquer la limite.

3.3 Définition de l'affichage

Pour pouvoir visualiser le comportement de la nuée, il est nécessaire d'ajouter à votre code les méthodes :

— `void run();`

— `void render();`

— `void borders();`

Ces méthodes sont disponibles sur le site web du cours. Toutes les opérations d'affichage à l'écran se font par l'intermédiaire d'une instance de la classe `PApplet` que l'on appelle généralement le *contexte graphique*.

Il est nécessaire d'ajouter un attribut pour stocker la référence vers le contexte graphique et de modifier le constructeur pour initialiser celle-ci.

⑤ Faites les modifications nécessaires à la classe `RandomBoid`.

Vous pouvez ensuite utiliser la classe `OneBoidGUI` pour voir votre oiseau voler.

4 La classe `RandomFlock`

La classe `RandomFlock` va nous permettre de représenter une nuée (un ensemble) d'oiseaux. Elle comporte deux attributs :

— un attribut de type `PApplet` représentant le contexte graphique dans lequel sera représenté la nuée ;

— un attribut de type `ArrayList` qui stockera l'ensemble des oiseaux (des `RandomBoid`) constituant la nuée.

Le constructeur permet d'initialiser les différents attributs. En particulier, il crée n oiseaux (n est un paramètre du constructeur).

⑥ Écrivez les attributs et le constructeur de la classe `RandomFlock`.

⑦ Ajoutez une méthode `addBoid(int x, int y)` qui permet d'ajouter à la nuée un nouvel oiseau dont la position initiale est passée en paramètre.

⑧ Ajoutez une méthode `void run()` qui appelle la méthode `run` de chaque oiseau de la nuée.

La classe `RandomGUI` vous permet de visualiser le vol de votre nuée.

5 Classe `Boid`

Nous allons maintenant modifier les classes précédentes afin de pouvoir implémenter les boids de Reynolds. Nous allons dans un premier temps ajouter un certains nombres de méthodes outils, puis définir les trois forces utilisées dans l'approche de Reynolds.

⑨ *Copier* la classe `RandomBoid` en `Boid` et la classe `RandomFlock` en `Flock`. Dans cette dernière, toutes les occurences de `RandomBoid` devront être remplacées par des `Boid`.

5.1 Méthodes outils

⑩ Ajoutez à la classe `Boid` une méthode `distance` permettant de calculer la distance entre cette instance et un autre `Boid` ;

⑪ Comment la méthode `distance` peut-elle accéder aux attributs `private` du `Boid` passé en paramètre ?

⑫ Ajoutez à la classe `Flock` une méthode dont la signature est `ArrayList<Boid> neighbors(Boid b, double neighborDist)` qui renvoie l'ensemble des oiseaux de la nuée situés à une distance inférieure à `neighborDist` d'un oiseau donné.

⑬ Pourquoi la méthode `neighbors` est-elle ajoutée à la classe `Flock` et non pas à la classe `Boid` ?

⑭ Modifiez la class `Boid` pour ajouter un attribut de type `Flock` représentant la nuée à laquelle appartient cet oiseau et mettez le constructeur à jour pour initialiser cet attribut.

5.2 Force d'alignement

La premiere force introduite par Reynolds est une force traduisant le fait que les oiseaux d'une nuée ont tendance à voler dans la même direction (Figure 1).

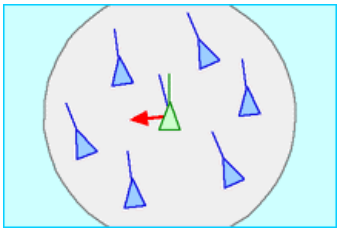

FIGURE 1 – Représentation symbolique de la force d'alignement (toutes les illustrations viennent du site de Reynolds)

Cette force est directement liée à la moyenne des vitesses des oiseaux volant dans le voisinage d'un oiseau :

$$F_{\text{alignement}}(b) = \frac{1}{n} \sum_{b' \in \text{voisinage}(b)} \text{vitesse(b')}$$

où voisinage(b) est l'ensemble des oiseaux situés à une distance inférieure à 25 de l'oiseau b.

Pour garantir l'homogénéité des forces, il est en plus nécessaire de normaliser $F_{\text{alignement}}$ en :

— normalisant le vecteur [5] ;

— en le multipliant par la vitesse maximale ;

5. Pensez à regarder la documentation de la classe `PVector` !

— en lui soustrayant la vitesse de l'oiseau ;
— en limitant sa norme à la force maximale que peut subir l'oiseau.

(15) Ajoutez une méthode `PVector align()` retournant la force d'alignement s'appliquant à un oiseau donné.

5.3 Force de séparation

La force de séparation traduit le fait que deux oiseaux ne peuvent pas être trop proches : dès que deux oiseaux sont suffisamment proches, ils changent leur direction pour éviter la collision.

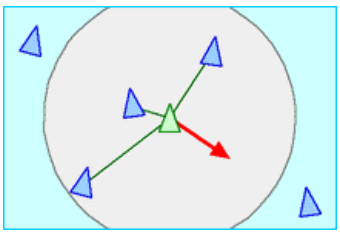

FIGURE 2 – Principe de la force de séparation

Cette force est déterminée de la manière suivante :

$$F_{\text{séparation}}(b) = \frac{1}{n} \times \sum_{b' \in \text{voisins}(b)} \frac{1}{d(b,b')} \left(\text{pos}(b) - \text{pos}(b')\right)$$

où b est l'oiseau pour lequel on est en train de calculer $F_{\text{séparation}}$, voisins(b) l'ensemble des oiseaux situés à une distance inférieures à 20, n le cardinal de cet ensemble, pos(b') la position d'un oiseau et $d(b,b')$ la distance entre deux oiseaux. Pour des raisons d'homogénéité il est important que la différence entre les vecteurs de positions soit normalisée avant d'être sommées.

La force doit ensuite être normalisée en suivant exactement la même procédure que pour $F_{\text{alignement}}$.

(16) Ajoutez une méthode `PVector separate()` retournant la force de séparation s'appliquant à un oiseau donné.

5.4 Force de cohésion

La force de cohésion pousse un oiseau à se déplacer vers la position moyenne de ses voisins.

Formellement, cette force est définie par :

$$F_{\text{cohésion}}(b) = \text{steer}(\frac{1}{n} \sum_{b' \in \text{voisinage}(b)} \text{pos}(b'))$$

Le voisinage est pris sur une distance de 25 ; steer est une méthode (fournie) qui pousse l'oiseau à se déplacer vers une position donnée.

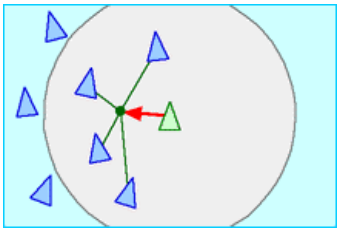

FIGURE 3 – Illustration du principe de la force de cohésion

⑰ Ajoutez une méthode `PVector cohesion()` retournant la force de séparation s'appliquant à un oiseau donné.

⑱ Pourquoi la méthode `steer` était-elle déclarée en `private` ?

5.5 Déplacement des oiseaux

⑲ Modifiez la méthode `updatePosition` pour tenir compte des trois nouvelles forces. La force totale appliquée à un oiseau est la somme pondérée des trois forces :

$$F = F_{\text{alignement}} + F_{\text{cohésion}} + \frac{3}{2} \times F_{\text{séparation}}$$

⑳ Observez ce qu'il se passe lorsque l'on supprime une des trois forces.