Documentation : Projet PFA

Groupe :
MARTIN D'ESCRIENNE Yann
GOULOT Thomas
HIRTH DAUMAS Jeremy

Exécutable remis :

Le projet de base :

Le projet de base possède 4 modules qui sont les suivants :

- vectors.rkt
- boids.rkt
- params.rkt
- canvas.rkt (renommé executable.rkt)

Cette version possède les fonctionnalité demandées :

- 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
- les boids changent de couleur suivant la façon dont est calculé leur vecteur vbut, autrement dit selon la présence ou non d'autres boids dans les différentes zones
- possibilité de changer les paramètres de la simulation.

Vectors.rkt :
Ce module exporte de nombreuse fonctions qui effectuent divers calculs sur les vecteurs (que l'on a choisis de représenter par une liste scheme à deux éléments). Par exemple *vect-mean* qui renvoi le barycentre d'une liste de vecteur, ou bien *cross-prod* le déterminant de deux vecteurs (appelé produit en croix). Mais *vectors.rkt* contient aussi des fonctions diverses comme *torify* ou *maths->canvas*.

Params.rkt :
Ce module a pour fonction de créer un panel contenant des sliders pour modifier les paramètres de la simulation (*v0, rd, ra, rc, Ov,* ….). On y fixe également les valeurs par défauts.

Boids.rkt :
Ce module exporte la classe *boid%*, une classe comportant des méthodes pour l'évolution du boid dans la simulation. Elle exporte également la liste *BOIDS* de tout les boids et les fonctions *draw-boids* et *update-boids* permettant d'accéder a certaine méthodes tout les éléments de *BOIDS*.

Canvas.rkt :
Ce module est le module final utilisant tout les précédents, il affiche le canevas et un thread pour rafraîchir automatiquement celui-ci (*counter*), il affiche également le panel avec les boutons pour rajouter un boid, les supprimer et mettre en pause ou en play.

L'extension Taille :

L'extension Taille possède 4 modules qui sont les suivants :

- vectors.rkt
- boids.rkt
- params.rkt
- canvas.rkt (renommé executable.rkt)

Cette version implémente la fonctionnalité suivante :

- elle ajoute comme paramètres dans un slider de params,rkt la taille des boids, modifiable même quand la simulation est en cours.

Partie personnelle de Yann MARTIN D'ESCRIENNE.

Je me suis occupé de la programmation de boid.rkt en son intégralité, d'une partie de canvas.rkt. Je me chargeais aussi de la programmation des fonctions les plus durs (ou le reste du groupe bloquait) dans tout les autres modules. Par exemple torify, maths->canvas, aide pour le thread...
Les phases de test se faisait en groupe ou chacun, après avoir trouvé la source de l'erreur, cherchait une solution dans la documentation racket ou par des modifications de la fonction concernée.
Bien souvent je m'occupait de fixer les erreurs. J'aborde celles que j'ai fixés dans la suite.

Concernant vectors.rkt :

La plupart des fonctions de ce module sont simple à coder car il suffit de recopier la formule mathématique.
La fonction *angle-vects* après des tests à la fin du projet eu besoin de modification, comme le cas ou l'angle entre v1 et v2 est 0 ou pi , car *crossprod* renvoie alors également 0 (car vecteurs colinéaires) et une division par 0 avait alors lieu.
Un autre problème plus subtil était que l'erreur « abs contract violation expected : real ? Givent 0+2,10....e-008i » apparaissait parfois lorsque les boids se déplaçaient sur le canevas. Nous avons supposé que cela était un problème de précision et que le nombre était complexe. J'ai donc gérer le problème avec une vérification (*real?*) dans la fonctions (nous renvoyons 0 si ce n'est pas un réel),

Néanmoins la fonction *torify* fut dure à implémenter, tout d'abord avec un approche par des modulos (selon la hauteur et la largeur du canevas), la sortie par un coté ramenait le boid au centre du repère et non à l'extrémité opposée. Ainsi j'ai repris l'idée du modulo à ramener une valeurs dans un intervalle donné par additions ou soustractions d'une constante selon que la valeur soit positive ou négative (le repère étant centré en 0).

Concernant params.rkt :

Aucune difficulté pour écrire ce module, il suffit juste d'exporter les valeurs avec le *provide*.

Concernant boids.rkt :

Ce module est le plus complexe au niveau programmation, notamment *desired-speed* où je m'y suis beaucoup repris .
Mais je souhaite d'abord m'attarder sur *draw* la méthode de la classe *boid%,* Le problème rencontré fut que *draw-polygon* accepte une liste d'objet *point%* mais pas sous la forme « '(point1 point2 ..) ». Répondant aux critères de la fonction je ne comprenais pas le problème, j'ai néanmoins finis par comprendre qu'il fallait utiliser *(list p1 p2....)* pour le résoudre.

Pour *can-see ?* Le problème fut le même que pour *angle-vects* , celui des nombres complexes , réglé par le *real ?.*Mais aussi le problème de quand le boid donné en argument est l'objet lui même. Il faut alors renvoyez #t directement sans essayer d'appeler *angle-vects* (qui renvoi alors l'erreur d'une division par 0) .

Desired-speed est la fonction qui effectue le plus de calcul, elle est donc celle qui rencontra le plus d'erreur.
Tout d'abord à cause de la fonction *rotate-cap* (les même problème que *angle-vects* et *can-see?*).
Ensuite un oubli de rafraîchir la valeur de vit en fonction de v0 au début de la fonction (vite corrigé).
Enfin l'erreur sûrement la plus stupide venais d'un sous fonction « *zone-List »:* dans le cond , lorsque la zone prenais comme valeur 1, 2 ou 3 , je rappelais le iter en remplaçant *boid* par *boid_pos* (qui contient alors la position du dit boid). Ce qui au deuxième appel me renvoyais l'erreur «target is not an object, target : '(real real)», en effet dans le let boid_pos envoyez donc un *send* à *boid* devenu la position du boid. Cette inattention au niveau du nom de l'argument à donner à la récursion m'a valu un sacré moment de recherche sans comprendre le problème.
Finalement un problème non résolu mais contourner fut celui pour changer de couleur le boid en fonction des autres qui l'entourent. En effet à la base *boid-brush* été créé directement dans l'*init-field*, je changeais alors la couleur par l'appel d'un (send boid-brush set-color « green ») où la couleur était celle désirée. Mais cela me renvoyais l'erreur « boid-brush object is locked ». Je n'ai pas réussis à résoudre cette erreur, Nous avons donc décider de nous y prendre autrement , en créant tout simplement 4 *brush%* et en changeant boid-brush avec un *set !* Et non un *send.*

Concernant canvas.rkt (renommé executable.rkt) :

Un des problèmes rencontrés fut la trop petite taille de la police dans les boutons, en cherchant dans la documentation officielle racket, il fut résolu en créant une *font%* appliquée à chaque *button%*
Un autre problème fut de créer le thread et ce qu'il devrais être dedans. Nous avions tout d'abord tenter de le mettre dans le *paint-callback* mais cela entraînait une erreur. Ensuite de supprimer le *paint-callback* et mettre les fonctions de celui-ci dans le thread, mais il ne peux prendre qu'une fonction sans arguments. Nous avons donc finis par trouver la solution en remarquant que (send a-canvas refresh-now) entraînait un appel de *paint-callback* et que cela ne nécessitait aucun arguments.

Enfin le problème qui n'est toujours pas résolu, celui du redimensionnement de la fenêtre que lorsque *BOIDS* est vide. Tout d'abord j'ai mit une condition lors de la création du *button%* mais cela ne s'effectue que dans sa création..
J'ai ensuite essayer de placer un (send main-canvas min-width (L)) et (send main-canvas stretchable-width #f) (repesctivement stretchable-height et (H)) dans le *call-back* du *BOUTON+1 ,*car dès que l'on appuie sur ce bouton on ne peux plus redimensionner le canevas mais il doit garder la taille qu'il avais avant que le premier boid apparaisse.

De même dans le *call-back* de *BOUTON-STOP* j'ai repris les même commandes mais avec cette fois #t , car dès qu'aucun boid n'est présent sur le canevas on peut le redimensionner.
Néanmoins dès qu'on appuie sur le bouton +1 ou le bouton stop , la fenêtre s'agrandit d'un cran et ne reste pas à sa taille avant l'appuie du bouton.. Je n'ai pas réussis à trouver une solution à ce problème.

Partie personnelle de Thomas GOULOT.

Durant ce projet, j'ai été impliqué dans une partie de la classe boid, dans le canvas, et dans l'animation.

Dans la classe boid, je me suis chargé de rédiger la fonction (draw dc) qui est la fonction affichant un boid sur la fenêtre. Une des difficultés durant la rédaction de celle-ci était de trouver un moyen de remplir d'une couleur l'intérieur du boid, ce que je n'arrivais pas à faire avec seulement des draw-line. Mais ce problème s'est résolu avec la fonction draw-polygon qui en plus de permettre de dessiner la forme souhaitée, permet de remplir l'intérieur. Toutefois une autre difficulté était de trouver la bonne rédaction de la fonction sachant que je n'avais pas trouvé d'exemple et la doc racket sur celle-ci n'en donne pas. Après plusieurs essais et l'aide des autres membres du groupes, on a trouvé la bonne manière de rédiger. J'ai aussi rédigé (draw-boids dc) vu qu'il applique tout simplement la fonction (draw dc) à tous les boids dans la liste BOIDS avec un map.

Dans canvas.rkt, j'ai rédigé les boutons "Stop" et "+1" sans rencontrer de réelle difficulté, mais aussi l'animation de ce projet. L'animation avec le thread a posé certaines difficultés, tout d'abord par le fait qu'il s'active dès le début sans avoir à l'appeler, mais aussi qu'une des étapes de celle-ci, le "refresh", marche d'une autre manière que prévu. Mais j'ai contourné les problèmes en ne mettant pas toutes mes étapes dans le thread. Le refresh rappelle la fenêtre désignée, ici main-canvas, du coup dans le thread j'ai juste inclus dedans cette fonction et le sleep, et j'ai rajouté dans le main-canvas un paint-callback la mise à jour des boids pour les mouvements et la fonction draw-boids. J'ai aussi ajouté un "when" au début de la fonction pour le thread afin qu'il ne s'active que lorsque le bouton est sur "play".

La relecture du code et les tests du code se sont fait en groupe, du coup il n'y a pas vraiment un membre du groupe qui en a fait plus que les autres. On a chacun de notre côté vérifié s'il n'y avait pas d'erreurs dans la rédaction du code pendant la relecture. Et lors des tests on a comparé ensemble si on rencontrait la même erreur pour voir si ça venait bien du projet en lui-même et pas une erreur du côté d'un des membres, puis on réfléchissait ensemble sur les moyens de régler les erreurs.

Partie personnelle de Jeremy HIRTH DAUMAS.

Lorsque que nous avons commencé le projet, nous nous somme attribuer différentes tâches et nous nous somme concerter pour choisir la meilleure représentation d'un vecteur pour les coordonnées des boids, nous avons choisi de représenter un vecteur par une liste pour une question de simplicité. Pour ma part j'ai essentiellement contribuer à la réalisation de *vector.rkt* et de *params.rkt*. Il y a eu plusieurs difficultés rencontrés dans vectors.rkt :

- Tout d'abord pour la fonction *(unitary v)* qui renvoie le vecteur unitaire de v, ou je ne savais pas ce que signifiait ce vecteur. Après une recherche sur internet j'ai pu trouver qu'il s'agissait du même vecteur mais de norme 1. Et j'ai pu trouver que pour calculer un vecteur unitaire associé à un vecteur donné il faut diviser ce vecteur par sa norme.
- Puis pour la fonction *(vect-sum . L)* qui renvoie la somme des vecteurs données en paramètre (fonction d'arité variable). Pour réaliser cette fonction, il fallait utiliser les fonction d'ordre supérieur *apply* et *map*.
- Et enfin pour la fonction *(torify v L H)* qui renvoie le vecteur en coordonnées canoniques, ou nous avons eu beaucoup de difficultés à la programmer. En effet notre idée de départ était de programmer cette fonction en utilisant le modulo, mais après de multiple essais nous ne sommes toujours pas parvenue à la programmer en utilisant cette méthode car il y avait toujours une erreur dans la sous fonction *(torify-val x L)* pour x = 2 et L = 4. Mais nous avons par la suite trouvé une solution par récurrence sans utiliser le modulo.

En ce qui concerne le module *params.rkt* qui permet de modifier les paramètres de la simulation, il n'y a pas eu de problèmes particulier. J'ai commencé par définir les variables : rd, rc, etc… Puis j'ai définit la fenêtre des paramètres, le panneaux horizontale contenant les deux panneaux verticale gauche et droit, puis le titre des variables avec leurs slider associer.

ET enfin pour ce qui concerne la relecture du code et pour les test nous les avons effectués en groupe. Pour les premier test nous avons eu quelques erreurs comme pour l'angle de vue d'un boid ou parfois la valeur devenait un nombre complexe car racket ne pouvais plus l'afficher en réel, nous avons donc rajouté une condition dans la fonction *can-see?* du module *boids.rkt* qui vérifiait que le nombre était bien un réel.

Du point de vue personnel ce projet a été une bonne expérience par son coté assez complexe car il m'a permis de m'investir totalement pour comprendre certaine erreurs en faisant un grand nombre de recherche internet.