Boids

Maël Forgeoux, Louison Berthe

December 2024

1 Utilisation du programme

1.1 Exemple sans les fonctionnalités

Nous avons implémenté la simulation ainsi que chacun des points optionnels dans la section "pour aller plus loin". Par défaut la simulation se déroule sans ces modification à l'exception de deux modification personnelles :

- La différenciation par la couleur de certains boids, l'un est en rouge et ceux considérés par ce boid de référence sont affichés en vert mais ces modifications ne sont que visuelles
- La possibilité de mettre la simulation en pause et de la reprendre en utilisant la touche espace

Ainsi une simulation minimale avec des coefficients donnant une résultat satisfaisant peut s'obtenir avec les paramètres suivants :

```
main.py 100 10 0.0005 10 100 0
```

La commande générale sans modifications mais avec tous les arguments est la suivante :

```
main.py 100 10 0.0005 10 50 0 False 0 True 10 1 0 0 3.2 -1
```

1.2 Fonctionnalités activables

1.2.1 Cône de vision

La première option ajouté est la possibilité d'activer la vision sous forme de cône avec les deux paramètres suivants :

- un booléen "True" ou "False", qui vaut "True" pour activer le cône
- un angle en radians pour le cône de vision.

Pour utiliser cette option uniquement on peut utiliser :

```
main.py 100 10 0.0005 10 100 0 True 4.5
```

```
main.py 100 10 0.0005 10 100 0 True 4.5 True 10 1 0 0 3.2 -1
```

1.2.2 Type d'espace

Ensuite viens la possibilité de se placer dans un espace non-torique et de faire rebondir les boids sur des parois au lieu de se téléporter de l'autre coté de la fenêtre. Et cela avec :

- un booléen qui vaut vrai si l'espace est un tore False sinon
- un paramètre padding qui vaut 50 si on ne précise pas et qui correspond au nombre de pixels entre la fenêtre et les murs
- un coefficient de force du rebondissement qui multiplie l'accélération au rebondissement et vaut par défaut 1

```
main.py 100 10 0.0005 10 100 0 False 0 False 10 1
```

```
main.py 100 10 0.0005 10 100 0 False 0 False 10 1 0 0 3.2 -1
```

1.2.3 Vent

Il est aussi possible d'ajouter du vent à la simulation, qui sera subit par tous les boids uniformément avec :

- un coefficient de force qui vaut par défaut 0 ce qui implique qu'il n'y a pas de vent
- un angle qui est donné par rapport à l'horizontale vers la droite

```
main.py 100 10 0.0005 10 100 0 False 0 True 10 1 0.5 0.8
```

```
main.py 100 10 0.0005 10 100 0 False 0 True 10 1 0.5 0.8 3.2 -1
```

1.2.4 Limite de rotation

Nous pouvons aussi limiter la rotation effectuable par chacun des boids entre chaque image de la simulation avec :

- un angle limite qui vaut par défaut 3.2 ce qui ne limite pas la rotation

Cette option n'influe que très peu dans le cas de la simulation standard car les boids ne changement pas de direction rapidement ou les empêche quasiment totalement de tourner.

```
main.py 100 10 0.0005 10 100 0 False 0 True 10 1 0 0 0.1
```

```
main.py 100 10 0.0005 10 100 0 False 0 True 10 1 0 0 0.1 -1
```

1.2.5 Fermeture prédeterminée de la fenêtre

Le dernier argument permet d'ajouter un temps limite en secondes, une fois ce temps atteint la simulation se ferme d'elle même :

- un temps en secondes qui vaut -1 si la simulation ne doit pas s'arrêter sans interruption manuelle

```
main.py 100 10 0.0005 10 100 0 False 0 True 10 1 0 0 3.2 7
```

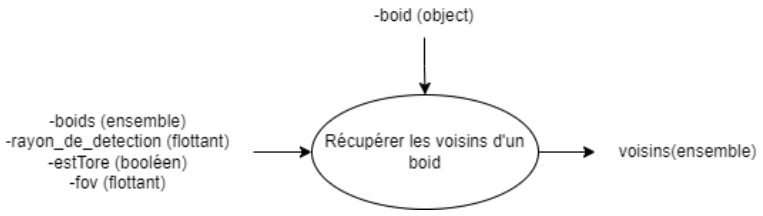
```
main.py 100 10 0.0005 10 100 0 False 0 True 10 1 0 0 3.2 7
```

2 Algorithmes

Liste des Algorithmes

2.1 Obtenir les voisins d'un boid

Afin de pouvoir faire interagir les boids entre eux, il est nécessaire d'écrire un algorithme permettant d'obtenir l'ensemble des boids qui doivent interagir avec un certain boid et apliquer cette fonctions à chaque boid



Algorithme 1 : Récupérer les voisins d'un boid

Entrées :

- *boid* : le boid dont on veut connaître les voisins

Données :

- *boids* : ensemble, tous les boids
- *rayon_de_detection* : flottant, distance à laquelle le boid peut en détecter un autre
- $estTore$: booléen, indique si l'espace est un tore
- fov : flottant, représentant l'amplitude du champ de vision du boid

Variables :

- $\vec{d}$: vecteur, distance entre le boid de référence et le voisin potentiel considéré
- ang_diff: flottant, différence angulaire entre la direction du boid de référence (et donc centre de sa vision) et la position relative du voisin considéré par rapport à la sienne

Sorties :

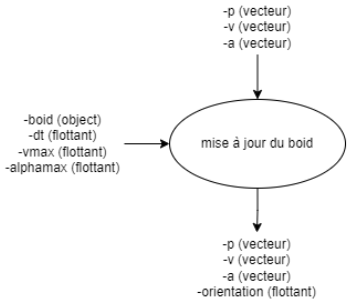
- *voisins* : L'ensemble des voisins du boid

début

```
voisins ← ∅
pour chaque autre_boid dans boids faire
    d⃗ ← distance(boid.position, autre_boid.position, largeur, hauteur, estTore)
    si ‖d⃗‖ ≤ rayon_de_detection alors
        ang_diff ← difference_angle(angle(v⃗), angle(d⃗))
        si estTore alors
            voisins ← voisins ∪ {autre_boid}
si boid dans voisins alors
    voisins ← voisins\{boid}
retourner voisins
```

2.2 Mettre à jour l'état d'un boid

Une fois les forces calculées et donc l'accélération, il est nécessaire de remonter à la vitesse puis à la position et ce en multipliant par dx pour intégrer. On utilise aussi le vecteur vitesse pour actualiser la direction du boid afin de pouvroi le représenter graphiquement.



Algorithme 2 : Mettre à jour l'état d'un boid

Entrées :

- *boid* : objet, le boid que l'on doit mettre à jour
- dt : flottant, temps estimé depuis la dernière image générée
- v_{max} : flottant, la vitesse maximale du boid
- α_{max} : flottant, la rotation maximale que peut effectuer le boid pendant le temps dt

Données :

- $\vec{p}$: vecteur, la position du boid
- $\vec{v}$: vecteur, sa vitesse
- $\vec{a}$: vecteur, son accélération

Résultat : Mise à jour de :

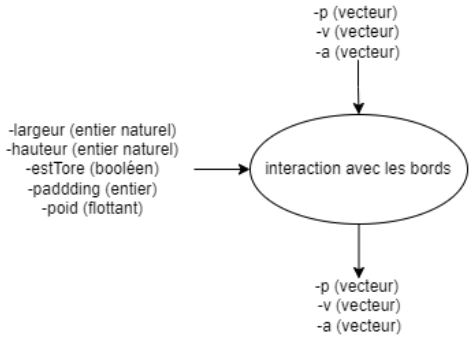
- $\vec{p}$: vecteur, la position du boid
- $\vec{v}$: vecteur, sa vitesse
- $\vec{a}$: vecteur, son accélération
- *orientation* : flottant, son orientation dans l'espace

début

$\vec{v}_{ini} \leftarrow \vec{v}$
$\vec{v} \leftarrow \vec{v} + dt \cdot \vec{a}$
$\vec{v} \leftarrow \text{limiter\_angle}(\vec{v}_{ini}, \vec{v}, rotation\_max)$
si $\|\vec{v}\| > vitesse\_max$ **alors**
  $\vec{v} \leftarrow \frac{\vec{v}}{\|\vec{v}\|} \cdot vitesse\_max$
$\vec{p} \leftarrow \vec{p} + dt \cdot \vec{v}$
$orientation \leftarrow \text{angle}(\vec{v}) + \frac{\pi}{2}$
$\vec{a} \leftarrow \vec{0}$

2.3 Faire interagir le boid avec les bords

Selon les options choisies les boids interagissent différemment avec les bords, ils peuvent soit se téléporter de l'autre coté sans modification de la vitesse ou de l'accélération pour simuler un tore. Soit rebondir sur ce bord en modifiant son accélération grâce à un coefficient et en symétrisant sa vitesse par rapport au bord rencontré.



Algorithme 3 : Fonction de rebond d'un boid

Entrées :

- *largeur* : entier naturel, nombre de pixels en largeur de la fenêtre
- *hauteur* : entier naturel, nombre de pixels en hauteur de la fenêtre
- *estTore* : booléen, indique si l'espace est un tore
- *padding* : entier naturel, nombre de pixels séparant la fenêtre des bords rebondissants
- *poid* : flottant, coefficient sur l'accélération gagnée en rebondissant

Données :

- $\vec{p} = \begin{pmatrix} x \\ y \end{pmatrix}$: vecteur, la position du boid
- $\vec{v} = \begin{pmatrix} v_x \\ v_y \end{pmatrix}$: vecteur, sa vitesse
- $\vec{a} = \begin{pmatrix} a_x \\ a_y \end{pmatrix}$: vecteur, son accélération

Résultat : Mise à jour de :

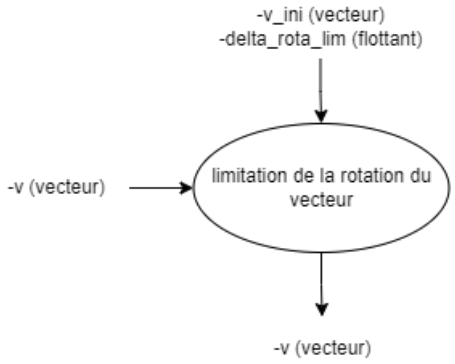
- $\vec{p} = \begin{pmatrix} x \\ y \end{pmatrix}$: vecteur, la position du boid
- $\vec{v} = \begin{pmatrix} v_x \\ v_y \end{pmatrix}$: vecteur, sa vitesse
- $\vec{a} = \begin{pmatrix} a_x \\ a_y \end{pmatrix}$: vecteur, son accélération

début

- **si** *l'espace est un tore* **alors**
  - $x \leftarrow x \mod largeur$
  - $y \leftarrow y \mod hauteur$
- **sinon**
  - **si** $x < padding$ **alors**
    - $x \leftarrow 2 \times padding - x$
    - $v_x \leftarrow |v_x|$
    - $a_x \leftarrow -poids \times a_x$
  - **sinon si** $x > largeur$ - $padding$ **alors**
    - $x \leftarrow 2 \times (largeur - padding) - x$
    - $v_x \leftarrow -|v_x|$
    - $a_x \leftarrow -poids \times a_x$
  - **si** $y < padding$ **alors**
    - $y \leftarrow 2 \times padding - y$
    - $v_y \leftarrow |v_y|$
    - $a_y \leftarrow -poids \times a_y$
  - **sinon si** $y > hauteur$ - $padding$ **alors**
    - $y \leftarrow 2 \times (hauteur - adding) - y$
    - $v_y \leftarrow -|v_y|$
    - $a_y \leftarrow -poids \times a_y$

2.4 Limiter la rotation des boids

Afin de limiter la rotation des boids, on limite en réalité la rotation de leur vecteur vitesse car c'est ce vecteur qui donne son orientation aux boids.



Algorithme 4 : Limitation la rotation d'un vecteur

Données :

- $\vec{v} = \begin{pmatrix} x \\ y \end{pmatrix}$: vecteur, vecteur dont il faut limiter la rotation

Entrées :

- $\vec{v}_{ini}$: vecteur, vecteur de référence pour la rotation
- $\Delta_{\alpha,lim}$: flottant, angle de différence limite entre ces deux vecteurs à ne pas dépasser

Variables :

- α : flottant, orientation du vecteur
- α_{ini} : flottant, orientation du vecteur précédemment
- Δ_α : flottant, différence entre les orientations des deux vecteurs
- $sens$: entier, -1 ou 1 permet de connaitre la direction de rotation du vecteur
- α_{fin} : flottant, orientation actualisée du vecteur pour respecter la limite de rotation
- n : flottant, norme du vecteur considéré

Résultat : Mise à jour de :

- $\vec{v} = \begin{pmatrix} x \\ y \end{pmatrix}$: vecteur, vecteur initial mais respectant la limte de rotation

début

$\alpha \leftarrow \text{angle}(\vec{v})$
$\alpha_{ini} \leftarrow \text{angle}(\vec{v}_{ini})$
$\Delta_\alpha \leftarrow ((\alpha - \alpha_{ini} + \pi) \bmod (2\pi)) - \pi$
si $|\Delta_\alpha| > \Delta_{\alpha,lim}$ **alors**
  si $\Delta_\alpha < 0$ **alors**
    $sens \leftarrow -1$
  sinon
    $sens \leftarrow 1$
  $\alpha_{fin} \leftarrow \alpha + \alpha_{lim} \times sens$
  $n \leftarrow ||\vec{v}||$
  $x \leftarrow n \times \cos(\alpha_{fin})$
  $y \leftarrow n \times \sin(\alpha_{fin})$