Génie Logiciel
Projet GL – *Simulation de Boids*

1 Contexte

Le modèle de **Boids** (Craig Reynolds, 1986) simule des comportements collectifs inspirés du vol en groupe des oiseaux ou du déplacement en banc des poissons. Chaque agent ("boid") ne prend pas de décision globale mais suit des **règles locales simples**, appliquées en fonction de ses voisins proches.

L'objectif est de réaliser une **implémentation orientée objet en C++** du modèle des Boids, avec une **visualisation interactive** via la bibliothèque graphique **SFML**.

2 Les trois règles locales

Chaque boid possède :
— une **position** $\vec{p}$,
— une **vitesse** $\vec{v}$ (limitée par une vitesse maximale $v_{\max}$),
— un **rayon de perception** r définissant quels boids sont considérés comme voisins.
Les forces de comportement sont exprimées comme des vecteurs de correction appliqués à la vitesse :

1. **Cohésion (se rapprocher du groupe)** :
$$\vec{f}_{coh} = \frac{1}{N}\sum_{i=1}^{N}\vec{p}_i - \vec{p}$$
2. **Séparation (éviter les collisions)** :
$$\vec{f}_{sep} = \sum_{i=1}^{N}\frac{\vec{p}-\vec{p}_i}{\|\vec{p}-\vec{p}_i\|^2}$$
3. **Alignement (suivre la direction du groupe)** :
$$\vec{f}_{ali} = \frac{1}{N}\sum_{i=1}^{N}\vec{v}_i - \vec{v}$$

Cas particulier - Aucun voisin : Si un boid n'a aucun voisin dans son rayon de perception, toutes les forces sont nulles. Le boid continue alors simplement à se déplacer selon sa vitesse courante, sans correction.

Mise à jour de la vitesse et de la position

À chaque pas de temps :
$$\vec{v_{t+1}} \leftarrow \vec{v_t} + w_{coh}\vec{f}_{coh} + w_{sep}\vec{f}_{sep} + w_{ali}\vec{f}_{ali}$$
puis normalisation de la vitesse pour respecter $v_{\max}$. Enfin, la position est mise à jour :
$$\vec{p} \leftarrow \vec{p} + \vec{v}\cdot\Delta t$$

3 Paramètres imposés (par défaut)

Votre implémentation devra utiliser les paramètres suivants :
— Nombre de boids : 50
— Taille de la fenêtre SFML : 800×600 pixels
— Forme : chaque boid est un triangle isocèle orienté dans la direction de la vitesse
— Vitesse maximale $v_{\max}$: 4 unités / pas de temps
— Accélération maximale : 0.1 unités / pas de temps^2
— Rayon de perception r : 50 pixels
— Distance minimale de séparation d_{min} : 20 pixels
— Poids des règles :
  — Cohésion $w_{coh} = 0.01$
  — Séparation $w_{sep} = 0.05$
  — Alignement $w_{ali} = 0.125$

4 Interface de contrôle (obligatoire)

Votre programme doit inclure une **interface permettant de modifier en temps réel les paramètres** de la simulation.

Au minimum, l'utilisateur doit pouvoir ajuster :
— le nombre de boids (10-200),
— le rayon de perception r (10-100),
— la distance minimale de séparation d_{min} (5-50),
— les poids $w_{coh}, w_{sep}, w_{ali}$ (0-1).
L'interface peut être réalisée par :
— des touches clavier (augmenter/diminuer une valeur),
— un affichage texte dans la fenêtre SFML (`sf::Text`).
Les modifications doivent être **prises en compte immédiatement**, sans relancer la simulation.

5 Contraintes pédagogiques

1. **Templates** : Implémenter au moins une classe template `Vec2<T>` pour représenter des vecteurs mathématiques 2D.
2. **Surcharge d'opérateurs** : La classe `Vec2<T>` doit inclure la surcharge des opérateurs : `+`, `-`, `*`, `/`, `+=`, `-=`, `*=`, `/=`, `==`, `!=`, et `«`.
3. **Polymorphisme** : Les règles de comportement doivent être représentées par une classe abstraite `Rule`, avec des sous-classes (Cohésion, Séparation, Alignement).
4. **Structures de données** : L'utilisation des conteneurs de la STL (`std::vector`, `std::list`, etc.) est interdite. Implémenter votre propre structure générique (par exemple `DynamicArray<T>` ou `LinkedList<T>`), utilisée pour stocker les boids dans la classe `Flock`.
5. Tout ce qui concerne le jeu sera fait dans le namespace (bd).

6 Cahier des charges

1. **Conception orientée objet** : définir les classes principales (`Boid`, `Flock`, `Simulation`, `Vec2<T>`, `Rule`, `Settings`).
2. **Implémentation de base** : règles principales (cohésion, séparation, alignement), rebonds aux bords, affichage des boids.
3. **Interface de contrôle** : modification des paramètres en temps réel et affichage des valeurs.
4. **Ensuite étendre** : obstacles, prédateur, nouvelles règles.
5. Sauvegarder / charger des configurations.
6. Une interface graphique fonctionnelle et efficace.

7 Livrables

— Code source complet documenté (C++11 ou supérieur, SFML),
— Votre rendu devra obligatoirement compiler en utilisant Visual Studio (Windows) ou un makefile (Linux) **uniquement** (e.g. cela revient à faire un `git clone` et pouvoir compiler sans soucis).
— Rapport court expliquant :
  — La conception orienté objet (diagramme UML recommandé),
  — L'utilisation des templates, surcharge d'opérateurs et polymorphisme,
  — Les choix effectués pour la structure de données maison,
  — L'implémentation de l'interface utilisateur,
  — Les extensions éventuellement développées.