INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

BANDE ORGANISÉE

MAILLARD Swan

SENAME Victor

LAALOUJ Adam

TOUIL Imad

Table des matières

1. Cahier des charges

Notre projet « Bande Organisée » a pour but de représenter le comportement d'espèces animales grégaires en simulant leur comportement en fonction de différents paramètres et en fonction de leur environnement. Notre projet s'inspire du programme Boids de Craig W. Reynolds qui simule le comportement d'une nuée d'oiseaux en vol.

Objectifs :

- Créer une simulation de comportement grégaire avec différents paramètres comme le nombre, la présence d'obstacles ou de prédateurs, ….
- Faire une interface permettant de modifier les paramètres et de visualiser les déplacements en temps réel
- Modéliser des déplacements à l'aide de lois mathématiques simples : cohésion (l'individu doit se diriger vers le centre de la nuée), séparation (il se repoussent les uns les autres), alignement (ils alignent leur vecteur vitesse sur celui de leurs voisins) et les parois repoussent les individus.

Fonctionnalités :

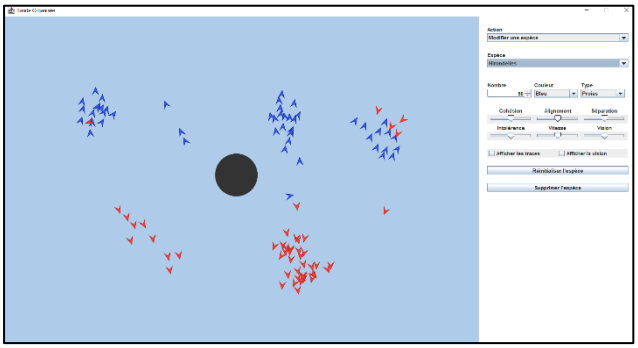
- Respecter les paramètres de base permettant la modélisation (récupérer les données permettant d'actualiser le dessin en fonction des paramètres)
- Pouvoir modifier les paramètres en direct pour observer l'évolution des comportements en fonction des paramètres.
- Pouvoir inclure des éléments perturbateurs qui changent le comportement des individus (espèces prédatrices, obstacles)
- Pouvoir créer plusieurs populations d'individus avec différents comportements qui puissent interagir entre elles (une espèce peut être plus ou moins intolérante)
- Avoir deux zones bien distinctes dans l'interface : simulation et panneau de contrôle
- Pouvoir voire la trajectoire des individus (traces)

Contraintes :

- Capacité de l'ordinateur à effectuer un grand nombre de calculs dans un intervalle court (cela limite le nombre d'individus en simulation)
- Modélisation mathématique simple donc pas forcément très réaliste
- Taille de la zone de déplacement des boids limitée
- Détermination des coefficients liés aux différentes forces de façon expérimentale

2. Description de l'IHM

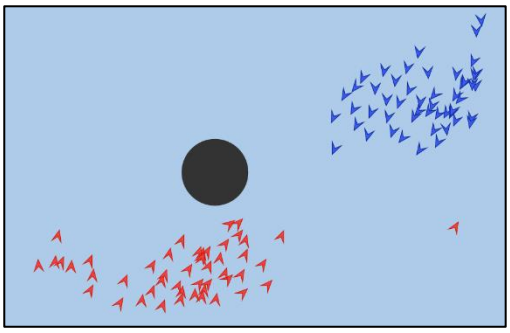
Notre application est composée d'une seule fenêtre séparée en différents JPanel.



AppView : Fenêtre de l'application

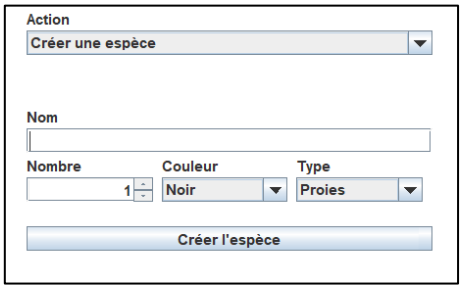
Le JPanel principal s'appelle AppView. Il prend toute la taille de la fenêtre et contient deux autres JPanel :

- SimulationView prend ¾ de la fenêtre. C'est à l'intérieur que va se dérouler la simulation avec l'affichage des boids et des obstacles. Lorsque l'on clique dessus cela fait apparaître un boid ou un obstacle à l'endroit de la souris (selon la configuration dans le panneau de contrôle).

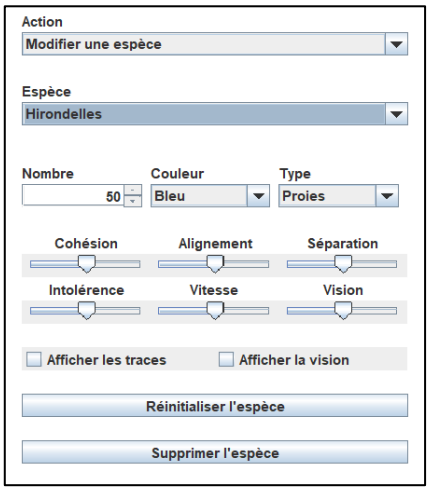


SimulationView : Simulation en cours

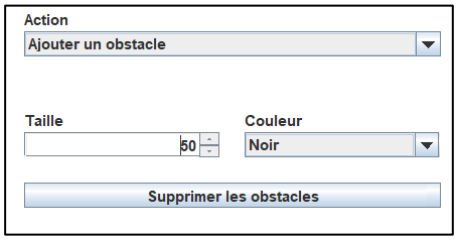
- ControlsView prend ¼ de la fenêtre. Il permet de créer différentes espèces de boids, de contrôler leurs paramètres (nom, nombre de boids, couleur, coefficient des différentes forces, affichage ou non des traces et des champs de vision) et d'ajouter des obstacles (taille et couleur). On peut se déplacer dans le panneau de contrôle à l'aide de listes déroulantes. Le nombre d'espèce est limitée à 5 et le nombre de boids par espèce est limité à 100.



ControlsView : Création d'espèce



ControlsView : Modification d'espèce



ControlsView : Ajout d'obstacle

3. Principes algorithmiques

Les principes algorithmiques les plus importants de notre programme se trouvent dans la classe Boid et permettent de donner la logique du déplacement de chaque boid afin de donner une impression de vol en nuée.

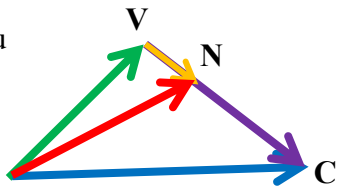
Chaque boid peut être caractérisé par un vecteur position (x ; y), et d'un vecteur vitesse (dx ; dy). À chaque intervalle dt (TIMER_DELAY), nous calculons un vecteur force (Fx ; Fy) qui est la somme de toutes les forces exercées d'après les différentes règles (cohésion, séparation, alignement, etc). D'après le Principe Fondamental de la Dynamique, la somme des forces extérieures égal la masse fois l'accélération, or la masse des boids est négligée et l'accélération correspond à la différence des vitesses entre l'intervalle dt, donc on calcule la nouvelle vitesse : (dx ; dt) = (dx + Fx ; dy + Fy). On limite ensuite la norme de la vitesse entre une valeur minimale et maximale.

De la même manière, la vitesse correspond à la différence de positions entre l'intervalle dt, donc on calcule la nouvelle position : (x ; y) = (x + dx ; y + dy).

Avant de calculer les forces, il faut d'abord trouver les voisins de notre boid (notons B1). Pour cela on parcourt tous les boids de chaque flock (notons B2) et on calcule le vecteur $\overrightarrow{B1B2}$. On peut également connaître l'angle de rotation α de B1 puisqu'il se déplace dans le sens de son vecteur vitesse : $\alpha = \tan^{-1}(\frac{dy}{dx})$ ainsi que l'angle $\widehat{B1B2} = \left\|\tan^{-1}(\frac{\overrightarrow{B1B2}.y}{\overrightarrow{B1B2}.x})\right\|$. On note β l'angle mort du champ de vision de B1. Alors, si $\widehat{B1B2} - \alpha \geq \pi - \frac{\beta}{2}$ et $\widehat{B1B2} - \alpha \leq \pi + \frac{\beta}{2}$, B2 se trouve dans l'angle mort de B1. Finalement, si $\left\|\overrightarrow{B1B2}\right\| \leq Rayon\ du\ champ\ de\ vision$ et B2 ne se trouve pas dans l'angle mort de B1, on ajoute B2 dans la liste de voisins.

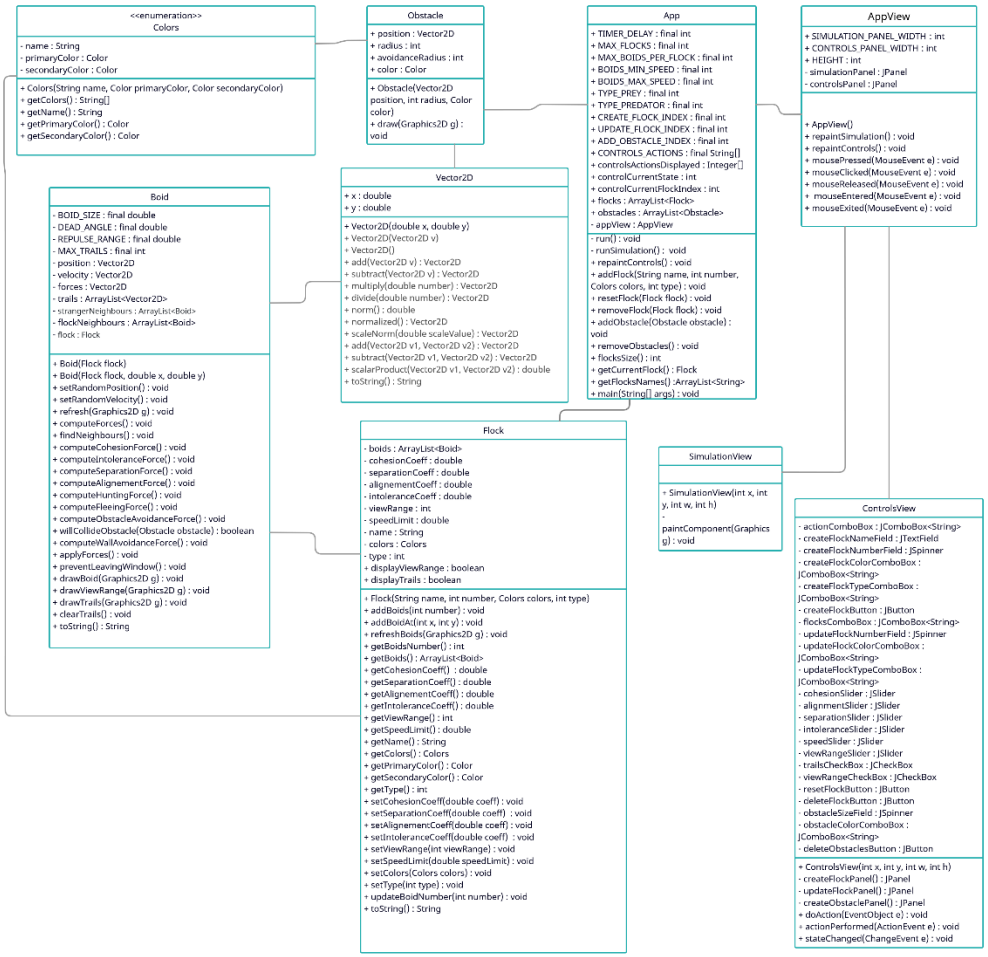
Parmi les forces calculées, la force de cohésion attire le boid vers le centre des boids voisins. Pour cela, on crée un vecteur nul $\vec{C}$. On parcourt donc les voisins de la même espèce en ajoutant leur vecteur positions à $\vec{C}$. S'il y a au moins un voisin (on ne doit pas diviser par zéro) on divise $\vec{C}$ par le nombre de voisins afin d'avoir la moyenne de leur position : cela représente leur « centre ». On veut ensuite ajouter une force dirigée du boid B vers C. Cependant, on ne veut pas que cette force soit trop forte pour ne pas que le boid change de direction trop brutalement.

D'après le schéma ci-contre, notre boid B a pour vitesse $\overrightarrow{BV}$. Il doit se rendre au centre C. Pour que le boid tourne petit à petit en direction de C, on veut que sa nouvelle vitesse soit $\overrightarrow{BN}$ et donc appliquer une force $\overrightarrow{VN}$. On utilise les relations de Chasles pour trouver $\overrightarrow{VN} = \left(\overrightarrow{BC} - \overrightarrow{BV}\right) \times \lambda = \overrightarrow{VC} \times \lambda$ avec λ le coefficient de cohésion.



Tous les calculs de forces utilisent ce principe, seul le calcul de $\overrightarrow{BC}$ utilise des mathématiques plus ou moins compliquées. Le calcul pour éviter les obstacles provient de l'article *Flocking Behaviour*, Paragraphe F.(b) (Voir Bibliographie).

4. Structures de données



La classe App est le squelette de l'application, elle permet de lier la partie algorithmique et la partie interface. Elle contient la plupart des constantes du programme ainsi que la liste des flocks (espèces contenant des boids) et des obstacles. C'est elle qui lance le timer pour recalculer et réafficher la position des boids de façon continue. Elle contient la méthode main pour lancer le programme.

La classe Flock contient les paramètres de l'espèce ainsi que sa liste de boids.

La classe Boid représente un individu. Des méthodes permettent de calculer les forces à lui appliquer, de le déplacer et de le dessiner ainsi que son champ de vision et ses traces si les options correspondantes sont activées.

La classe Obstacle représente un obstacle et peut le dessiner.

La classe AppView correspond à la fenêtre principale de l'application. Elle contient le panneau de simulation et le panneau de contrôle.

La classe SimulationView représente le panneau de simulation. Une méthode parcourt la liste de boids de chaque flock ainsi que les obstacles pour les dessiner.

La classe ControlsView représente le panneau de contrôle. Il gère les différentes actions de l'utilisateur.

La classe Colors est une énumération contenant les différentes couleurs disponibles dans deux teintes : une teinte primaire et secondaire.

La classe Vector2D permet de manipuler des vecteurs à 2 dimensions.

5. Suggestion d'amélioration

Le projet est globalement abouti, mais nous avons pensé à des améliorations qui pourraient être apportées.

Le problème majeur auquel nous nous sommes heurtés est dû à l'inertie des boids. En effet, lorsque l'on applique une force au boid, celle-ci ne doit pas être trop brutale afin d'éviter des changements peu naturels de directions. Ainsi sa vitesse a une certaine inertie et plus le boid avance rapidement, moins il est capable d'éviter les collisions avec les obstacles ou les bords de la fenêtre. Nous n'avons pas trouvé comment régler ce problème de la façon la plus naturelle possible (actuellement, lorsque le boid heurte les bords de la fenêtre, il s'arrête net).

D'autre part, les boids ont dû mal à gérer les obstacles quand il y en a plusieurs qui sont rapprochés et ils restent parfois coincés entre. Cela est dû aux mathématiques utilisées pour contourner les obstacles. Il faudrait davantage se pencher sur l'aspect mathématique pour éviter de tels problèmes.

Une autre amélioration pourrait être de pouvoir afficher les forces appliquées en temps réel sur les boids sous forme de vecteur. Cependant cela pourrait surcharger l'affichage.

Nous pourrions également ajouter des perturbations extérieures tel que du vent soufflant dans une certaine direction.

6. Évolution du projet et contributions

Évolution du projet :

07/03 : Réflexion sur le cahier des charges, la structuration des fichiers, le diagramme UML et le partage des tâches.

14/03 : On commence à coder une IHM pour la base, la classe Vecteur et on ajoute des sphères que l'on fait bouger.

21/03 : Amélioration de l'IHM pour pouvoir créer différentes espèces, ajout des règles de cohésion, de séparation et d'évitement des bords de l'écran.

28/03 : Ajout des règles d'alignement et d'intolérance, création d'une forme personnalisée pour les boids.

04/04 : Ajout des obstacles, la règle pour les éviter ainsi que la possibilité d'afficher les traces des boids.

11/04 : Séparation des fichiers d'IHM (AppView, SimulationView et ControlsView), création de Colors pour gérer les couleurs des boids, ajout de boids « Prédateurs » chassant les boids « Proies ».

Contributions :

Swan : IHM, affichage des boids, gestion des couleurs, ajout des traces des boids.

Victor : Implémentation des règles de cohésion, de séparation, d'alignement et d'intolérance.

Adam : Implémentation du système de position / vitesse / forces, évitements des bords de la fenêtre, implémentation des obstacles.

Imad : Gestion des vecteurs et implémentation du système proies / prédateurs.

Nous avons chacun contribué à environ ¼ du projet.

Bibliographie

- SmarterEveryDay. How Birds Do the Thing – Smarter Every Day 234 [En ligne]. Disponible sur :

  https://www.youtube.com/watch?v=4LWmRuB-uNU (Consulté le 01/05/22)

- C. REYNOLDS. Boids, Background and Update [En ligne]. Disponible sur :

  http://www.red3d.com/cwr/boids/ (Consulté le 01/05/22)

- Auteur inconnu. Flocking Behaviour [En ligne]. Disponible sur :

  http://www2.cs.uregina.ca/~anima/408/Notes/ControllingGroups/Flocking.htm (Consulté le 01/05/22)

- Wikipedia. Boids [En ligne]. Disponible sur :

  https://fr.wikipedia.org/wiki/Boids (Consulté le 01/05/22)

- Auteur inconnu. Implémenter l'algorithme Boids [En ligne]. Disponible sur :

  https://qastack.fr/codegolf/154277/implement-the-boids-algorithm (Consulté le 01/05/22)