Programmation fonctionnelle avancée

Projet: Boids: les moutons de Panurge

L2 – Université Nice Sophia Antipolis

Le projet est à rendre au plus tard le 21 décembre minuit.

Il y aura 2 points de pénalité par jour de retard.

- envoi des projets par mail à elozes@unice.fr
- un seul mail par binôme/trinôme, en mettant en copie les autres participants dans le mail
- indiquez les noms, prénoms, et numéros d'étudiant de chacun dans le mail
- le projet est à rendre en un seul fichier archive **.zip** ou **.tgz** contenant tous les fichiers demandés (fichiers `.rkt`, mais aussi rapport au format texte ascii ou PDF).

Table des matières

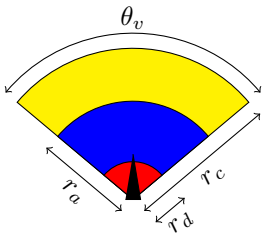

FIGURE 1 – Les zones de distanciation (en rouge), alignement (en bleu) et cohésion (en jaune).

1 Le modèle des « *boids* »

« Boids » est le nom d'un programme informatique de vie artificielle, développé par Craig W. Reynolds en 1986 [Rey87], simulant le comportement d'une nuée d'oiseaux en vol. Le mot boid est une contraction de bird-oid (qui a la forme d'un oiseau). Les boids sont souvent utilisés pour créer des images de synthèse car ils fournissent des représentations réalistes de nuées d'oiseaux, bancs de poissons, essaims d'insectes ou autres troupeaux d'animaux. Pour simplifier, on se place dans un espace bidimensionnel et on représente un boid par un triangle.

1.1 Les trois zones

La nuée est un ensemble de boids en mouvement. Chaque boids évolue en obéissant à trois règles simples :

distanciation il faut éviter d'être trop proche des boids voisins

alignement il faut aller dans la même direction que les boids voisins

cohésion il faut se rapprocher du cœur de la nuée des boids voisins

Dans les faits, cela se traduit par la réaction du boid à son voisinage. Celui-ci se divise en trois zones disjointes, de la plus proche à la plus éloignée : une zone de distanciation, une zone d'alignement et une zone de cohésion. Lorsqu'un voisin entre dans la zone de distanciation, le boid s'en éloigne, s'il est dans la zone d'alignement, le boid l'imite, et s'il est dans la zone d'attraction, il s'en rapproche. On ajoute également à la simulation un angle mort, dans lequel le boid ne peut pas percevoir ses voisins.

On note θ_v l'angle de vision d'un boid et $r_c < r_a < r_d$ les distances définissant les trois zones (voir figure 1).

1.2 Principes généraux de l'évolution des boids

On discrétise le temps : la nouvelle position de chaque boid est calculée à partir de l'ancienne au bout d'un temps dt fixé par l'utilisateur. Si on note P la position d'un boid à l'instant t, $\vec{v}$ son vecteur vitesse, et P' sa position à l'instant $t + dt$, on a donc

$$\overrightarrow{PP'} = dt \cdot \vec{v} \tag{1}$$

On suppose pour simplifier que tous les boids avancent à une même vitesse constante, autrement dit $\|\vec{v}\| = v_0$ ne dépend ni du boid ni de t. Le seul paramètre qui évolue est l'orientation du boid, autrement dit l'angle formé par le vecteur $\vec{v}$ avec l'axe des abscisses.

Les trois règles auxquelles obéit le boid permettent de calculer au temps t un vecteur vistesse $\overrightarrow{v_{\text{but}}}$ que désire prendre le boid au temps $t + dt$. Cependant, un boid ne peut pas changer trop brutalement de direction, et il doit respecter une vitesse rotation maximale $\dot{\omega}_{\text{max}}$ (cf figure 2) :

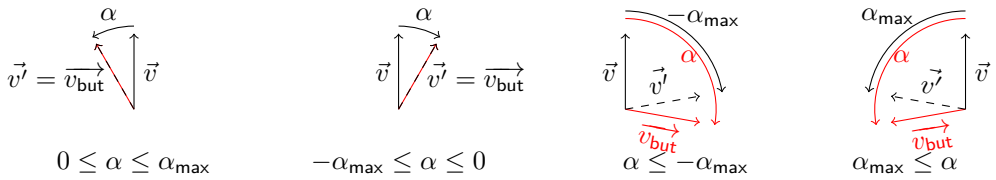

FIGURE 2 – Plafonnement de la rotation d'un boids : l'angle entre le vecteur vitesse $\vec{v}$ au temps t et le vecteur vitesse $\vec{v'}$ au temps $t + dt$ est au plus $\alpha_{\text{max}} = \dot{\omega}_{\text{max}} \times dt$ en valeur absolue.

— si l'angle entre $\vec{v}$ et $\overrightarrow{v_{\text{but}}}$ est plus petit (en valeur absolue) que $\dot{\omega}_{\text{max}} \times dt$, alors le nouveau vecteur vitesse du boid au temps $t + dt$ est $\overrightarrow{v_{\text{but}}}$
— sinon, c'est le vecteur $\vec{v'}$ formant un angle $\pm\dot{\omega}_{\text{max}} \times dt$ avec $\vec{v}$ de même signe que l'angle entre $\overrightarrow{v_{\text{but}}}$ et $\vec{v}$.

1.3 Les règles d'évolution d'un boid

Pour calculer le vecteur $\overrightarrow{v_{\text{but}}}$, on procède comme suit :

1. s'il y a des boids présents dans la zone de distanciation, le vecteur vitesse désiré est le vecteur permettant de s'éloigner le plus rapidement possible du barycentre de ces boids. Autrement dit, si P est la position du boid pour lequel on veut calculer $\overrightarrow{v_{\text{but}}}$, et si $P_1, \ldots, P_n$ sont les positions des autres boids dans la zone distanciation, on a

$$\overrightarrow{v_{\text{but}}} = v_0 \frac{\vec{w_d}}{\|\vec{w_d}\|} \tag{2}$$

où

$$\vec{w_d} = -\frac{1}{n} \sum_{i=1}^{n} \overrightarrow{PP_i} \tag{3}$$

en particulier, dans ce cas, les boids potentiellement présents dans les zones d'alignement et de cohésion n'ont aucune influence sur $\overrightarrow{v_{\text{but}}}$.

2. sinon, s'il n'y a pas de boids dans la zone de distanciation, mais s'il y en a dans les zones d'alignement et de cohésion, $\overrightarrow{v_{\text{but}}}$ est la moyenne (renormalisée) de deux vecteurs $\overrightarrow{w_a}$ et $\overrightarrow{w_c}$ calculés en fonction des populations de boids respectivement dans les zones d'alignement et de cohésion

$$\overrightarrow{v_{\text{but}}} = v_0 \left(\frac{\overrightarrow{w_a}}{2\|\overrightarrow{w_a}\|} + \frac{\overrightarrow{w_c}}{2\|\overrightarrow{w_c}\|} \right) \tag{4}$$

où $\overrightarrow{w_a}$ est la vitesse moyenne des boids de la zone d'alignement, et $\overrightarrow{w_c}$ pointe vers le barycentre des boids de la zone de cohésion. Plus précisément, si $\overrightarrow{v_1}, \ldots, \overrightarrow{v_m}$ sont les vecteurs vitesses des boids de la zone d'alignement, et si $P_1, \ldots, P_s$ sont les positions des boids de la zone de cohésion, on a

$$\overrightarrow{w_a} = \frac{1}{m} \sum_{i=1}^{m} \overrightarrow{v_i} \tag{5}$$

et

$$\overrightarrow{w_c} = \frac{1}{s} \sum_{i=1}^{s} \overrightarrow{PP_i} \tag{6}$$

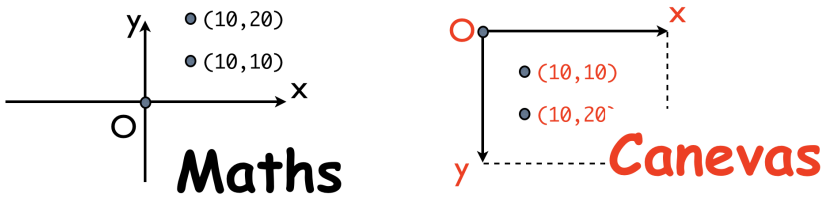

FIGURE 3 – Coordonnées mathématiques et coordonnées informatiques (par exemple dans un canevas).

3. s'il y a des boids uniquement dans la zone d'alignement (et pas dans la zone de cohésion), $\overrightarrow{v_{\text{but}}} = \overrightarrow{w_a}$
4. de même, s'il y a uniquement des boids dans la zone de cohésion, $\overrightarrow{v_{\text{but}}} = \overrightarrow{w_c}$;
5. enfin, s'il n'y a aucun boids dans les trois zones, notre boid va avancer au hasard pour chercher des copains : $\overrightarrow{v_{\text{but}}}$ est un vecteur formant un angle $\pm\alpha$ avec $\vec{v}$ tel que $|\alpha| \leq \dot{\omega}_{\text{max}} \times dt$ tiré au hasard avec une loi uniforme.

1.4 Le monde est un tore

Les boids ont une tendance naturelle à partir dans une direction et à s'y tenir : il vont donc très vite sortir de votre fenêtre. Pour que l'animation ne se résume pas rapidement à un écran blanc, on va faire en sorte que les boids qui sortent par un côté de la fenêtre re-rentrent aussitôt par l'autre côté. Le monde est dit *torique*, comme un pneu de vélo ou une bouée dont on verrait l'intégralité de la surface dans la fenêtre.

Un même point sur un tore peut avoir plusieurs coordonnées. Par exemple, le point au centre d'un monde torique de largeur L et hauteur H aura pour coordonnées le couple $(0,0)$, mais aussi le couple (L,H), ou encore $(-L,3H)$. On appelera *coordonnée canonique* la coordonnée l'unique coordonée $(x,y) \in [-\frac{L}{2},\frac{L}{2}[\times[-\frac{H}{2},\frac{H}{2}[$.

Toutes les positions de boids, ainsi que tous les vecteurs seront représentés par leurs coordonnées canoniques ; il faudra donc régulièrement "rendre canonique" les coordonnées calculées.

Il faudra aussi convertir les coordonnées du monde torique (nombres flottants signés) en coordonnées du canevas (nombres entiers positifs ou nuls). On prendra soin de placer le centre du monde torique au centre du canevas et l'axe des y du monde torique orienté vers le haut du canevas (voir Figure 3).

1.5 Quelques rappels utiles de math

Soit $\overrightarrow{u} = (x,y)$ et $\overrightarrow{v} = (x',y')$ deux **vecteurs <u>unitaires</u>** (i.e. $\|\overrightarrow{u}\| = \|\overrightarrow{v}\| = 1$) formant un angle orienté α de $\overrightarrow{u}$ vers $\overrightarrow{v}$. Dés qu'on fait de la trigonométrie, il est implicite que **les angles sont mesurés en radian**, et non en degrés, donc on suppose dans la suite de ce rappel de math que $\alpha \in [-\pi,\pi]$ est un angle mesuré en radian. Un angle positif indique une rotation dans le sens inverse des aiguilles d'une montre.

On rappelle que le produit scalaire $\overrightarrow{u}.\overrightarrow{v}$ et que le produit en croix $[\overrightarrow{u},\overrightarrow{v}]$ vérifient les identités suivantes (valables seulement pour des vecteurs unitaires) :

$$\overrightarrow{u}.\overrightarrow{v} \quad = \quad xx' + yy' \quad = \quad \cos(\alpha) \qquad \text{et} \qquad [\overrightarrow{u},\overrightarrow{v}] \quad = \quad xy' - x'y \quad = \quad \sin(\alpha) \tag{7}$$

On peut par ailleurs déduire les coordonnées de $\overrightarrow{v}$ de celles de $\overrightarrow{u}$ et de l'angle α comme suit

$$x' \quad = \quad x\cos(\alpha) - y\sin(\alpha) \qquad \text{et} \qquad y' \quad = \quad x\sin(\alpha) + y\cos(\alpha) \tag{8}$$

On rappelle que les fonctions sinus et cosinus admettent des pseudo-inverses arc sinus (`asin`) et arc cosinus (`acos`). Pour $\alpha \in [-\pi, \pi]$, $\sin(\alpha)$ et α sont de même signe. Enfin, la fonction cosinus est une fonction décroissante sur $[0, \pi]$ et paire ($\cos(\alpha) = \cos(-\alpha)$). On a donc l'équivalence, pour tout $\alpha \in [-\pi, \pi]$ et $\beta \in [0, \pi]$,

$$|\alpha| \leq \beta \qquad \text{ssi} \qquad \cos(\alpha) \geq \cos(\beta). \tag{9}$$

1.6 Quelques conseils pour optimiser le code

Pour pouvoir simuler correctement un grand nombre de boids avec une fréquence d'échantillonage élevée, il est nécessaire d'optimiser le code pour réduire le temps pris par les calculs de mises à jour des boids. Il y a au moins deux pistes à explorer pour optimiser le code.

Tout d'abord, certaines fonctions mathématiques prennent plus de temps que d'autres. Les additions et multiplications sont très rapides, mais les calculs de racine carrée et de fonctions trigonométriques sont plus lents. Il faut donc, lorsque c'est possible, éviter de calculer une racine carrée (par exemple, en raisonnant sur la norme au carré d'un vecteur plutôt que sur sa norme), et éviter les calculs de fonction trigonométrique. Avec un peu d'astuce, vous vous passerez des fonctions arc sinus et arc cosinus...

Ensuite, lors de la mise à jour des boids, il vous faudra déterminer pour chacun d'entre eux quels sont les boids voisins qui peuvent l'influencer. Une approche naïve consiste à parcourir tous les couples de boids et à déterminer pour chaque couple si le premier boids du couple a une influence sur le second. Cette approche est quadratique en le nombre de boids. D'autres solutions plus efficaces existent. Par exemple, il est possible de définir un quadrillage en « zones rectangulaires » du monde, et pour chaque boid restreindre la recherche de ses boids influents à la zone rectangulaire dans laquelle se trouve le boid et aux zones immédiatement voisines.

2 Le projet et les fonctionnalités à implémenter

Le but du projet est de réaliser en équipe un programme permettant de simuler un troupeau de boids. Vous aurez à rendre plusieurs versions de votre programme correspondant à l'implémentation de différentes fonctionnalités. Les fonctionnalités minimales de la version de base vous sont spécifiées. Pour les fonctionnalités avancées, vous pouvez choisir dans la liste des suggestions proposées ou définir vous-mêmes d'autres fonctionnalités. Chaque participant du projet devra implémenter au moins une fonctionnalité avancée tout seul.

2.1 La version de base

La version de base comporte les fonctionnalités suivantes

- possibilité d'ajouter un à un des boids à la simulation, en les insérant à une position aléatoire.
- mise en pause et reprise de la simulation. Lorsque la simulation est en pause, on ne peut pas ajouter de boids
- remise à zéro : tous les boids sont effacés.
- redimensionnement de la fenêtre : cette fonctionnalité n'est activée que lorsque la simulation n'a pas commencé, au lancement du programme ou après une remise à zéro.
- les boids changent de couleur suivant la façon dont est calculé leur vecteur $\overrightarrow{v}_{but}$, autrement dit selon la présence ou non d'autres boids dans les différentes zones
- possibilité de changer les paramètres de la simulation :
  - θ_v : l'angle de vue d'un boid (en degrés),
  - r_d, r_a, r_c : les rayons des trois zones (en pixels),
  - v_0 la vitesse de déplacement d'un boid (en pixels par seconde),
  - $\dot{\omega}_{\text{max}}$: la vitesse de rotation maximale d'un boid (en degrés par secondes),

— $\frac{1}{dt}$: la fréquence d'échantillonage de la simulation (en nombre d'images par secondes)

2.2 Extensions

Il s'agit d'une liste non exhaustive, rangée plus ou moins par difficulté croissante.

— ajout et suppression de boids à la souris en cliquant sur le canevas
— boids fatigués : « atterissage » (mise en une pause d'un boid) au bout d'un certain temps de vol
— ajout d'autres agents obéissant à des règles différentes : obstacles immobiles, proies immobiles, proies mobiles sans fuite, proies mobiles avec fuite, etc.
— les boids courent après la souris, ou l'évitent, ou tout autre « effet » de la souris sur les boids
— enrichissement artistique de la simulation (véritables animaux avec mouvements décomposés, scène de fond, etc)
— optimisation du code pour pouvoir simuler à très haute fréquence avec un grand nombre de boids (voir Section 1.6)
— vent ou courant marin « du monde réel » : en vous basant sur des données météorologiques en ligne (par exemple, la direction et la force du vent sur le port de Nice au moment présent) ajoutez une force de déviation.
— zoom avant/arrière sur certaines zones du monde
— monde 3D

2.3 Contraintes à respecter sur le contenu du code

— le projet est entièrement programmé en racket (voir conditions d'évaluations)
— le projet doit mettre en oeuvre une interface graphique basée sur la librairie standard de racket
— le projet doit inclure un module `boids.rkt` exportant une classe `boid%`, et doit donc mettre en oeuvre la programmation orientée objets Racket (« *objets hards* », dans la terminologie de Jean-Paul Roy).
— le projet doit inclure un module `vectors.rkt` basé sur celui fourni en annexe de ce sujet, en gardant les mêmes noms de fonctions.

3 Évaluation du projet

3.1 Les exécutables

Votre projet sera évalué à partir des *exécutables* que vous fournirez. On appelle exécutable un fichier `.rkt` qui s'exécute et qui réalise correctement une tâche donnée. Pour lancer un exécutable, on l'ouvre avec DrRacket et on appuie sur le bouton Exécuter. Le lancement doit fonctionner **sans avoir à saisir du code supplémentaire**.

Les exécutables ont pour nom `executable.rkt`. Chaque exécutable est rangé avec les modules dont il a besoin dans un sous-répertoire à part. Si vous rendez 4 exécutables, vous avez donc 4 sous-répertoires.

Typiquement, il y aura un exécutable pour la version de base, puis un exécutable pour chacune des extensions considérées. Il peut aussi y avoir des exécutables qui font coexister plusieurs extensions, ou qui montrent différentes étapes dans l'avancement du projet.

3.2 La documentation

La documentation est le point de départ de l'évaluation, il est important de la soigner. C'est un fichier (doc.txt ou `doc.pdf`) à rédiger en fin de projet par tous les participants qui contient obligatoirement les informations

suivantes :

1. description des exécutables remis, des fonctionnalités implémentées dans chacun, et des problèmes connus pour chacun
2. pour chaque participant, bilan personnel écrit par le participant lui-même du déroulement du projet permettant de répondre aux questions suivantes :
   — quelle a été l'implication du participant dans la rédaction de telle ou telle partie de code ? dans sa relecture ? dans son test ?
   — quelles ont été les difficultés rencontrées ? si certaines n'ont pas pu être résolues, expliquer pourquoi.

3.3 Critères d'évaluation du code

La clarté, la concision et la modularité du code seront des éléments importants dans l'appréciation du projet. Un code clair est un code aéré, indenté (utilisez `Ctrl+I`), commenté, dont le choix des noms de fonctions et de variables répond à une certaine logique systématique. Chaque définition de fonction et chaque variable globale doit être accompagnée d'un commentaire précisant ce que sont les arguments et ce que fait ou renvoie la fonction. Les identifiants des variables d'une fonction doivent aussi être choisis en appliquant des conventions les plus systématiques possibles. Par exemple, on pourra, en suivant les conventions classiques en Racket, utiliser – pour séparer systématiquement les mots d'un identifiant, faire terminer systématiquement un identifiant de fonction par `!` pour indiquer qu'elle modifie son argument principal par effet de bord, etc. L'identifiant d'une variable indiquera ce qu'elle contient : une variable qui contient un graphe s'appelera `graph`, `graph1`, `graph2`, etc. En cas de travail en équipe, les membres de l'équipe se mettront d'accord sur les conventions de choix des identifiants.

Un code concis est un code qui ne contient pas de redondances inutiles, avec des fonctions dépassant rarement la dizaine de lignes.

Un code bien structuré en modules est un code dans lequel le rôle et la limite de chaque module est claire. Lorsqu'on cherche une fonction, il ne doit pas y avoir d'ambiguïté sur le module dans lequel il faut la chercher. Les dépendances entre modules doivent être pensées pour pouvoir modifier aisément un module sans avoir à changer les autres significativement, de façon à pouvoir travailler aisément en équipe.

3.4 Travail en équipe

Vous devez faire ce projet **en binôme ou en trinôme**. Un travail en solo ou à plus de trois est interdit. Le découpage du travail doit être clair dans l'équipe. Un découpage du travail assez classique en binôme consiste à laisser une personne écrire un module, et l'autre un banc de tests pour ce module (ou avoir un rôle de relecteur si le module est assez court).

Chaque personne de l'équipe devra réaliser seul au moins une extension du projet de base, à choisir parmi les extensions proposées ou à inventer.

Références

[Rey87] Craig W. Reynolds. Flocks, herds and schools : A distributed behavioral model. *ACM SIGGRAPH Computer Graphics*, 21(4) :25–34, 1987.

A Modèle à suivre pour le module `vectors.rkt`

```
#lang racket/gui

(provide
 vect x-coord y-coord sqr-norm norm extern-prod scalar-prod cross-prod
 unitary vect-sum vect-diff vect-mean
 torify angle-vects rotate-vect
 math->canevas canevas->math)

(define (vect x y) ; number * number -> vecteur
  ; construit un "vecteur" 2D a partir de ses coordonnees
  ; a vous de choisir comment vous voulez representer un vecteur...
  ; et a rendre se choix le plus facile a changer par la suite (abstraction!)
  (error "not implemented"))

(define (x-coord v) ; vecteur -> number
  ; renvoie la coordonnee des x d'un vecteur donne
  (error "not implemented"))

(define (y-coord v) ; vecteur -> number
  ; renvoie la coordonnee des x d'un vecteur donne
  (error "not implemented"))

(define (sqr-norm v) ; vecteur -> number
  ; la norme au carre d'un vecteur
  (error "not implemented"))

(define (norm v) ; vecteur -> number
  ; la norme d'un vecteur
  (error "not implemented"))

(define (extern-prod k v) ; number * vecteur -> vecteur
  ; le produit externe (homothetie)
  (error "not implemented"))

(define (scalar-prod v1 v2) ; vecteur * vecteur -> number
  ; le produit scalaire
  (error "not implemented"))

(define (cross-prod v1 v2) ; vecteur * vecteur -> number
  ; le produit en croix
  (error "not implemented"))

(define (unitary v) ; vecteur -> vecteur
  ; le vecteur unitaire (de norme 1) associe a v
  (error "not implemented"))

(define (vect-sum . L) ; vecteur * ... * vecteur -> vecteur
  ; la somme de vecteurs
  (error "not implemented"))
```

```
50
51 (define (vect-diff v1 v2) ; vecteur * vecteur -> vecteur
52    ; la difference entre deux vecteurs
53   (error "not implemented"))
54
55 (define (vect-mean L) ; vecteur list -> vecteur
56   ; le barycentre (ou moyenne) d'une liste de vecteurs
57   ; renvoie une erreur si la liste L est vide
58   (error "not implemented"))
59
60 (define (torify v L H) ; vecteur -> vecteur
61  ; renvoie le vecteur en coordonnees canoniques
62  ; en supposant un tore de largeur L et de hauteur H
63   (error "not implemented"))
64
65
66 (define (angle-vects v1 v2) ; vecteur * vecteur -> number
67   ; renvoie l'angle dans [-pi,pi] entre v1 et v2
68   ; ATTENTION, pour un code optimise, il peut etre utile de ne
69   ; pas utiliser cette fonction
70   (error "not implemented"))
71
72
73 (define (rotate-vect v theta) ; vecteur * number -> vecteur
74   ; renvoie le vecteur v apres rotation de l'angle theta dans [-pi,pi]
75   (error "not implemented"))
76
77 ;; RAPPEL
78 ;; coordonnees maths : nombres flottants signes, (0,0) est au centre du canevas
79 ;; coordonnees canevas : nombres entiers positifs, (0,0) est en haut a gauche
80 ;; du canevas
81
82 (define (math->canevas v L H) ; vecteur -> vecteur
83   ; passe des coordonnees mathematiques aux coordonnees canevas
84   (error "not implemented"))
85
86 (define (canevas->math v L H) ; vecteur -> vecteur
87   ; passe des coordonnees canevas aux coordonnees mathematiques
88   (error "not implemented"))
```

vectors.rkt

B Modèle possible pour le module `boids.rkt`

```
1 #lang racket
2
3 (require "vectors.rkt" "params.rkt")
4
5 ;; note : ici le module params.rkt designe un autre module a programmer,
6 ;; qui permet de determiner les parametres de la
7 ;; simulation (dt, v0, etc...)
```

```
8
9  (provide boid% draw-boids update-boids remove-all-boids)
10
11
12 ;; LA LISTE DES BOIDS CREES
13 (define BOIDS '())
14
15 ;; LA CLASSE DES BOIDS
16 (define boid%
17   (class object%
18
19     (init-field
20      ; a completer)
21
22     (define/public (advance) ; avance le boid selon son vecteur vitesse et dt
23       (error "not implemented"))
24
25     (define/public (draw dc)  ; dessine le boid en utilisant le peintre dc
26       (error "not implemented"))
27
28     (define/public (can-see? boid) ; renvoie vrai si peut voir boid
29
30     (define/public (desired-speed) ; calcule le vecteur vitesse vbut
31
32       ;; autres methodes ... a completer...
33
34     (set! BOIDS (cons this BOIDS))
35     (super-new)))
36
37
38 ; LA FONCTION DE DESSIN
39 (define (draw-boids dc) ; rajoute les boids sur le canevas
40   (error "not implemented"))
41
42
43 ; LA FONCTION D'ACTUALISATION
44 (define (update-boids) ; effectue un pas de la simulation
45   (error "not implemented"))
46
47 ; LA FONCTION RESET
48 (define (remove-all-boids) (set! BOIDS empty))
```

boids.rkt